3.1 NETWORK MANAGEMENT STANDARDS

There are several network management standards that are in use today.

Table 3.1 lists four standards, along with a fifth class based on emerging technologies, and their salient points. The first four are the OSI model, the Internet model, TMN, and IEEE LAN/MAN. A detailed treatment of the various standards can be found in [Black, 1995]. The first category in Table 3.1, Open System Interconnection (OSI) management standard, is the standard adopted by the International Standards Organization (ISO). The OSI management protocol standard is Common Management Information Protocol (CMIP). The OSI management protocol has built-in services, Common Management Information Service (CMIS), which specify the basic services needed to perform the various functions. It is the most comprehensive set of specifications and addresses all seven layers. OSI specifications are structured and deal with all seven layers of the OSI Reference Model. The specifications are object oriented and hence managed objects are based on object classes and inheritance rules. Besides specifying the management protocols, CMIP/CMIS also address network management applications. Some of the

Table 3.1 Network Management Standards

STANDARD	SALIENT POINTS
OSI/CMIP	• International standard (ISO/OSI) • Management of data communications network—LAN and WAN • Deals with all seven layers • Most complete • Object oriented • Well structured and layered • Consumes large resource in implementation
SNMP/Internet	• Industry standard (IETF) • Originally intended for management of Internet components, currently adopted for WAN and telecommunication systems • Easy to implement • Most widely implemented
TMN	• International standard (ITU-T) • Management of telecommunications network • Based on OSI network management framework • Addresses both network and administrative aspects of management • eTOM industry standard for business processes for implementing TMN using NGOSS framework
IEEE	• IEEE standards adopted internationally • Addresses LAN and MAN management • Adopts OSI standards significantly • Deals with first two layers of OSI RM
Emerging Technologies	• Web-Based Enterprise Management (WBEM) • Java Management Extension (JMX) • XML-based Network Management • CORBA-based Network Management

major drawbacks of the OSI management standard were that it was complex and that the CMIP stack was large. Although these are no longer impediments to the implementation of the CMIP/CMIS network management, SNMP is the protocol that is extensively deployed.

In contrast to the CMIP protocol, the Simple Network Management Protocol (SNMP) presented in Table 3.1 is truly simple as its name indicates. It started as an industry standard and has since become very much like standard specifications of a standards organization. The Internet Engineering Task Force (IETF) is responsible for all Internet specifications including network management. The managed objects are defined as scalar objects in SNMP. It was primarily intended to manage Internet components, but is now used to manage WAN and telecommunications systems. It is easy to implement and is the most widely implemented network management system today. We will discuss SNMP management in more detail in this book.

The third category in Table 3.1, TMN, is designed to manage the telecommunications network and is oriented toward the needs of telecommunications service providers. TMN is ITU-T (International Telecommunications Union—Telecommunications) standard and is based on OSI CMIP/CMIS specifications. TMN extends the concept of management beyond managing networks and network components. Its specifications address service and business considerations (M3000). Chapter 10 is devoted to the discussion of TMN.

Enhanced Telecommunications Operations Map (eTOM) is a guidebook for business processes in the telecommunications industry. It is an extension of TMN. It is being developed by TeleManagement Forum (TM Forum) as component of NGOSS (New Generation OSS) [Reillly and Creaner, 2005]. The main difference between the TMN and eTOM approaches is that the former has been developed starting from networks and network equipment (bottom up), while eTOM is a top-down approach. The eTOM framework has been incorporated within the TMN framework as a set of standards (M.3050.x).

The IEEE standards for Local Area Network (LAN) and Metropolitan Area Network (MAN) specifications shown in Table 3.1 are only concerned with OSI layers 1 (physical) and 2 (data link). Those specifications are structured similar to OSI specifications. Both OSI/CMIP and Internet/SNMP protocols use IEEE standards for the lower layers. The IEEE 802.x series of specifications define the standards for the various physical media and data link protocols. IEEE 802.1 specifications present overview, architecture, and management. The IEEE 802.2 standard specifies the Logical Link Control (LLC) layer. As we saw in Chapter 1 (Figure 1.14), the LLC layer provides transparency of the various physical media and protocols to the network layer. The others in series are for specific media and protocols. For example, 802.3 specifications are for Ethernet LANs.

The last category in Table 3.1 addresses several emerging management technologies. One of them is based on using Web technology, Web server for the management system and Web browsers for network management stations. In Web-based management, the organization model uses Web server–Web browser architecture. Much of the object-oriented technology, such as hypermedia server, CORBA-oriented transportation, and client–server push-technology influence the Web-based management.

The Web-Based Enterprise Management (WBEM) standard is developed by the Desktop Management Task Force (DMTF). It is based on the Common Information Model (CIM) data model transported using CIM Operations over HTTP.

The Java Management Extension [JMX] is an open Java technology for management. It defines management architecture, application programming interfaces (APIs), and management services under a single umbrella specification. It was developed under Sun Microsystems's JMAPI (Java Management API) initiative.

XML is a meta-markup language standardized by the Worldwide Web Consortium (W3C) for document exchange in the Web. XML-based network management is based on a network management method, which defines management information by XML and the exchange of data for management in the form of an XML document, and it uses an XML document-processing standard method for processing data.

Common Object Request Broker Architecture (CORBA)-based Network Management is an object-oriented client–server model that uses CORBA. The objects are defined using Interface Description Language (IDL) and uses a distributed managed objects architecture.

With the Web-based management system, not only can object-oriented technology be implemented but also the dedicated workstation constraint is removed by the use of a Web browser. However, which object-oriented technology should an IT manager choose? There is no clear-cut answer to this question and different vendors have implemented NMSs using different technologies for different applications.

ISO - International org for standardization

3.2 NETWORK MANAGEMENT MODELS

The OSI network model is an ISO standard and is most complete of all the models. It is structured and it addresses all aspects of management. Figure 3.1 shows an OSI network management architectural model that comprises four models. They are the organization model, the information model, the communication model, and the functional model. Although, the above classification is based on the OSI architectural model, and only parts of it are applicable to other models, it helps us understand the holistic picture of different aspects of network management.

The organization model describes the components of a network management system, their functions, and their interactions. The organization model is defined in ISO 10040 OSI Systems Management Overview. It defines the terms object, agent, and manager.

The OSI information model deals with the structure and the organization of management information. ISO 10165 specifies the Structure of Management Information (SMI) and the information database, management information base (MIB). SMI describes how the management information is structured and MIB deals with the relationship and storage of management information.

The third model in OSI management is the communication model. There are three components to this—management application processes that function in the application layer, layer management between layers, and layer operation, which is within the layer. We will focus on the application processes in this book.

The functional model is the fourth component of OSI management, which deals with the user-oriented requirements of network management. As mentioned in Chapter 1, there are five functional application areas defined in OSI, namely configuration, fault, performance, security, and accounting. These are defined as system management functions in OSI.

As mentioned earlier, only OSI presents the most complete model for network management, while the others either deal with only a subset or are still in the process of development of standards. Although a discussion of OSI management is not part of this book, it is briefly covered in Appendix A for completeness of the subject. OSI deals with all seven networking layers. Besides, as we shall see in Chapter 10, it lends itself to addressing service and business management that are more than just networking. The second standard listed in Table 3.1 is SNMP/Internet standard. IETF does not define architecture for the SNMP management model explicitly. However, it does exist implicitly. The organization, information, and communication models are similar to OSI models. The SNMP network management model addresses the functional model in terms of operations, administration, and security. SNMP-based management is widely used for campus-wide networks, although enterprise-wide networks are also managed by using distributed configurations of SNMP-based network management systems (NMSs). SNMP-based management systems, tools, and applications are addressed in Chapter 9. The third standard listed in Table 3.1 is the TMN, which is based on the OSI model. Thus, the four models apply to TMN. The focus of the TMN standard is towards managing telecommunications networks. As mentioned earlier,

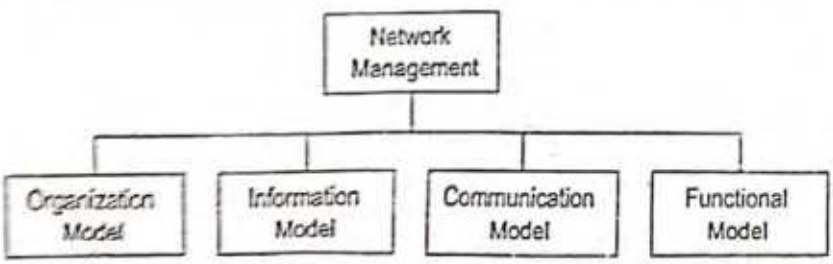


Figure 3.1 OSI Network Management Model

it extends the application functions of OSI further into business and service considerations. Operations systems support service and business management. The fourth standard in Table 3.1 is the IEEE Standard on management and is dedicated to the management of layers 1 and 2 of the OSI Reference Model. It is applicable to LAN and MAN. LAN refers to local area network and MAN (Metropolitan Area Network) refers to metropolitan (intra-city) network. It also addresses standards on broadband network management, which is of great relevance to the current technology. Broadband management is covered in Part IV. Since it deals only with physical and data link layers, it is primarily concerned with the communication model.

In Web-based and object-oriented management, the organization model uses Web server–Web browser architecture. Much of the object-oriented technology, such as hypermedia server, CORBA-oriented transportation, and client–server push-technology are influencing Web-based management. Applications developed under Web-based management could still fall under the OSI functional model. We will now look at each of the models.

3.3 ORGANIZATION MODEL

The organization model describes the components of network management and their relationships. Figure 3.2 shows a representation of a two-tier model. Network objects consist of network elements such as hosts, hubs, bridges, routers, etc. They can be classified into managed and unmanaged objects or elements. The managed elements have a management process running in them called an agent. The unmanaged elements do not have a management process running in them. For example, one can buy a managed or unmanaged hub. Obviously the managed hub has management capability built into it and hence is more expensive than the unmanaged hub, which does not have an agent running in it. The manager communicates with the agent in the managed element.

The manager manages the managed element. As shown in Figure 3.2, there is a database in the manager, but not in the agent. The manager queries and receives management data from the agent, processes them, and stores them in its database. The agent can also send a minimal set of alarm information to the manager unsolicited.

Figure 3.3 presents a three-tier configuration. The intermediate layer acts as both agent and manager. As manager, it collects data from the network elements, processes them, and stores the results in its database. As agent, it transmits information to the top-level manager. For example, an intermediate

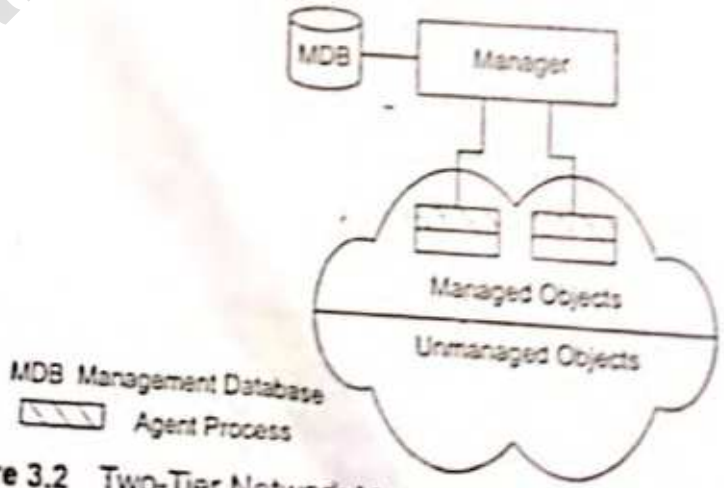


Figure 3.2 Two-Tier Network Management Organization Model

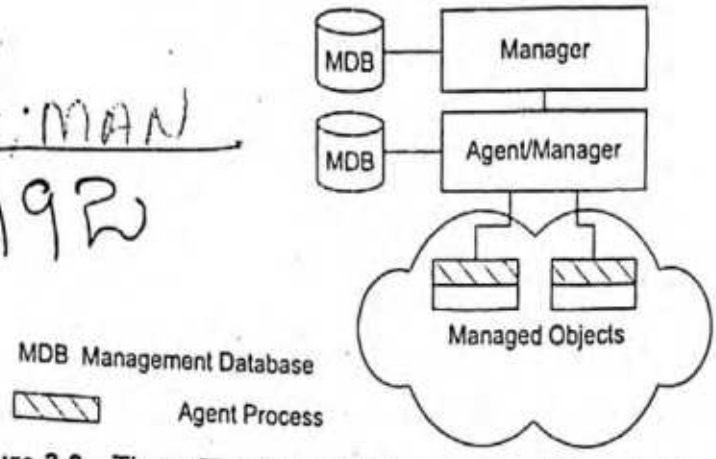


Figure 3.3 Three-Tier Network Management Organization Model

system is used for making statistical measurements on a network and passes the information as needed to the top-level manager. Alternatively, an intermediate NMS could be at a local site of a network and the information is passed on to a remote site.

Network domains can be managed locally; and a global view of the networks can be monitored by a manager of managers (MoM), as shown in Figure 3.4. This configuration uses an enterprise NMS and is applicable to organizations with sites distributed across cities. It is also applicable to a configuration where vendor management systems manage the domains of their respective components, and MoM manages the entire network.

Network management systems can also be configured on a peer-to-peer relationship as shown in Figure 3.5. This is the dumbbell architecture shown in Figure 1.24. We can recognize the similarity between this and the client–server architecture where a host serves as both a client and a server. An example of such a situation would be two network service providers needing to exchange management information

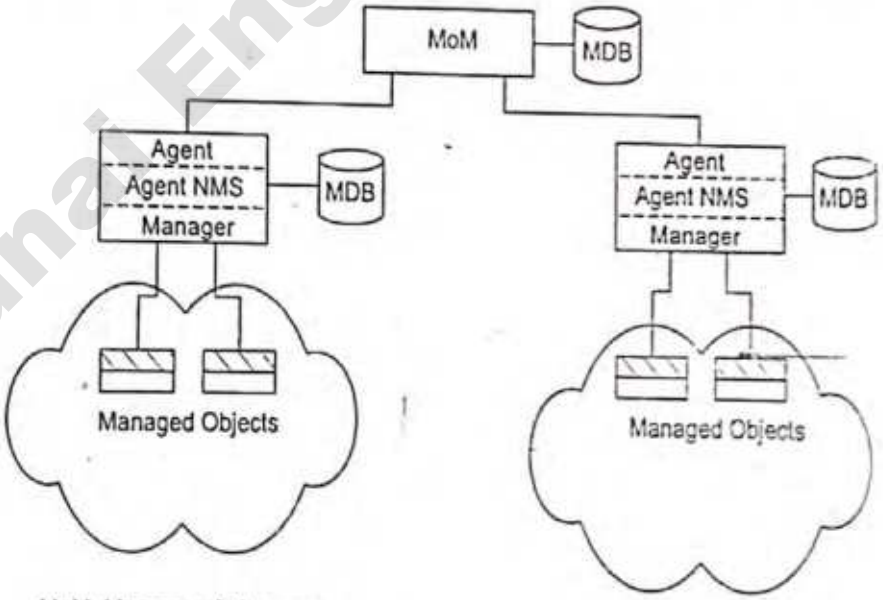


Figure 3.4 Network Management Organization Model with MoM

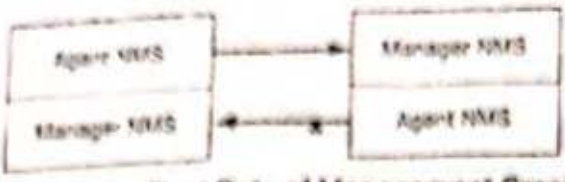


Figure 3.5 Dual Role of Management Process

between them. From the user's point of view, the information traverses both networks and needs to be monitored end-to-end.

In the above discussion, we have used the term network management system (NMS) to mean a system that runs a management process, not just a managed object. Thus, the agent and the manager devices are defined as agent NMS and manager NMS, as shown in Figure 3.4 and Figure 3.5.

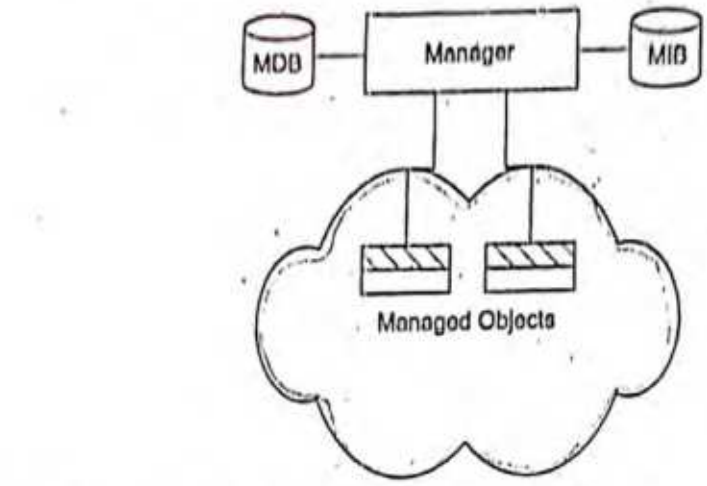
3.4 INFORMATION MODEL

An information model is concerned with the structure and storage of information. Let us consider, for example, how information is structured and stored in a library and is accessed by all. A book is uniquely identified by an International Standard Book Number (ISBN). It is a ten-digit number identification that refers to a specific edition of a specific book. For example, ISBN 0-13-437708-7 refers to the book "Understanding SNMP MIBs" by David Perkins and Evan McGinnis. We can refer to a specific figure in the book by identifying a chapter number and a figure number; e.g., Fig. 3.1 refers to Figure 1 in Chapter 3. Thus, a hierarchy of designation {ISBN, Chapter, Figure} uniquely identifies the object, which is a figure in the book. "ISBN," "Chapter," and "Figure" define the syntax of the three pieces of information associated with the figure; and the definition of their meaning in a dictionary would be the semantics associated with them.

The representation of objects and information that are relevant to their management forms the management information model. As discussed in Section 3.3, information on network components is passed between the agent and management processes. The information model specifies the information base to describe managed objects and the relationship between managed objects. The structure defining the syntax and semantics of management information is specified by Structure of Management Information (SMI). The information base is called the Management Information Base (MIB). The MIB is used by both agent and management processes to store and exchange management information. The MIB associated with an agent is called an agent MIB and the MIB associated with a manager is designated as the manager MIB. The manager MIB consists of information on all the network components that it manages; whereas the MIB associated with an agent process needs to know only its local information, its MIB view. For example, a county may have many libraries. Each library has an index of all the books in that location—its MIB view. However, the central index at the county's main library, which manages all other libraries, has the index of all books in all the county's libraries—global manager MIB view.

Figure 3.6 expands the network configuration that is shown in Figure 3.2 to include the MIB that is associated with the manager. Thus, the manager has both the management database (MDB) and the MIB. It is important to distinguish between the two. The MDB is a real database and contains the measured or administratively configured value of the elements of the network. On the other hand, the MIB is a virtual database and contains the information necessary for processes to exchange information among themselves.

Let us illustrate the distinction between MIB and MDB by considering the scenario of adding a component to the network. Assume that all the hubs in the network are made by a single vendor,



MDB: Management Database
MIB: Management Information Base

Agent Process

Figure 3.6 Network Configuration with Data and Information Base

say Cabletron. The manager in Figure 3.6 has knowledge about the Cabletron hub and its associated parameters in its MIB; and the values associated with the parameters with the hubs are in its MDB. For example, the number of ports in the hub is a parameter associated with the hub (MIB information); and if they are 12-port hubs, the values associated with the number of ports are 12 (MDB information). Suppose we now add another Cabletron hub to the network. The manager would discover the newly added hub during its next discovery process, which could be just a broadcast ping from the manager. The new hub is another instance of the hub with a new IP address, and its MIB information is already in the manager's MIB. Its address and the number of ports associated with it are added to MDB by the manager querying the agent.

Now, let us add a 3Com hub to the network. Let this be the first time that a 3Com hub is added to the network. The manager would recognize the addition of a new component to the network by the periodic broadcast ping of the network by the manager. However, it would not know what component has been added until the MIB information on the 3Com hub is added to the manager's MIB. This information is actually compiled into the manager's MIB schema. After the information on the 3Com hub has been added to the manager's MIB, it can send queries to the agent residing in the 3Com hub. It then retrieves the values for the type of hub, the number of ports, etc. and adds them to its MDB.

The MIB that contains data on managed objects need not be limited to just physical elements. For example, in network management, management information extends information beyond that associated with the description of network elements or objects. Here are some examples of information that can be stored in the MIB.

Network Elements:	hubs, bridges, routers, transmission facilities, etc.
Software Processes:	programs, algorithms, protocol functions, databases, etc.
Administrative Information:	contact person, account number, etc.

In fact, any type of information could be included as an object in the MIB.

3.4 Management Information Tree

The managed objects are uniquely defined by a tree structure specified by the OSI model and are used in the Internet model. Figure 3.7 shows the generic representation of the tree, defined as the Management Information Tree (MIT). There is a root node and well-defined nodes underneath each node at different levels, designated as Level 1, Level 2, etc. Each managed object occupies a node in the tree. In the OSI model, the managed objects are defined by a containment tree representing the MIT.

Figure 3.8 shows the internationally adopted OSI MIT. The root node does not have an explicit designation. The root has three nodes in the layer beneath it—iso, ccitt (itu), and iso–ccitt, (iso–itu). The iso defines the International Standards Organization and ccitt, or itu, defines the International Telecommunications Union (the old name is ccitt). The two standards organizations are on the first layer defining managing objects under them. The joint iso–itu node is for management objects jointly defined by the two organizations. The number in each circle identifies the designation of the object in each layer. Thus, iso is designated as 1, org as 1.3, dod, Department of Defense, as 1.3.6, and the internet as 1.3.6.1. All Internet-managed objects will be that number followed by more dots and numbers. It is to be noted that the names of the nodes are all in lowercase letters as a convention, which we will formally define in Section 3.6.

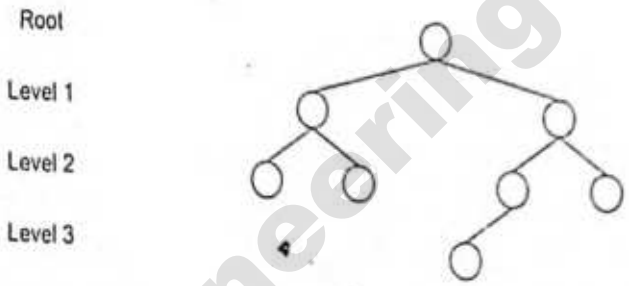


Figure 3.7 Generic Representation of the Management Information Tree

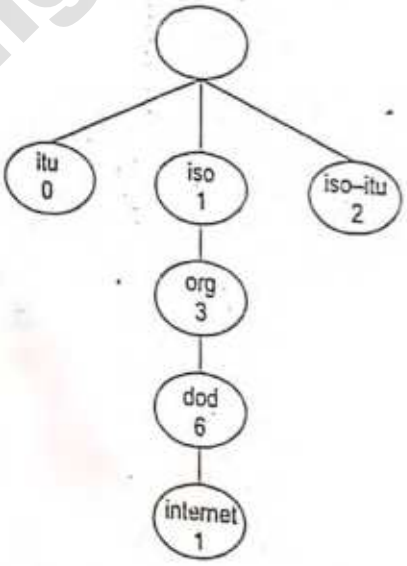


Figure 3.8 OSI Management Information Tree

3.4.2 Managed Object Perspective

Although a managed object need not be a physical object that can be seen, touched, or felt, it is convenient to use a physical representation to understand the characteristics and operations associated with a managed object. Let us consider an object, which is circular in shape. We can define the object in English language syntax as *circle*. To associate a meaning with the object name, *circle*, we can use Webster's dictionary *definition* "a plane figure bounded by a single curved line every point of which is equally distant from the point at the center of the figure." In other words, the definition is a textual description of the object. The object can be viewed and its parameters changed by people who have *access* to it. The *access* privilege could be limited to just accessing it or performing some action on it; for example, resetting a counter value to two. These are all defined as *access* attributes. If we envision a scenario in which the object is used by a nursery school to explain shapes to children, it should at least have some basic shapes, such as a circle, a square, etc. We can define the basic objects that are required of a group (of objects) as the status of the object—whether it is mandatory or optional to have (implement) that object. This attribute of the object is defined as the *status* of the object. There could be many types of objects in the nursery school that are circular in shape. There is a unique identification and name (*object identifier* and *descriptor*) associated with each of them, such as a ring, a donut, etc. There could be many instances of ring and donut; but we are only addressing the types of object, not instances of them here. We have thus defined the five basic attributes of a managed object type from the Internet perspective. They are *name, definition, syntax, access, and status.*

A pictorial view of a circular object in the Internet is shown in Figure 3.9(a).

A managed object in the Internet is defined by five parameters [RFC 1155]. They are:

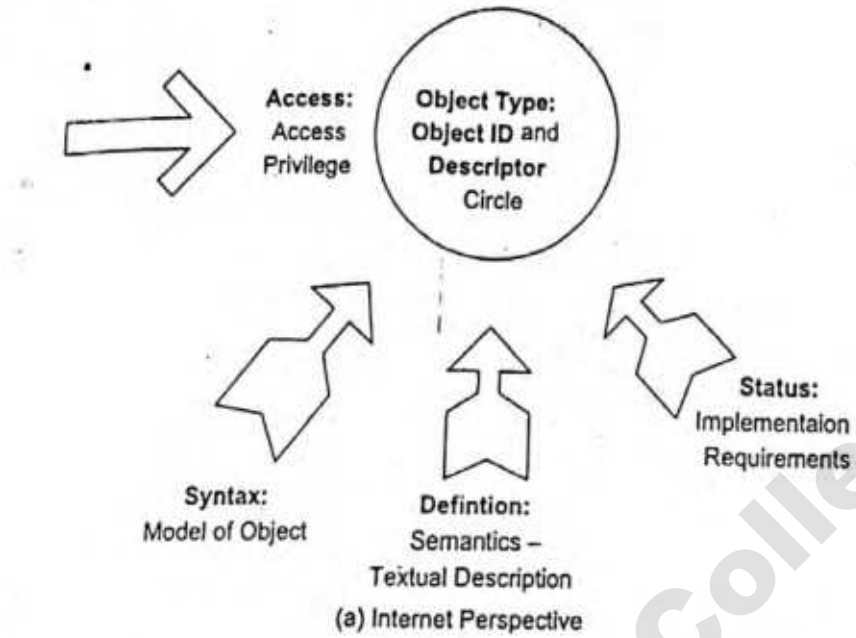
- *object identifier* and *descriptor* — unique ID and name for the object
- *syntax* — used to model the object
- *access* — access privilege to a managed object
- *status* — implementation requirements
- *definition* — textual description of the semantics of object type

A modification of this is specified in RFC 1212, as we shall see in the next chapter.

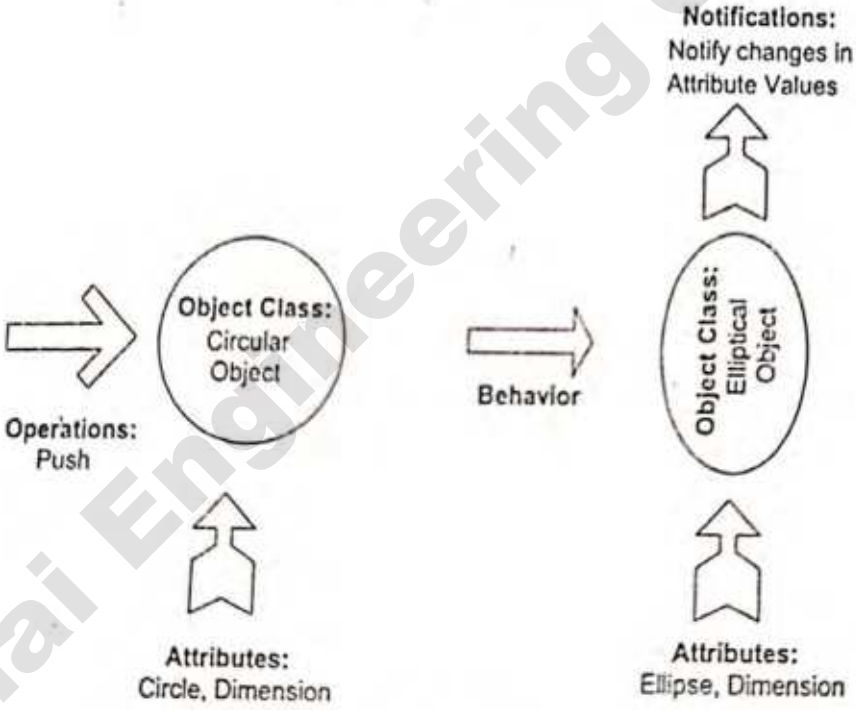
The Internet object model is a scalar model and is easy to understand, as seen above. In contrast, the OSI perspective of a managed object is complex and has a different set of characteristics. We will extend the above analogy of the circular object in a nursery to illustrate an OSI perspective.

Figure 3.9(b) presents the conceptual OSI representation of the various characteristics of a managed object. As mentioned earlier, OSI specifications are object oriented, and hence a managed object belongs to an object class. The left side of Figure 3.9(b) presents the same circular object in the OSI model. The definition of an object in an object-oriented perception would include both the shape and values. Thus, the *attribute* of the object is a circle with given dimensions. The *attribute* of an object defines the external perspective of the object. It undergoes an *operation* "push." Push is not really an OSI operational entity, but is used here to illustrate the concept. The *behavior* of the object is to change its shape or attribute from a circle to an ellipse. It then sends *notifications* to the relevant community informing of its change. Thus, the characteristics of an OSI managed object are:

- *object class* — managed object
- *attributes* — attributes visible at its boundary
- *operations* — operations that may be applied to it



(a) Internet Perspective



(b) OSI Perspective

Figure 3.9 Conceptual Views of Managed Object

- *behavior* behavior exhibited by it in response to an operation
- *notifications* notifications emitted by the object

It is hard to compare the characteristics of a managed object in the Internet and OSI models on a one-to-one basis, as they are very much different. However, it can be observed in the conceptual models in Figure 3.9 that the OSI characteristics—*operations*, *behavior*, and *notification*—are a part of the Internet communications model. Operation in the Internet is done by get and set commands. Notification is done by response and alarm messages. The *syntax* characteristic of the Internet is part of OSI *attributes*

Characteristics	Example
Object type	PktCounter
Syntax	Counter
Access	Read-only
Status	Mandatory
Description	Counts number of packets

(a) Internet Perspective

Characteristics	Example
Object class	Packet Counter
Attributes	Single-valued
Operations	get, set
Behavior	Retrieves or resets values
Notifications	Generates notifications on new value

(b) OSI Perspective

Figure 3.10 Packet Counter as an Example of a Managed Object

The *access* characteristic of the Internet is a part of the security function in the OSI functional model. The *status* characteristic of the Internet is handled by conformance as a part of application services in OSI. Further, in OSI we can create and delete objects, while these concepts do not exist in the Internet. Objects in early SNMP management are assumed to exist for management purposes.

Figure 3.10 shows the comparison between Internet and OSI specifications for the object, packet counter. An example of a packet counter as a managed object in the Internet model is given in Figure 3.10(a). The *object type* (we will define *object id* later) is PktCounter. The *syntax* is Counter. The *access* mode is read-only. The *status* implementation is mandatory, which mandates that this object must be implemented if the group it belongs to is implemented. The *description* provides the semantics that the packet counter counts the number of packets.

The example of the same counter as a managed object in the OSI model is given in Figure 3.10(b). The counter is defined as an *object class, Packet Counter.* It could be related to either a sub- or superclass. The *attribute* value is single-valued. We can perform get and set *operations* on its *attribute.* Its *behavior* to a set operation would be to reset the counter, or just retrieve data if the operation is get. The new value is sent out as *notification.*

3.5 COMMUNICATION MODEL

We have discussed in the previous section how information content is defined (SMI) and stored (MIB). We will now address the model associated with how the information is exchanged between systems. Management data are communicated between agent and manager processes, as well as between manager processes. Three aspects need to be addressed in the communication of information between two entities: transport medium of message exchange (transport protocol), message format of communication (application protocol), and the actual message (commands and responses). Let us illustrate this by an example of Azita buying a car from an automobile salesperson, Roberto.

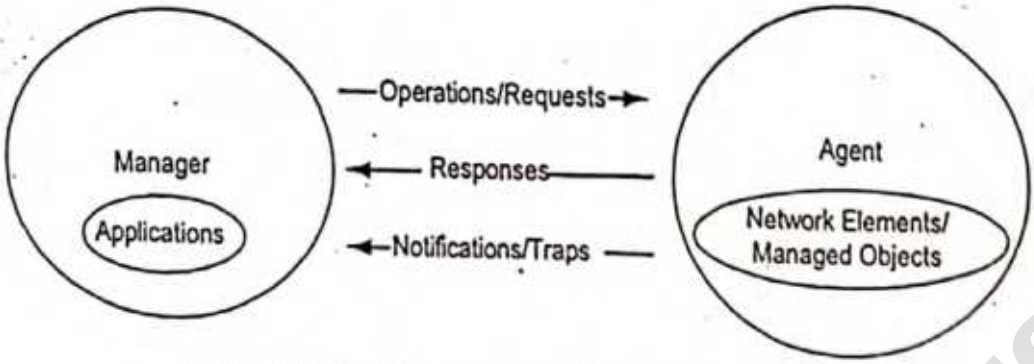


Figure 3.11 Management Gommunication Model

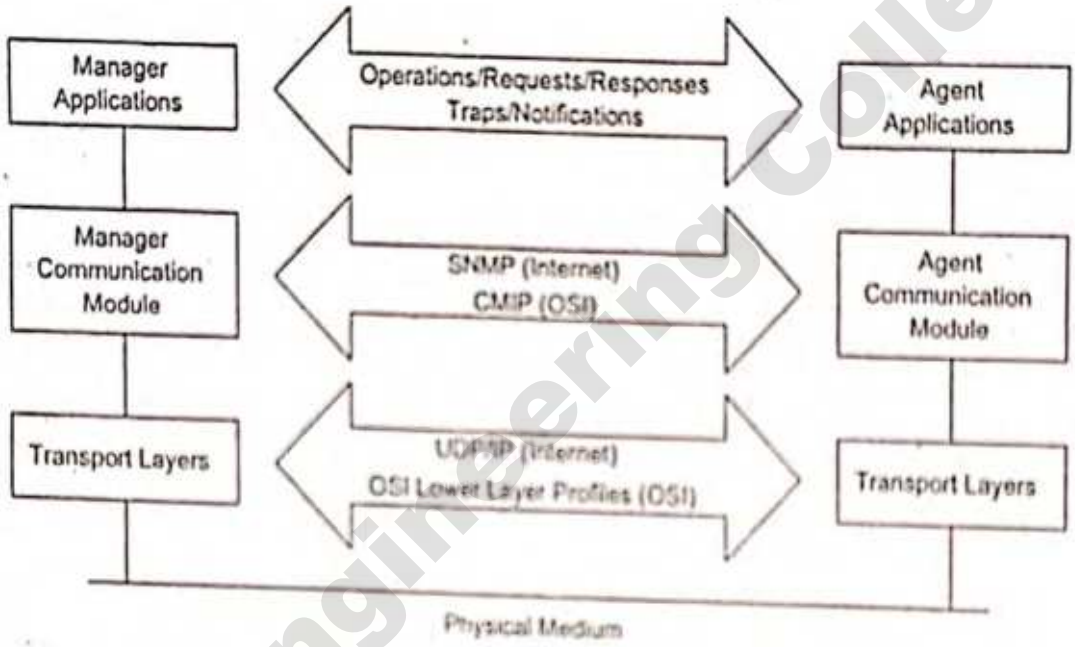


Figure 3.12 Management Communication Transfer Protocols

Azita could go to the automobile dealer and communicate in person with Roberto. Alternatively, she could communicate with Roberto via the Internet. In the former, visual and audio media are the transport mechanisms, and electronic exchange is used in the latter. The communication at the application level could be exchanged in English, Spanish, or any other mutually understandable language between the two. This would be the application-level protocol that is decided between Azita and Roberto. Finally, there are messages exchanged between Azita and Roberto. For example, Azita could request what cars are available and Roberto would respond with the cars that are in stock. Azita could then set a price range and Roberto responds with cars that match the price range. These exchanged messages are the commands/requests/operations and responses notifications. They can be considered services requested by Azita and provided by Roberto.

Figure 3.11 presents the communication model. The applications in the manager module initiate *requests* to the agent in the Internet model. It is part of the *operations* in the OSI model. The agent executes the request on the network element; i.e., managed object, and returns *responses* to the manager. The traps/notifications are the unsolicited messages, such as alarms, generated by the agent.

Figure 3.12 presents the communication protocol used to transfer information between managed object and managing processes, as well as between management processes. The OSI model uses CMIP

along with CMIS. The Internet uses SNMP for communication. The services are part of operations using requests, responses, and alarm notifications.

OSI uses both connection-oriented and connectionless protocols for transportation. For example, the TP4 transport layer protocol riding on top of the x.25 protocol could be used for connection-oriented transporting and application messages. TP4 over Connectionless Network Protocol is used for connectionless transportation. The Internet uses connectionless UDP/IP protocol for transporting messages.

CMIP and SNMP specify the management communication protocols for OSI and Internet management. CMIP is addressed in Appendix A. SNMP is extensively covered throughout the book.

The application processes invoke the management communication layer protocols. OSI deals with messages in the specification of managed objects. Managed objects and their attributes could be manipulated by operations. Basic application service modules are defined by CMIS. In the Internet, operations are executed by SNMP messages.

3.6 ABSTRACT SYNTAX NOTATION ONE: ASN.1

In both the information model and the communication model, discussed in the previous sections, we have addressed functions. In these models, SMI needs to be specified syntactically and semantically, which will be the content of this section.

It is important for communication among systems that a formalized set of rules is agreed upon on the structure and meaning of the language of communication, namely syntax and semantics of the language. There are numerous sets of application and transport protocols. Thus, it is beneficial to choose a syntactical format for the language that specifies the management protocol in the application layer, which is transparent to the rest of the protocol layers. One such format is an old and well-proven format, Abstract Syntax Notation One, ASN.1. We will introduce ASN.1 here to the extent needed to understand its use in network management. The reader is referred to other references [Cassel and Austing, 1996; Larmouth, 1997; Stallings, 1998] for greater depth on the subject.

ASN.1 is more than just syntax. It is a formal language developed jointly by CCITT (now ITU-T) and ISO for use with application layers for data transfer between systems. It is also applicable within the system for clearly separating the abstract syntax and the transfer syntax at the presentation layer. We define the *abstract syntax* as the set of rules used to specify data types and structures for the storage of information. The *transfer syntax* represents the set of rules for communicating information between systems. Thus, the abstract syntax would be applicable to the information model discussed in Section 3.4 and the transfer syntax to the communication model discussed in Section 3.5. The abstract syntax can be used with any presentation syntax, the latter depending on the medium of presentation. The abstract syntax in ASN.1 makes it independent of the lower-layer protocols. ISO 8824/X.208 standards specify ASN.1. The algorithm to convert the textual ASN.1 syntax to machine-readable code is defined in ISO 8825/X.209 standards. It is called Basic Encoding Rules (BER).

3.6.1 Terminology, Symbols, and Conventions

ASN.1 syntax is based on the Backus system and uses the formal syntax language and grammar of Backus–Nauer Form (BNF), which looks like:

<name> ::= <definition>

where the notation "<entity>" denotes an "entity" and the symbol "::=" represents "defined as"

Let us illustrate the Backus system by developing a simple arithmetic expression <SAE> [Maurer 1977]:

We can define an entity <digit> as

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

where the symbol "|" represent "or." We can also define an operation entity <op> as

<op> ::= + | - | x | /

The definitions on the right side are called *primitives*. Using these primitives, we can construct more entities. Thus, an entity *number* can be constructed from the primitive, <digit>

<number> ::= <digit> | <digit><number>

For example, the number 9 is the digit 9; the number 19 is the concatenation of the digit 1 and the number 9; and the number 219 is the concatenation of digit 2 with the number 19.

We can now construct a simple arithmetic expression <SAE> from the primitives and the construct <number>. Thus,

<SAE> ::= <number> | <SAE> | <SAE><op><SAE>

The format of each line is defined as a *production* or *assignment*.

Let us consider an example with the following two assignments:

<BooleanType> ::= BOOLEAN

<BooleanValue> ::= TRUE | FALSE

The expression on the left side specifies the name of the type and the right side is the definition or value of the type. Thus, BooleanType is defined as BOOLEAN and BooleanValue is defined as either TRUE or FALSE. The above example illustrates the two basic parameters associated with an entity, namely, *data type* and *value*. The first line is called *data type assignment* and it defines the name of the entity; and the second line, value assignment, specifies the assigned value to the data type. Thus, in the above example the entity BOOLEAN can have assigned values of TRUE or FALSE. Entities that are all in capital letters, such as TRUE and FALSE, are called *keywords*.

A group of assignments makes up an ASN.1 module. For example, a name consists of first, middle, and last names, and they can be specified as:

```
person-name    Person-Name::=
               {
               first     "John",
               middle    "I",
               last      "Smith"
               }
```

Here person-name, beginning with lowercase letters, is the name of the data type. Person-Name is a module and begins with capital letters. The module comprises three assignments, whose names are first, middle, and last with values "John," "I," and "Smith."

Figure 3.13 and Figure 3.14 show two examples of ASN.1 data type definition [Larmouth, 1997]. They are two ASN.1 modules defining data types personnelRecord and trade-Message. Because they are modules, they start with capital letters. PersonnelRecord describes the personnel record of an employee in a global corporation. The Trade-Message is a module specifying a list of invoices defining customer name, part numbers, quantity, charge, and security authentication.

```
PersonnelRecord ::= SET
        {       name            Name,
        title           GraphicString,
        division    CHOICE
          marketing         [0]   SEQUENCE
              {Sector,
              Country},
          research          [1]   CHOICE
              {product-based    [0]   NULL,
              basic             [1]   NULL},
          production        [2]   SEQUENCE
              {Product-line,
              Country     }                 }
etc.
```

Figure 3.13 ASN.1 Data Type Definition Example 1

```
Trade-Message ::= SEQUENCE
        {invoice-no         INTEGER
        name                GraphicString,
        details             SEQUENCE OF
                                SEQUENCE
                    {part-no            INTEGER
                    quantity            INTEGER },
        charge              REAL,
        authenticator       Security-Type}

Security-Type ::= SET
        {       ...
                ...
                ...     }
```

Figure 3.14 ASN.1 Data Type Definition Example 2

Note that in the examples of Figure 3.13 and Figure 3.14, the data types are built-up from primitive data types: INTEGER, REAL, NULL, and GraphicString. GraphicString is one of several Character-String type primitives. These examples present three kinds of data types. which are built using three construction mechanisms:

alternatives:	CHOICE
list:	SET and SEQUENCE
repetition:	SET OF and SEQUENCE OF

These constructs are used to build structured data types. Just as we saw in the <SAE> example earlier, all data types are built from the ground up using primitive (also called atomic) entities. ASN.1 definition allows both backward and forward references, as well as in-line definition. For instance, in Figure 3.13 the data types Name, Sector, Country, and Product-line are defined externally either before or after the module defining PersonnelRecord. The data type whose name is title is defined in-line as the data type GraphicString. It could have been defined as data type Title as follows:

```
title     Title::= GraphicString
```

Let us analyze the three construct types. In PersonnelRecord, the person works in one of the three divisions—marketing, research, or production. This is built using CHOICE construction. Notice that in each of those divisions, research could be either product-based or basic.

The constructs SET and SEQUENCE are list builders. The PersonnelRecord module is a set of data types, Name, GraphicString, Sector, Country, etc., which are all different data types. Since they are different and each is uniquely associated with a name, they can be encoded and transmitted in any order. For example, they could be arranged in any of the following orders:

"Smith", "Manager", {"North", "Chile"}
"Manager", "Smith", {"North", "Chile"}
{"North", "Chile"}, "Manager", "Smith"
etc.

Notice that "North" and "Chile" are always in the same order. This is because it is a list built with SEQUENCE construction, and the order in the list should be maintained.

The third type of construction is the repetitive types SET OF and SEQUENCE OF. In the example on TradeMessage in Figure 3.14, the SEQUENCE OF construction is shown. The "details" in the invoice are a repetition of data consisting of the ordered list (SEQUENCE construct) of part number and quantity in each invoice. The repetitive records themselves are ordered in a SEQUENCE OF construction. This means that the data will be transmitted in the order in which they are entered. The encoding scheme will preserve that order while transmitting the data from one process to another. For example, if data are entered for *details* in Figure 3.14 as a sequence {part-no, quantity} in the order {1, 5}, {60, 3}, {120, 40}, they will be transmitted in that order by the sending process. If they had been a SET OF construct instead of a SEQUENCE OF construct for *details* in Figure 3.14, the order is irrelevant. The order in this case for the example could be encoded and transmitted by the sending process as any of the combinations, {1, 5}, {60, 3}, {120, 40}; or {60, 3}, {1, 5}, {120, 40}; or {120, 40}, {1, 5}, {60, 3}; etc. without relevance to the order.

The NULL data type used in Figure 3.13, PersonnelRecord, is a placeholder. No value needs to be associated with it except indicating that such a data type exists.

We observe in the PersonnelRecord example in Figure 3.13 that some assignments have integers in square brackets. For instance,

```
{product-based [0] NULL,
basic          [1] NULL}
```

These are called *tags*. The definition of a tag is introduced in ASN.1 to uniquely identify a data type and will be discussed in detail later.

We have used several symbols and primitive data types including *keywords* in the preceding examples. A complete list of ASN.1 symbols is shown in Table 3.2.

Table 3.3 lists some of the frequently used ASN.1 keywords. The reader is directed to the reference [Perkins and McGinnis, 1997] for a more complete list.

As we said earlier, we can group assignments that are related to each other; this group is called a module. A formal definition of a module is as follows:

```
<module name> DEFINITIONS ::= BEGIN
        <name> ::= <definition>
        <name> ::= <definition>
END
```

Table 3.2 ASN.1 Symbols

SYMBOL	MEANING
::=	Defined as or assignment
|	Or, alternatives, options of a list
-	Signed number
--	Following the symbol are comments
{}	Start and end of a list
[]	Start and end of a tag
()	Start and end of a subtype
..	Range

Table 3.3 ASN.1 Keywords

KEYWORD	BRIEF DESCRIPTION
BEGIN	Start of an ASN.1 module
CHOICE	List of alternatives
DEFINITIONS	Definition of a data type or managed object
END	End of an ASN.1 module
EXPORTS	Data types that can be exported to other modules
IDENTIFIER	A sequence of non-negative numbers
IMPORTS	Data types defined in external modules
INTEGER	Any negative or non-negative number
NULL	A placeholder
OBJECT	Used with IDENTIFIER to uniquely identify an object
OCTET	Unbounded 8-bit bytes (octets) of binary data
OF	Used with SET and SEQUENCE
SEQUENCE	Ordered list maker
SEQUENCE OF	Ordered array of repetitive data
SET	Unordered list maker
SET OF	Unordered list of repetitive data
STRING	Used with OCTET for denoting a string of octets

For example, a MIB definition module will look like:

```
RFC1213-MIB DEFINITIONS ::= BEGIN
...
...
...
END
```

The terms DEFINITIONS, BEGIN, and END are primitives and are called keywords in ASN.1. They are built-in expressions and have special meaning. The DEFINITIONS indicate that the named module,

Table 3.4 ASN.1 Data Type Conventions

DATA TYPES	CONVENTION	EXAMPLE
Object name	Initial lowercase letter	sysDescr, etherStatsPkts
Application data type	Initial uppercase letter	Counter, IpAddress
Module	Initial uppercase letter	PersonnelRecord
Macro, MIB module	All uppercase letters	RMON-MIB
Keywords	All uppercase letters	INTEGER, BEGIN

RFC 1213-MIB, is being defined. The body of a module always starts with BEGIN and ends with END. Grouping assignments into modules has the great advantage that modules can be imported into and exported from other modules. Thus, they are reusable.

We notice in the examples described so far in this section that we have used both lowercase and uppercase letters. There are ASN.1 conventions to designate the data. These are shown in Table 3.4.

3.6.2 Objects and Data Types

We will now use ASN.1 notation to define the various data types and apply them to describe objects in the context of SMI and MIB.

We observed in Section 3.6.1 that the data type could be either a simple type (also called primitive, atomic, or basic), or it could be structured. In addition, we talked about tag designation, which uniquely identifies the data type irrespective of the syntax version. In general, data types are defined based on structure and tag. The structure is subdivided into four categories. The tag is subdivided into class and tag number. This is shown in Figure 3.15. An object can be uniquely defined by its tag, namely class and tag number. For exchange of information between systems, the structure information is also included.

The four categories of data type structure, shown in Figure 3.15 are, *simple type*, *structured type*, *tagged type*, and *other type*.

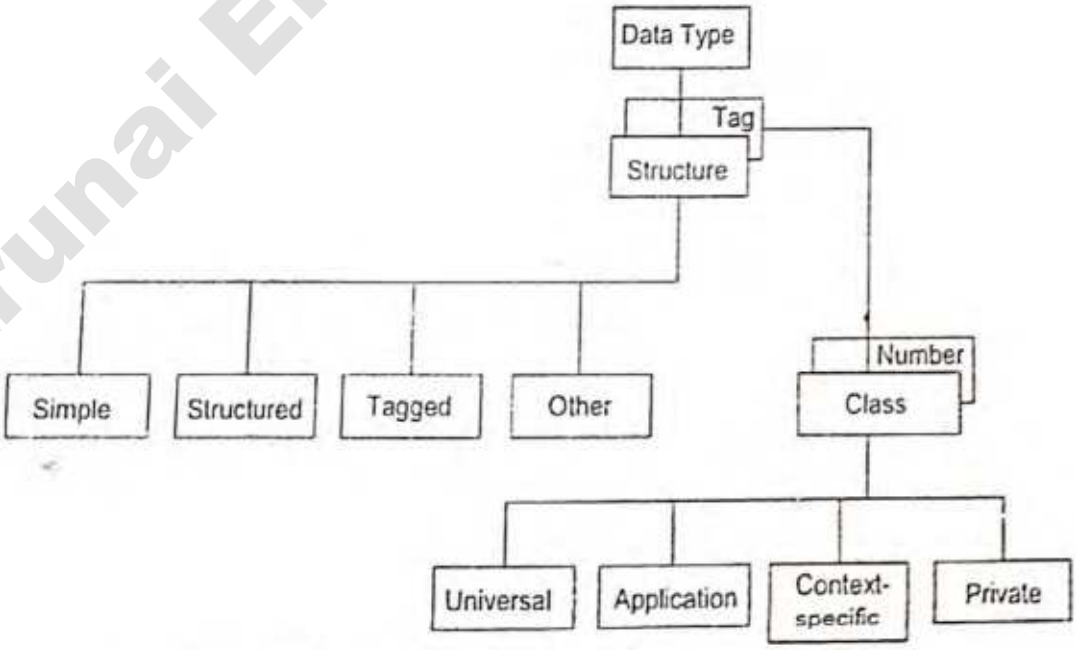


Figure 3.15 ASN.1 Data Type Structure and Tag

A *simple type* is one for which the values are specified directly. For example, we can define a page of a book as PageNumber of simple type, which can take on any integer value. INTEGER is a *simple type*. Thus,

```
PageNumber ::= INTEGER
```

Similarly, we can define the chapter number of the book as

```
ChapterNumber ::= INTEGER
```

Values for PageNumber can be specified as 1, 2, 3,... and for ChapterNumber as 1, 2, 3,...

A data type is defined as a *structured type* when it contains other types. Types that are within a *structured type* are called *component types*. In the above example, a page number of a book could be defined as a *structured type* by a SEQUENCE construction of ChapterNumber and PageNumber component data types. Let us call it BookPageNumber.

```
BookPageNumber ::= SEQUENCE
                        {ChapterNumber, Separator, PageNumber}
```

where Separator is a data type with value "-." BookPageNumber is a *structured type*. Values for BookPageNumber would then be like 1-1, 2-3, or 6-25.

We can define all the pages of the book as a collection of individual pages. If we want to define them in a sequential order from the first page of the first chapter to the last page of the last chapter, we would use a SEQUENCE OF construction. Let us call it BookPages.

```
BookPages ::= SEQUENCE OF { BookPageNumber}
```

We could define the same in an alternative manner as

```
BookPages ::= SEQUENCE OF
                        {
                        SEQUENCE
{ChapterNumber, Separator, PageNumber}
                        }
```

The above two definitions have identical meaning. Values for BookPages would then be 1-1, 1-2, 1-3, ... 2-1, 2-2, 2-3,... The ordering of the values is by the order in which the data are specified and not by sorting of the component data types in the structured construct.

The pages of a book could also be specified as a collection of individual pages in random order. The *structured type* for BookPages would then be constructed with the SET OF data type construct:

```
BookPages ::= SET OF
                        {
                        SEQUENCE
{ChapterNumber, Separator, PageNumber}
                        }
```

Note that we could not have used a SET OF construct for BookPageNumber as the order of the chapter number, separator, and page number is important to keep. However, we could have used the SET construct to define BookPages as

```
BookPages ::= SET {ChapterNumber, Separator, PageNumber}
```

and assigned values 1-2, 2-3, 1-1, ... in a random order. The order of the values in the transmission of data between the sender and the receiver is unimportant. Thus, SET is distinguished from SEQUENCE in two respects. First, the data types should all be distinct; and second, the order of values in SET is of no consequence, whereas it is critical in a SEQUENCE construct. It is also worth noting that the component data types in a SEQUENCE construct need not be distinct since the order is preserved.

Tagged type is a type derived from another type and is given a new tag id. Although a data type has a unique tag associated with it, a tag data type is defined to distinguish types within an application. For instance, in Figure 3.14 although "invoice-no" is an INTEGER type, which we will soon learn as a universal class with a tag number [1], it could have been assigned a local tag id. This is sometimes done to improve the efficiency of encoding.

The fourth and last category of structure is *other type* which is a data type that is not pre-defined. It is chosen from CHOICE and ANY types, which are contained in other types. Type CHOICE defines the selection of one value from a specified list of distinct types. Thus, in Figure 3.13, "research" uses a CHOICE construct to select one of the two alternatives between "product-based," and "basic." We can represent them with specific values instead of NULL, as follows:

```
research        Research ::= CHOICE
                             {
                                product-based      ProductType,
                                basic              VisibleString
                             }
                             ProductType ::=       VisibleString
```

Type ANY is always supplemented with any valid ASN.1 type defined in another module. We have given two representations for Research, the one above and the other in Figure 3.13. We could give a definition of these two options by defining Research as follows:

```
Research ::= CHOICE
        {
        product-based  ANY,
        BASIC          ANY
        }
```

This definition using ANY specifies that the "product-based" entity could be either a NULL or a ProductType data type, and similarly "basic" could be either VisibleString or NULL.

Figure 3.15 shows two perspectives of data type—*structure* and *tag*. The structure that we have so far described addresses how the data type is constructed. On the other hand, *tag* uniquely identifies the data type. It is required for encoding the data types for communication. Every data type except CHOICE and ANY has a tag associated with it. Tag has two components—*class* and *tag number*. There are four classes of tag. They are: *Universal*, *Application*, *Context-specific*, and *Private;* and each data type belonging to each class is assigned a unique number.

The *universal class* is the most commonly used class, and the ASN.1 list of *universal class* assignments is given in Table 3.5. A core set of assignments is used in all applications. Data types belonging to the *universal class* are application-independent. It is similar to the use of a global variable in a software program, and is applicable anywhere in a program. It need not be defined repeatedly in the subroutines of the program. BOOLEAN and INTEGER are examples of a *universal class*, whose tag numbers are [1] and [2], respectively.

Table 3.5 Universal Class Tag Assignments

TAG	TYPE NAME	SET OF VALUES
Universal 1	BOOLEAN	TRUE or FALSE
Universal 2	INTEGER	0, Positive and negative numbers
Universal 3	BIT STRING	A string of binary digits or null set
Universal 4	OCTET STRING	A string of octets or null set
Universal 5	NULL	Null, single-valued
Universal 6	OBJECT IDENTIFIER	Set of values associated with the object
Universal 7	Object description	Human readable text describing the object
Universal 8	EXTERNAL	The type is external to the standard
Universal 9	REAL	Real numbers, expressed in scientific notation Mantissa × $Base^{exponent}$
Universal 10	ENUMERATED	Specified list of integers
Universal 11	ENCRYPTED	Encrypted information
Universal 12–15	Reserved for future use	
Universal 16	SEQUENCE and SEQUENCE OF	Ordered list of types
Universal 17	SET and SET OF	Unordered list of types
Universal 18	NumericString	Digits 0–9, space
Universal 19	PrintableString	Printable characters
Universal 20	TeletexString	Character set specified by CCITT Recommendation T.61
Universal 21	VideotexString	Character set specified by CCITT Recommendation T.100 and T.101
Universal 22	IA5String	International Alphabet 5, which is equivalent to ASCII
Universal 23	UTCTime	Time format YYMMDDHHMM[SS][local time differential from universal standard time]
Universal 24	GeneralizedTime	Time format YYYYMMDDHHMM[SS][local time differential from universal standard time]
Universal 25	GraphicString	Graphic character set specified by ISO 8824
Universal 26	VisibleString	Character set specified by ISO 646, equivalent to ASCII
Universal 27	GeneralString	General character string
Universal 28	CharacterString	Character set
Universal 29–	Reserved for future use	

Tags belonging to the *application class* are specific to applications. Examples of application-spec tag numbers are used in examples in Figure 3.13. A universal class tag number can be overridden w an application-specific *tag number*. Types in two different applications can have the same applicat specific tag, but carry two different meanings.

Application-specific assignments are classified as such. For instance, in the above example of Book-PageNumber, if we assign PageId, ChapterNumber, PageNumber, and the tags APPLICATION 1, 2, and 3, respectively, the assignment will read

```
PageId ::= [APPLICATION 1] SEQUENCE {
[APPLICATION 2] ChapterNumber,
[APPLICATION 3] PageNumber}
```

When defining large modules, the structure can become large. We can introduce descriptive names and comments on the structure for easy reading. Let us expand the above example as follows:

```
PageId ::= [APPLICATION 1] SEQUENCE {
        chapter-number      [APPLICATION 2] ChapterNumber,
        page-number   [APPLICATION 3] PageNumber}
    -- page numbers are grouped by chapter numbers
```

The descriptive words "chapter number" and "page number" do not affect the result when encoding this structure, neither do the comments following the "--".

In the previous example, both PageNumber and ChapterNumber have INTEGER values. INTEGER can be classified as either UNIVERSAL 2 or APPLICATION 3. This could be encoded either way. The efficiency of encoding can be improved if we had added the data designation IMPLICIT as below:

```
PageNumber ::=  [APPLICATION 3] IMPLICIT INTEGER
```

Such an expression forces the encoding to follow the local tag assignment.

The *context-specific type*, a subset of an application, is limited to that application. Thus, in Example 1 of Figure 3.13, research has a tag [1] associated with the application of PersonnelRecord and under that application, research has two context-specific tags [0] and [1] for product-based and basic.

The *private type* is used extensively by vendors of network products. A vendor is assigned a node on the MIT, and all branches and leaves under that node will be assigned a private data type by the vendor.

Before leaving the subject of tags, it is worth noting a special case of data type INTEGER. It is an ENUMERATED type and is similar to INTEGER. For example, we can define the colors of the rainbow as ENUMERATED type integers.

```
RainbowColors ::= ENUMERATED
        {
            violet   (0)
            indigo   (1)
            blue     (2)
            green    (3)
            yellow   (4)
            orange   (5)
            red      (6)
        }
```

In this case, when a value of 5 is designated for the object type RainbowColors, it is implied that it is orange. RainbowColors could take on only the seven integer values defined.

An example for the ENUMERATED type for INTEGER from SNMP MIB, which we will cover in Chapter 5, *error status* in a *get-response* message is:

```
ErrorStatus ::=
        INTEGER{
                NoError(0)
                tooBig(1)
                noSuchName(2)
                badValues(3)
                readOnly(4)
                genErr(5)
                }
```

A subtype data type is derived from a parent type. For example, in the PageNumber example, if we limit the maximum page number to 255 (based on 2^8), then the assignment would read

```
PageNumber ::= Integer (0..255)
```

The parenthesis indicating that it is a subtype expression (see Table 3.2), where the integer range is from 0 to 255.

Let us conclude this section with a real-life example in network management of a data type, which is the address translation table in SNMP IP MIB. An entry in the table is of data type IpNetMediaEntry, which is a sequence of four managed objects with associated data types as shown below. Each of the four objects starts with a lowercase letter, and the associated data type with either a capital letter or is all capital letters.

```
IpNetMediaEntry ::=SEQUENCE {
        ipNetToMediaIfIndex             INTEGER
        ipNetToMediaPhysAddress         PhysAddress
        ipNetToMediaNetAddress                  IpAddress
        ipNetToMediaType                INTEGER}
```

3.6.3 Object Name

In a MIB, there is an identifier for each occurrence of an object. In the ASN.1 notation, it is the OBJECT IDENTIFIER. The object identifier for the Internet shown in Figure 3.8 is

```
internet OBJECT IDENTIFIER ::= {iso(1) org(3) dod(6) internet(1)}
```

Thus, the object identifier for the Internet has the value 1.3.6.1, which we discussed in Section 3.5.1. The MIT shown in Figure 3.8 has been extended to include the class private type in the MIB and is shown in Figure 3.16. Thus, the object identifier for private enterprise IBM is 1.3.6.1.4.1.2.

3.6.4 An Example of Use of ASN.1 from ISO 8824

Figure 3.17 shows the ASN.1 structure for a personnel record. Part (a) shows the informal description, part (b) shows the ASN.1 description of the record, and part (c) shows the description of the record value. There are several salient points to note in this example. First, there are no simple types in this example such as the page number defined in Section 3.6.3. The data type, Name, does not have an associated object name, although we could define one, for example, personnel-name. In such a case, the second line in Figure 3.17(b) would read

```
personnel-name          Name
```

PersonnelRecord is a structured data type, SET with the basic component types Name, EmployeeNumber, Date, Name (nameOfSpouse), and ChildInformation. ChildInformation itself is a

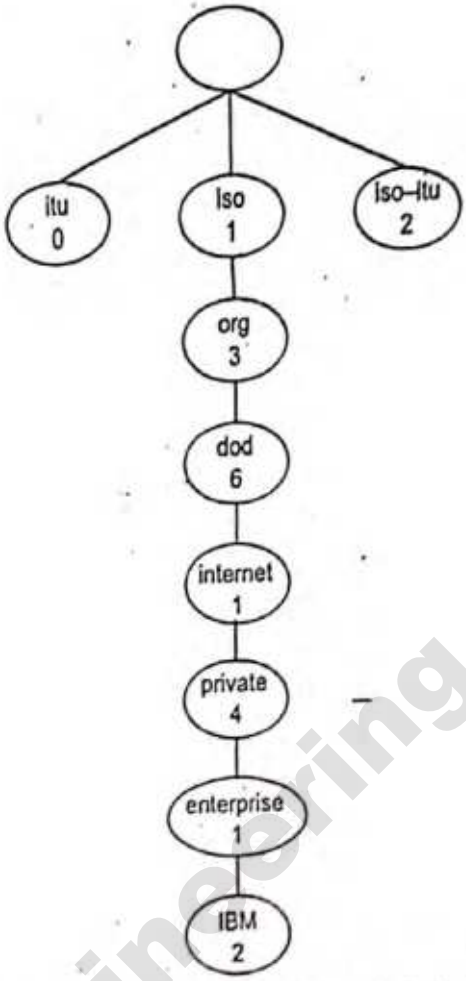


Figure 3.16 IBM as an Example of a Private Class in MIT

structured data type, a SET consisting of Name and Date as component types. A third structured data type that we notice is SEQUENCE for the data type Name with VisibleString as the component types.

The SEQUENCE type is used for Name and the SEQUENCE OF type is used for children, which contains component type SEQUENCE. Thus, the first occurrence of Name in PersonnelRecord is a SEQUENCE construct, and the same construct is embedded in children, which is a SEQUENCE OF construct. Thus, we see a nested structure in this example.

The structure for PersonnelRecord is a structured type and it could have been defined without the data designation IMPLICIT as well as the local tag [APPLICATION 0]. However, as mentioned in Section 3.6.2, the local tag type has been used to improve the efficiency of coding. Further use of the IMPLICIT designation makes the coding more efficient in that it will be encoded with the [APPLICATION 2] tag and not the UNIVERSAL tag, which is also applicable. In this situation, it would not be encoded as UNIVERSAL type 1.

3.7 ENCODING STRUCTURE

The ASN.1 syntax containing the management information is encoded using the BER defined for the transfer syntax. The ASCII text data are converted to bit-oriented data. We will describe one specific encoding structure, called TLV, denoting Type, Length, and Value components of the structure. This is shown in Figure 3.18. The full record consists of type, length, and value.

```
Name:                 John P Smith
Title:                Director
Employee Number       51
Date of Hire:         17 September 1971
Name of Spouse;       Mary T Smith
Number of Children    2
Child Information
        Name          Ralph T Smith
        Date of Birth 11 November 1957
Child Information
        Name          Susan B Jones
        Date of Birth 17 July 1959
```

(a) Informal description of personnel record

```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
        Name,
        title [0] VisibleString,
        number EmployeeNumber,
        dateOfHire [1] Date,
        nameOfSpouse [2] Name,
        children [3] IMPLICIT  SEQUENCE OF ChildInformation DEFAULT { } }
ChildInformation ::= SET {
        Name,
        dateOfBirth [0] Date }
Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
        givenName VisibleString,
        initial VisibleString,
        familyName VisibleString }

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT VisibleString    --YYYYMMDD
```

(b) ASN.1 description of the record structure

```
        {                   {givenName "John", initial "T", familyName "Smith"},
title                       "Director"
number                      "51"
dateOfHire                  "19710917"
nameOfSpouse                {givenName "Mary", initial "T", familyName "Smith"},
children
{{                          {givenName "Ralph", initial "T", familyName "Smith"},
  dateOfBirth               "19571111"},
 {                          {givenName "Susan", initial "B", familyName "Jones"}
  dateOfBirth               "19590717"}}}
```

(c) ASN.1 description of a record value

Figure 3.17 ISO 8824 Example of Use of ASN.1

The type has three subcomponents—class, P/C, and tag number. P/C specifies whether the structure is primitive, i.e., a simple type or a construct, anything other than a simple type. It is encoded as a 1-byte (an octet) field. The two most significant bits (7th and 8th bit) specifying the class are coded as per values

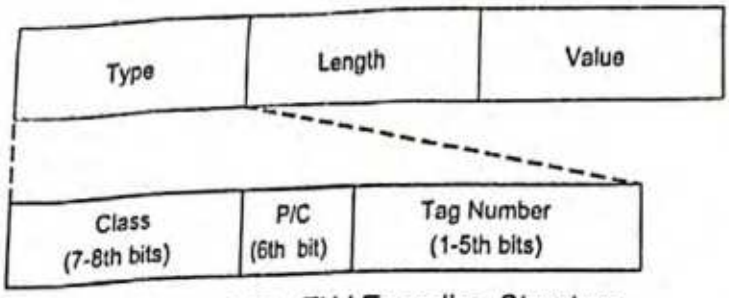


Figure 3.18 TLV Encoding Structure

defined in Table 3.6. The value of P/C is 0 for Primitive and 1 for Construct. The lowest 5 bits (1–5) designate the tag value in binary. For example, INTEGER, from Table 3.5, belongs to a universal class with a tag value of 2 and is a primitive data type. Hence, the type is 00000010.

The length specifies the length of the value field in the number of octets. The length is defined as a series of octets. It is either one octet (short) or more than one octet (long). The most significant bit (8th bit) is set to 0 for a short length with the low 7 bits indicating the length of the value. If the value field is longer than 127 (maximum specified by 7 bits), then the long form is used for length. The 8th bit of the first octet is marked as 1 and the rest of the seven bits of the first octet indicate how many octets follow to specify the length. For example, a value length of 128 would look like

10000001 10000000

The value field is encoded based on the data type. It is a multiple number of octets. The simplest data type value to encode is an OCTET STRING. An octet string of '0C1B'H (the string is designated with apostrophes on both sides and an H denoting hexadecimal notation) would look like

00001100 00011011

The complete TLV for the string of octets '0C1B'H is made up from universal (00) Primitive (0) data type of a tag value of 4 with a one-octet length field to indicate that there are two octets of value field. It is

00000100 00000010 00001100 00011011

The integer value is encoded using a twos-complement form. For a positive value, the actual value is the binary representation with the most significant always being 0 to indicate a positive sign. If the integer exceeds 127, an additional octet of 0s is prefixed. Thus, a value of 255 is written as 00000000 11111111, with the leading 0 indicating the positive sign bit. For a negative integer, the absolute value of the integer is written in a binary form. The leading sign bit should be 0 to indicate the positive sign. Invert all the 1s to 0s and all the 0s to 1s. Then add 1 to the inverted binary digits. The leading sign bit will automatically become 1, indicating a negative integer. For example, a −5 will start as 00000101.

Table 3.6 Value of Class in Type

CLASS	8TH BIT	7TH BIT
Universal	0	0
Application	0	1
Context-specific	1	0
Private	1	1

Inverting the bits and adding 1, it becomes 11111011. Refer to Perkins and McGinnis [1997] for the encoding of other values.

3.8 MACROS

The data types and values that we have so far discussed use ASN.1 notation of syntax directly and explicitly. ASN.1 language permits extension of this capability to define new data types and values by defining ASN.1 macros. The ASN.1 macros also facilitate grouping of instances of an object or concisely defining various characteristics associated with an object.

The structure of a macro takes the form shown in Figure 3.19.

As can be observed from Table 3.4, the keyword for a macro is all in capital letters. TYPE NOTATION defines the syntax of the new types and VALUE NOTATION defines the syntax of the new values. The auxiliary assignments define and describe any new types identified.

The OBJECT-IDENTITY macro is used to define information about an OBJECT IDENTIFIER assignment. Figure 3.20 shows an example from RFC 2578 of creating an Internet object using an OBJECT-IDENTITY macro. The two syntactical expressions STATUS and DESCRIPTION are mandatory and the type ReferPart is optional. The value in VALUE NOTATION defines the object identifier.

As an example of the usage of the OBJECT-IDENTITY macro, let us consider a registration authority that registers all computer science courses that are offered in the College of Computing. Suppose we

```
<macroname> MACRO ::=
BEGIN
        TYPE NOTATION ::= <syntaxOfNewType>
        VALUE NOTATION ::= <syntaxOfNewValue>
        <auxiliaryAssignments>
END
```

Figure 3.19 Structure of an ASN.1 Macro

```
OBJECT-IDENTITY MACRO
BEGIN
TYPE NOTATION ::=
"STATUS" Status
"DESCRIPTION" Text
ReferPart
VALUE NOTATION ::=
value(VALUE OBJECT IDENTIFIER)
Status ::= "current" | "deprecated" | "obsolete"
ReferPart ::= "REFERENCE" Text | empty
Text ::= "value (IASString)"
END
```

Figure 3.20 OBJECT-IDENTITY Macro [RFC 1902]

```
cs8113          OBJECT-IDENTITY
        STATUS          current
        DESCRIPTION     "A graduate-level network management course
                        offered every fall by College of Computing in
                        Georgia Institute of Technology."
                ::= {csclasses 50}
```

Figure 3.21 Example for OBJECT-IDENTITY Macro

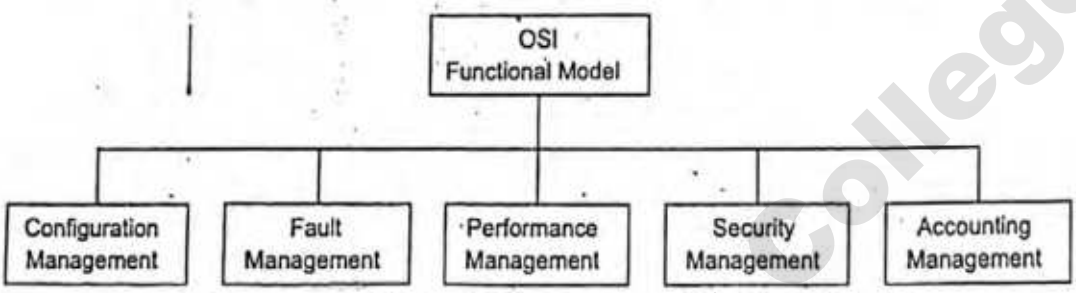


Figure 3.22 Network Management Functional Model

want to formally register the network management course *cs8113* under the object descriptor *csclasses* as the 50th subnode. We can specify an ASN.1 OBJECT-IDENTITY macro shown in Figure 3.21. The object identifier *cs8113* has a value {csclasses 1}. Its status is current and has a description explaining the course offering.

3.9 FUNCTIONAL MODEL

The functional model component of an OSI model addresses user-oriented applications. They are formally specified in the OSI model and are shown in Figure 3.22. The model consists of five models: configuration management, fault management, performance management, security management, and accounting management. Part III of the book is devoted to the application aspects of network management.

Configuration management addresses the setting and changing of configurations of networks and network components. Relevant management information is embedded in managed objects, such as switches, hubs, bridges, and routers. Configuration management involves setting up these parameters. For example, alarm thresholds could be set to generate alarms when packet loss exceeds a defined value. Information on the object name and contact person to be contacted when the component fails could be entered in the management agent. The configuration data are gathered automatically by, and are stored in, the NMS at the network operations center (NOC). NMS displays in real-time the configuration of the network and its status.

Fault management involves detection and isolation of the problem causing the failure in the network. An NMS constantly monitors and displays in real-time major and minor alarms based on the severity of failures. Restoration of service is done as soon as possible and it could involve reconfiguration of the network, which is part of configuration management. In several failure situations, the network could do this automatically. This network feature is called self-healing. In other situations, restoration of service

does not include fixing the cause of the problem. A trouble ticket is generated and followed up for resolution of the problem using a trouble ticket administration system.

This is the trouble ticket administration of fault management and is used to track problems in the network. All problems—including non-problems—are to be tracked until resolved. Periodic analysis of the data, which are maintained in a database, is done to establish patterns of the problems for follow-up action. There are trouble-tracking systems to automate the tracking of troubles from the automatic generation of a trouble ticket by an NMS to the resolution of the problem.

Performance management is concerned with the performance behavior of the network. The status of the network is displayed by a network-monitoring system that measures the traffic and performance statistics on the network. Network statistics include data on traffic volume, network availability, and network delay. Traffic data can be captured based on the traffic volume in various segments of the network. Data need to be gathered by the NOC and updated in a timely fashion in order to administer performance management. Any configuration changes needed to relieve temporary congestion in traffic are made by the NOC. Permanent relief is engineered by the addition of equipment and facilities as well as policy changes. Performance-monitoring tools can gather statistics of all protocol layers. We can analyze the various application-oriented traffic such as Web traffic, Internet mail, file transfers, etc. The statistics on applications could be used to make policy decisions on managing the applications. Performance data on availability and delay are useful for tuning the network to increase the reliability and to improve its response time.

Security management covers a broad range of security aspects. It involves physically securing the network, access to the network resources, and secured communication over the network. A security database is established and maintained by the NOC for access to the network and network information. Any unauthorized access to the network resources generates an alarm on the NMS at the NOC. Firewalls are implemented to protect corporate networks and network resources from being accessed by unauthorized personnel and programs, including virus programs. Secured communication is concerned with the tampering of information as it traverses the network. The content of the information should neither be accessed nor altered by unauthorized personnel. Cryptography plays a vital part in security management.

Accounting management administers cost allocation of the usage of network. Metrics are established to measure the usage of resources and services provided. Traffic data gathered by performance management serve as input to this process.

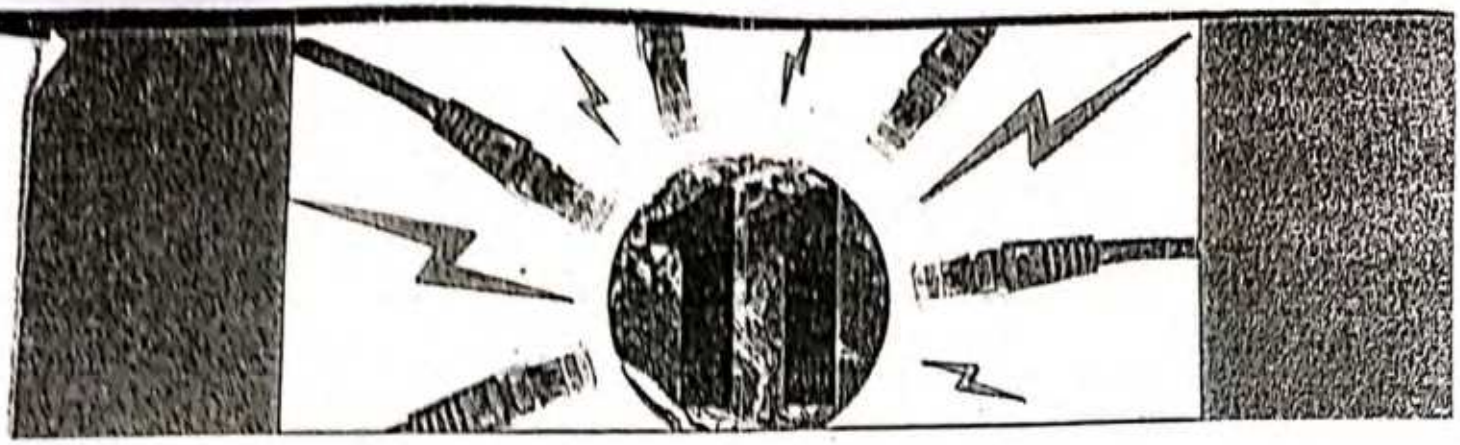
Another dimension of application management is concerned with service and business management, which we discuss in Chapter 7. Service and business management is directed toward service providers, in order for them to accomplish customer satisfaction and to ensure the profitability of business. The traffic statistics, trouble ticket administration data, and accounting management results are inputs to service and business management.

Summary

The foundations of standards, models, and language needed to delve into the study of network management have been addressed in this chapter. These are the four network management models—OSI, Internet, Telecommunications Network Management, and IEEE 802—and a fifth emerging one using Web technology.

The OSI management model categorizes the four functions of network management into four models. They are configuration, information, communication, and application functions. Each of these has been addressed in detail. Some parts of the OSI model are applicable to the other three management models.

4. A proxy server configuration (Figure 6.46) is used to manage SNMPv1 network elements by an SNMPv2 network management system.
   (a) What TMN function does a proxy server play in an NMS environment?
   (b) Identify the interfaces of the proxy server to the network manager and network elements.
5. In Figure 10.19, identify all:
   (a) TMN reference points
   (b) TMN interfaces
6. Associate M1 through M5 interfaces in ATM management (Figure 10.9 and Figure 10.10) with TMN reference points and TMN service interfaces.
7. CMISE services are listed in Table A.3. Map these services, wherever possible, to SNMPv1 services.
8. Repeat Exercise 7 comparing CMISE and SNMPv2 services.
9. TMN can be applied to ATM switch management using either SNMP or CMIP specifications. Research the ATM Forum specifications referenced in Table 12.3 and identify the OBJECT IDENTIFIERS for the two modules, *atmfM4CmipNEView* and *atmfM4Snmpview*.
10. The ATM objects are defined under the node *informationModule(0)*, which is the subclass of *atmfCmipNEView*. Five managed object classes are defined under the *informationModule*, which are *atmfM4ObjectClass*, *atmfM4Package*, *atmfM4Attribute*, *atmfM4NameBinding*, and *atmfM4Action*. Draw a naming tree for these, explicitly identifying the ObjectID.
1. Figure 10.18 shows mapping of eTOM Level 2 processes-to-M.3400 Configuration Function. Do similar mapping of eTOM Level 2 processes-to-M.3400 specifications for:
   (a) Fault Management
   (b) Performace Management
   (c) Accounting Management



Network Management Applications

OBJECTIVES

- Network management and system management
- Network management
  - Configuration
  - Fault
  - Performance
  - Security
  - Accounting
- Configuration management
  - Configuration management
  - Service/network provisioning
  - Inventory management
- Fault management
  - Fault detection and isolation
  - Correlation techniques for root cause analysis
- Performance management
  - Performance metrics
  - Data monitoring
  - Problem isolation
  - Performance statistics
- Security management
  - Security policies and procedures
  - Security threats
  - Firewall
  - Cryptography: keys, algorithms, authentication, and authorization schemes
  - Secure message transfer methods
- Accounting management
- Report management
- Policy-based management
- Service level management
  - Quality of service, QoS
  - Service level agreement, SLA

The management of networked information services involves management of network and system resources. OSI defines network management as a five-layer architecture. We have extended the model to include system management and have presented the integrated architecture in Figure 11.1. At the highest level of TMN are the functions associated with managing the business. business management.

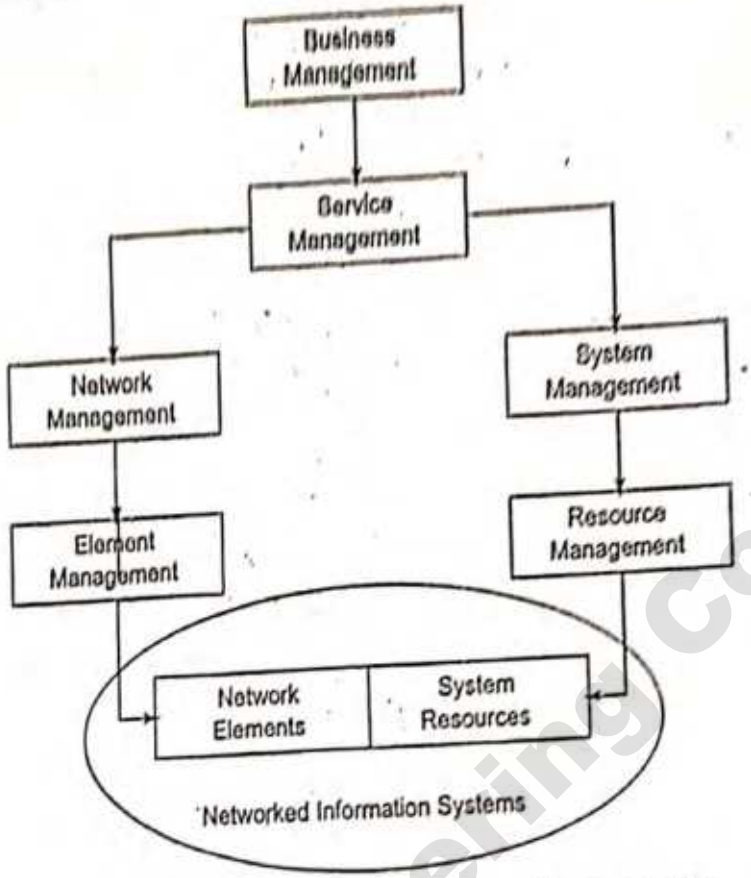


Figure 11.1 Network and System Management

This applies to all institutions, be it a commercial business, educational institute, telecommunications service provider, or any other organization that uses networked systems to manage their business.

An institution is a business that provides either a product or service. In either case, there are service considerations. For example, a product-oriented business has to be concerned with customer service as well as internal services. The service management for a service provider, including a telecommunication service provider, is an absolute necessity. Please see Section 10.8, Figure 10.14 for some of the service applications.

The third layer of TMN deals with network management or system management. Network management manages the global network by aggregating and correlating data obtained from the element management systems. Likewise, the system management aggregates and coordinates system resources by acquisition of data from the resource management systems. The complementary functions of network and system management manage the networked information system composed of network elements and system resources.

Our focus in this chapter will be on network management applications. As we learned in Chapter 3, there are five different categories of applications: configuration management, fault management, performance management, security management, and account management. Others [Leinwand

and Conroy, 1996] have treated the five categories of applications and presented simple and complex tools to manage them.

The subject of configuration management may be looked at not only from an operational viewpoint, but also from engineering and planning viewpoints. In our treatment of configuration management in Section 11.1, we have included network provisioning and inventory management. This is in addition to the configuration of network topology, which is part of traditional network management.

Fault management involves detection of a fault as it occurs in the network, and subsequently locating the source of the problem. We should finally isolate the root cause of the problem. This is covered in Section 11.2.

It is harder to define performance of a network in quantitative terms than in qualitative terms. For example, when a user observes that the network performance is slow, we need to define what slowness is, and which segment of the network is slow. On the other hand, it could be that the application, which could be running on a server, is behaving slowly. We discuss performance management in Section 11.3. We will discuss performance metrics and learn how to monitor a network for performance. Performance statistics play a very important part in network management, and several system tools available for gathering statistics will be covered.

When a fault occurs in a network, either due to failure of a component or due to performance, it may manifest itself in many places. Thus, from a centralized management system, we observe alarms coming from multiple locations. Correlating these alarm events and finding the root cause of the problem is a challenge. We will discuss the various correlation technologies in Section 11.4.

Security in network is concerned with preventing illegal access to information by unauthorized personnel. It involves not only technical issues, but also establishment of well-defined policies and procedures. We will discuss the various issues associated with authentication and authorization in Section 11.5. We will also deal with the establishment of secure (i.e., without illegal monitoring and manipulation) communication between the source and the receiver.

The business health of an institution or corporation depends on well-maintained accounting management and reporting. Reports for management have a different purpose from that of reports generated for day-to-day network operation. There are also reports needed for the user to measure the quality of service to be provided by service level agreement (SLA). These are covered in Sections 11.6 and 11.7

We have addressed the five layers of management with network elements being at the lowest level and business management being at the top. Element management at the second layer maintains the network. Network management at the third level and service management at the fourth level are based not just on technical issues but also on policy issues. Once policies are established, some of them could be implemented in the system. For example, if a network is congested due to heavy traffic, should the

network parameters be automatically adjusted to increase bandwidth, or should traffic into the network be decreased. A policy decision that has been made on this can be implemented as part of the management system. We will discuss this in Section 11.8.

Service level management is an important aspect of network and system management. It goes beyond managing resources. It is concerned with the SLA between the customer and the service provider regarding the quality of service of network, systems, and business applications. This is covered in Section 11.9.

11.1 CONFIGURATION MANAGEMENT

Configuration management in network management is normally used in the context of discovering network topology, mapping the network, and setting up the configuration parameters in management agents and management systems. However, as discussed in Section 1.9 and shown in Figure 1.21, network management in the broad sense also includes network provisioning. Network provisioning includes network planning and design and is considered part of configuration management.

11.1.1 Network Provisioning

Network provisioning, also called circuit provisioning in the telephone industry, is an automated process. The design of a trunk (circuit from the originating switching center to the destination switching center) and a special service circuit (customized for customer specifications) is done by application programs written in operation systems. Planning systems and inventory systems are integrated with design systems to build a system of systems. Thus, a circuit designed for the future automatically derives its turn-up date from the planning system and ensures that the components are available in the inventory system. Likewise, when a circuit is to be disconnected, it is coordinated with the planning system and the freed-up components are added to the inventory system. Thus, the design system is made aware of the availability of components for future designs.

An example of a circuit provisioning system is a system of systems developed by Bell System (before it was split), called Trunk Integrated Record Keeping System (TIRKS). TIRKS is used in automated circuit provisioning of trunks. A trunk is a logical circuit between two switching offices and it traverses over many facilities. TIRKS is an operations system in the context of Telecommunications Management Network (TMN) that we dealt with in Chapter 10. Given the requirements of a trunk, such as transmission loss and noise, type of circuit, availability date, etc., as input to the system, the system automatically designs the components of the trunk. The designed circuit will identify transmission facilities between switching offices and equipment in intermediate and end offices. The equipment will be selected based on what would be available in the future when the circuit needs to be installed.

Network provisioning in a computer communications network has different requirements. Instead of circuit-switched connections, we have packet-switched paths for information to be transmitted from the source to the destination. In a connectionless packet-switched circuit, each packet takes an independent path, and the routers at various nodes switch each packet based on the load in the links. The links are provisioned based on average and peak demands. In store-and-forward communication, excess packets can be stored in buffers in routers or retransmitted in the event the packets are lost or discarded. In the connection-oriented circuit requirements, permanent and switched virtual circuit demands need to be accommodated for end-to-end demands on the various links. Network provisioning for packet-switched network is based on performance statistics and quality of service requirements.

Network provisioning in broadband wireless area network (WAN) communication using ATM technology is more complex. The virtual-circuit concept is always used and has to be taken into account in the provisioning process. The switches are cell-based, in contrast to frame-based packet switching. Each ATM switch has knowledge of the virtual path–virtual circuit (VP–VC) of each session connection only to the neighboring nodes and not end-to-end. Each ATM switch vendor has built their proprietary assignment of VP–VC for end-to-end design into the ATM switch. The architecture of end-to-end provisioning of ATM circuits could be either centralized or distributed, and is based on whether the circuit is a permanent virtual circuit (PVC) or a switched virtual circuit (SVC). Commercial products, which provision PVCs across multiple vendor products, have recently been introduced in the market.

11.1.2 Inventory Management

We have addressed the importance of inventory management in circuit provisioning. An efficient database system is an essential part of the inventory management system. We need to be aware of all the details associated with components, which should be accessible using different indices. Some of the obvious access keys are the component description or part number, components that match a set of characteristics, components in use and in spare, and components to be freed-up for future use.

In Section 11.1.1 we cited the example of TIRKS, which is a system of systems. Two of the systems that TIRKS uses are equipment inventory (E1) and facilities inventory (F1). The E1 system has an inventory of all equipment identifying what is currently available and what will become available in the future with dates of availability. Similar information is maintained on facilities by the F1 system. With such a detailed inventory system, TIRKS can anticipate circuit provisioning for the future with components that would be available.

Legacy inventory management systems use hierarchical and scalar-based database systems. Such databases limit the addition of new components or extend the properties of existing components by adding new fields. These limitations can be removed by using relational database technology. Further, new NMSs, such as in OSI CMIP and Web-based management, use object-oriented technology. These manage object-oriented managed objects. An object-oriented relational database is helpful in configuration and inventory management in such an environment.

11.1.3 Network Topology

Network management is based on knowledge of network topology. As a network grows, shrinks, or otherwise changes, the network topology needs to be updated automatically. This is done by the discovery application in the NMS as discussed in Section 9.4.4. The discovery process needs to be constrained as to the scope of the network that it discovers. For example, the *arp* command can discover any network component that responds with an IP address, which can then be mapped by the NMS. If this includes workstations that are turned on only when they are in use, the NMS would indicate failure whenever they are off. Obviously, that is not desirable. In addition, some hosts should not be discovered for security reasons. These should be filtered out during the discovery process so the discovery application should have the capability to set filter parameters to implement these constraints.

Autodiscovery can be done using the broadcast ping on each segment and following up with further SNMP queries to gather more details on the system. The more efficient method is to look at the ARP cache in the local router. The ARP cache table is large and contains the addresses of all the recently communicated hosts and nodes. Using this table, subsequent ARP queries could be sent to other routers. This process is continued until information is obtained on all IP addresses defined by the scope of the

autodiscovery procedure. A map, showing network topology, is presented by the autodiscovery procedure after the addresses of the network entities have been discovered.

The autodiscovery procedure becomes more complex in the virtual local area network (LAN) configuration. Figure 11.2 shows the *physical* configuration of a conventional LAN. The router in the figure can be visualized as part of a backbone (not shown). There are two LAN segments connected to the router, Segment A and Segment B. They are physically connected to two physical ports in the router (i.e., there is one port for each segment used on the interface card). They are identified as Port A and Port B, corresponding to Segment A and Segment B, respectively. Both LANs are Ethernet LANs and use hub configuration. Two hosts, A1 and A2, are connected to Hub 1 on LAN segment A and two hosts, B1 and B2, are connected to Hub 2 on LAN Segment B.

Figure 11.3 shows the *logical* configuration for Figure 11.2. The logical configuration is what the autodiscovery process detects. It is very similar to the physical configuration. Segment A corresponds to LAN on Hub 1 with the hosts A1 and A2. It is easy to conceptually visualize this and easy to configure.

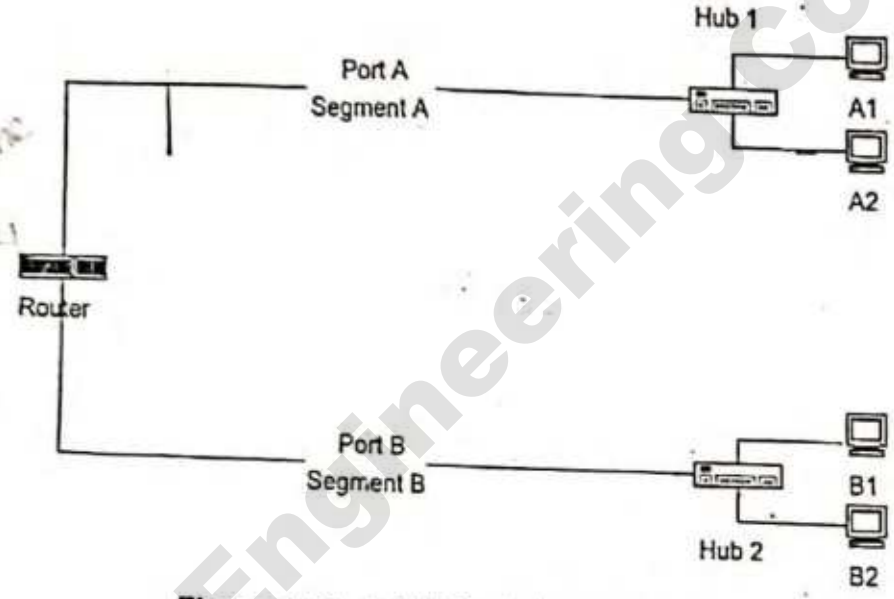


Figure 11.2 LAN Physical Configuration

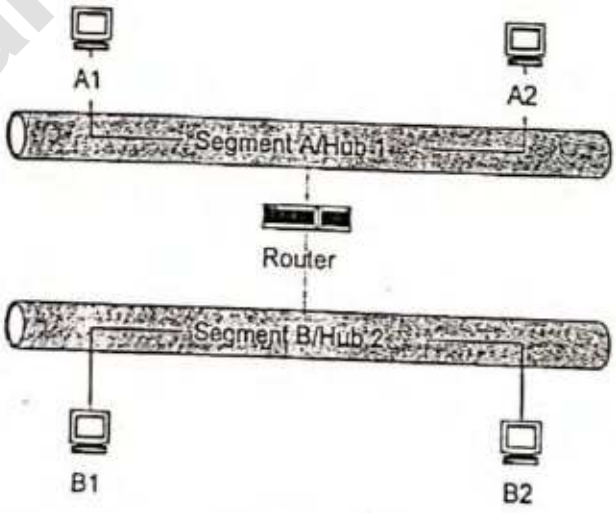


Figure 11.3 Logical Configuration of Two LAN Segments

Let us now contrast Figure 11.2 with Figure 11.4, which shows the *physical* configuration of virtual LANs (VLAN). We notice that only one physical port, Port A, is used in the router, for the in the case of a traditional LAN. Hosts A1 and A2 are configured to be on VLAN 1, and hosts B1 B2 are configured to be on VLAN 2. Although VLAN grouping can be done on different criteria us assume that it is done on port basis on the switch. Thus, the two ports marked Segment A on switch are grouped as VLAN 1. The other two ports, marked Segment B, are grouped as VLAN Thus, Segment A corresponds to VLAN 1 and Segment B corresponds to VLAN 2. We observe VLAN 1 and VLAN 2 are spread across the two physical hubs, Hub 1 and Hub 2. With a layer-2 bridge network, the VLAN network is efficient. As IEEE 802.3 standards are established and widely adopted this configuration has been deployed more and more, along with a backbone network.

The *logical* view of the physical VLAN configuration shown in Figure 11.4 is presented in Figure 11.5. We see that Hosts A1 and A2 still belong to Segment A, but are on different hubs. Likewise Hosts B1 and B2 belong to Segment B. The autodiscovery process would not detect the physical hubs that are identified in Figure 11.5. In many situations, the switch would also be transparent, as there are no IP addresses associated with switch ports. Consequently, it would be harder to associate the logical configuration with the physical configuration.

This makes the network management task a complex one. First, two separate maps must be maintained on an on-going basis as changes to the network are made. Second, when a new component is added and autodiscovered by the system, a manual procedure is needed to follow up on the physical configuration.

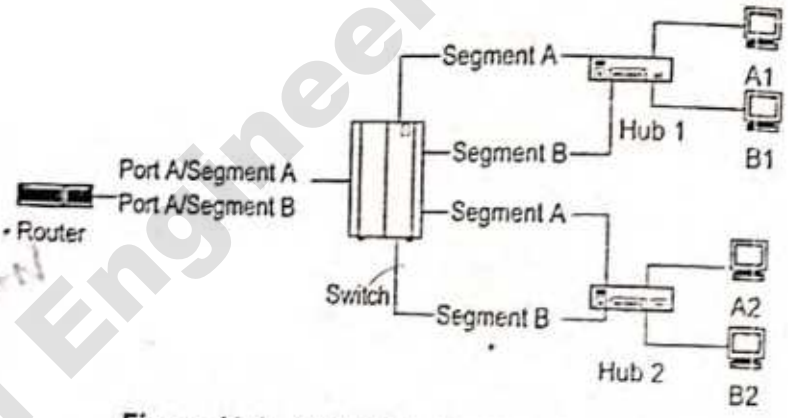


Figure 11.4 VLAN Physical Configuration

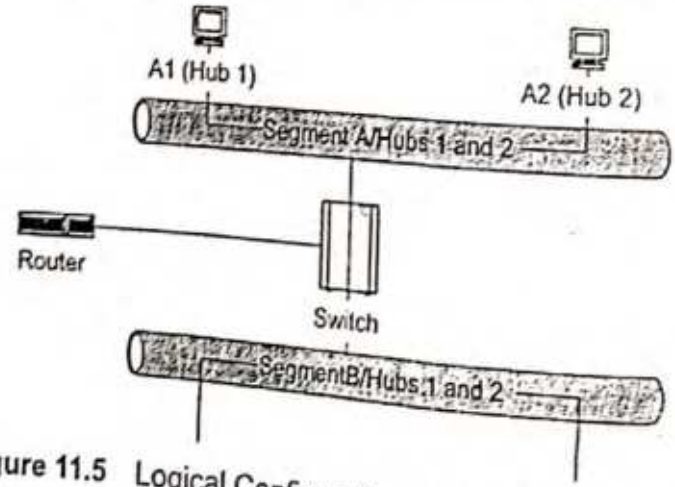


Figure 11.5 Logical Configuration of Two VLAN Segments

In the example above, we talked about grouping of VLAN based on the ports on the switches. We could also group VLAN based on MAC address, IP address, or protocol type. Grouping by IP address has some benefits in the management of VLAN network. The logical grouping of components based on IP network segments makes sense. In addition, as a policy the *sysLocation* entity in a system group should be filled in for easier management.

11.2 FAULT MANAGEMENT

Fault in a network is normally associated with failure of a network component and subsequent loss of connectivity. Fault management involves a five-step process: (1) fault detection, (2) fault location, (3) restoration of service, (4) identification of root cause of the problem, and (5) problem resolution. The fault should be detected as quickly as possible by the centralized management system, preferably before or at about the same time as when the users notice it. Fault location involves identifying where the problem is located. We distinguish this from problem isolation, although in practice it could be the same. The reason for doing this is that it is important to restore service to the users as quickly as possible, using alternative means. The restoration of service takes a higher priority over diagnosing the problem and fixing it. However, it may not always be possible to do this. Identification of the root cause of the problem would be a complex process, which we will go into greater depth soon. After identifying the source of the problem, a trouble ticket can be generated to resolve the problem. In an automated network operations center, the trouble ticket could be generated automatically by the NMS.

11.2.1 Fault Detection

Fault detection is accomplished using either a polling scheme (the NMS polling management agents periodically for status) or by the generation of traps (management agents based on information from the network elements sending unsolicited alarms to the NMS). An application program in NMS generates the ping command periodically and waits for response. Connectivity is declared broken when a pre-set number of consecutive responses are not received. The frequency of pinging and the preset number for failure detection may be optimized for balance between traffic overhead and the rapidity with which failure is to be detected.

The alternative detection scheme is to use traps. For example, the generic trap messages *linkDown* and *egpNeighborLoss* in SNMPv1 can be set in the agents giving them the capability to report events to the NMS with the legitimate community name. One of the advantages of traps is that failure detection is accomplished faster with less traffic overhead.

11.2.2 Fault Location and Isolation Techniques

Fault location using a simple approach (we will look at the complex approach using the correlation technology in Section 11.4) would be to detect all the network components that have failed. The origin of the problem could then be traced by walking down the topology tree where the problem starts. Thus, if an interface card on a router has failed, all managed components connected to that interface would indicate failure.

After having located where the fault is, the next step is to isolate the fault (i.e., determine the source of the problem). First, we should delineate the problem between failure of the component and the physical link. Thus, in the above example, the interface card may be functioning well, but the link to the interface may be down. We need to use various diagnostic tools to isolate the cause.

Let us assume for the moment that the link is not the problem but that the interface card is. We then proceed to isolate the problem to the layer that is causing it. It is possible that excessive packet loss is causing disconnection. We can measure packet loss by pinging, if pinging can be used. We can query the various Management Information Base (MIB) parameters on the node itself or other related nodes to further localize the cause of the problem. For example, error rates calculated from the interface group parameters, *ifInDiscards*, *ifInErrors*, *ifOutDiscards*, and *ifOutErrors* with respect to the input and output packet rates, could help us isolate the problem in the interface card.

The ideal solution to locating and isolating the fault is to have an artificial intelligence solution. By observing all the symptoms, we might be able to identify the source of the problem. There are several techniques to accomplish this, which we will address in Section 11.4.

11.3 PERFORMANCE MANAGEMENT

We have already addressed performance management applications directly and indirectly under the various headings. We discussed two popular protocol analyzers, Sniffer and NetMetrix, in Chapter 9. In Section 9.2, we used the protocol analyzer as a system tool to measure traffic monitoring on Ethernet LANs, which is in the realm of performance management. We looked at load monitoring based on various parameters such as source and destination addresses, protocols at different layers, etc. We addressed traffic statistics collected over a period of from hours to a year using the Multi Router Traffic Grapher (MRTG) tool in Section 9.2.4. The statistics obtained using a protocol analyzer as a remote monitoring (RMON) tool was detailed in the case study in Section 8.6. We noticed how we were able to obtain the overall trend in Internet-related traffic and the type of traffic.

Performance of a network is a nebulous term, which is hard to define or quantify in terms of global metrics. The purpose of the network is to carry information and thus performance management is really (data) traffic management. It involves the following: data monitoring, problem isolation, performance tuning, analysis of statistical data for recognizing trends, and resource planning.

11.3.1 Performance Metrics

The parameters that can be attributed to defining network performance on a global level are throughput, response time, network availability, and network reliability. The metrics on these are dependent on what, when, and where the measurements are made. Real-time traffic performance metrics are latency (i.e., delay) and jitter, which are addressed in Section 11.3.4.

These macro-level parameters can be defined in terms of micro-level parameters. Some of the parameters that impact network throughput are bandwidth or capacity of the transmission media, its utilization, error rate of the channel, peak load, and average load of the traffic. These can be measured at specific points in the network. For example, bandwidth or capacity will be different in different segments of the network. An Ethernet LAN with a capacity of 10 Mbps can function to full capacity with a single workstation on it, but reaches full capacity with a utilization factor of 30–40% when densely populated with workstations. This utilization factor can further be defined in terms of collision rate, which is measurable. In contrast, in a WAN, the bandwidth is fully utilized except for the packet overhead.

The response time of a network not only depends on the throughput of the network, but also on the application; in other words, it depends on both the network and system performance. Thus, in a client–server environment, the response time as seen by the client could be slow either due to the server being heavily used, or the network traffic being overloaded, or a combination of both. According to Feldmeir [1997], "the application responsiveness on the network, more than any other measure, reflects whether

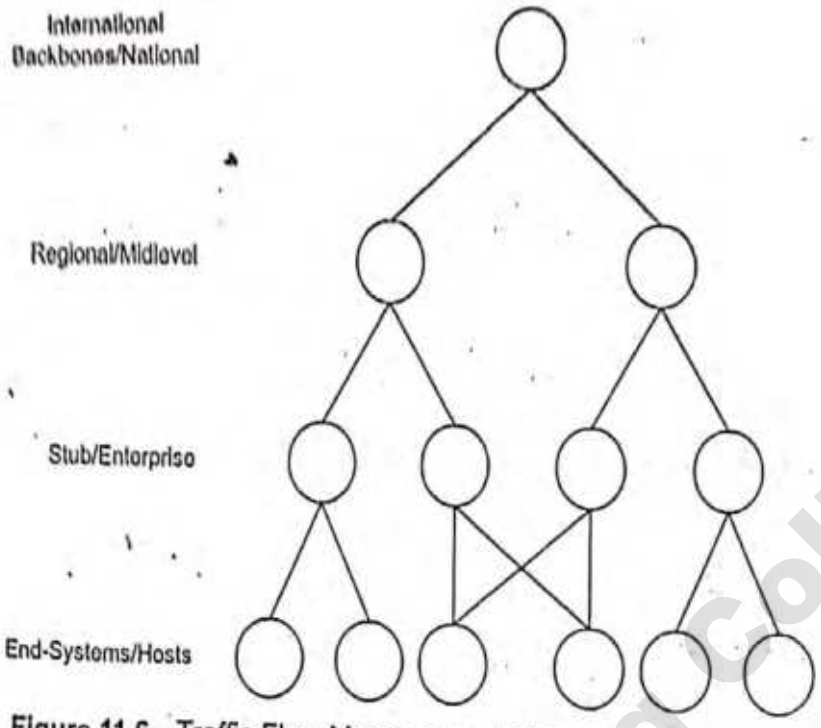


Figure 11.6 Traffic Flow Measurement Network Characterization

the network is meeting the end users' expectations and requirements." He defines three types of metrics to measure application responsiveness: application availability, response time between the user and the server, and the burst frame rate, which is the rate at which the requested data arrive at the user station.

IETF Network Working Group has developed several Request for Comments (RFCs) on traffic flow measurement. RFC 2063 defines the architecture for the measurement and reporting of network traffic flows. The network is characterized as traffic passing through four representative levels, as shown in Figure 11.6. Backbone networks are those that are typically connected to other networks, and do not have individual hosts connected to them. A regional network is similar to a backbone, but smaller. It may have individual hosts connected to it. Regional hosts are subscribers to a backbone network. Stub/enterprise networks connect hosts and LANs and are subscribers to regional and backbone networks. End systems or hosts are subscribers to all of the above.

The architecture defines three entities for traffic flow measurements: meters, meter readers, and managers. Meters observe network traffic flows and build up a table of flow data records for them. Meter readers collect traffic flow data from meters. Managers oversee the operation of meters and meter readers. RFC 2064 defines the MIB for the meter. RFC 2123 describes NeTraMet, an implementation of flow meter based on RFC 2063 and RFC 2064.

11.3.2 Data Monitoring

Data monitoring in the network for abnormal performance behavior, such as high collision rate in Ethernet LAN, excessive packet drop due to overload, etc., are detected by traps generated by agents and RMON. Performance-related issues are detected primarily using trap messages generated by RMON probes. Thresholds are set for important SNMP parameters in the RMON, which then generate alarms when the pre-set thresholds are crossed. For example, the parameters in the alarm group and the event group in RMON MIB [RFC 1757] may be set for the object identifier to be monitored. The time interval over which the data are to be collected for calculation and the rising and falling thresholds are specified

In addition, the community names are set for who would receive the alarm. Although we
such alarms under performance category, they could also be defined under fault category as it
Section 9.4.6.

NMSs generally report all events selected for display including alarms. Alarms are set for
and the icon changes color based on the criticality. Depending on the implementation, the alarm is
automatically cleared when the alarm condition clears or is manually cleared by an operator. The
case is useful for alerting the operations personnel as to what happened.

11.3.3 Problem Isolation

Problem isolation for performance-related issues depends on the type of problem. As we have
before, a high percentage of packet loss will cause loss of connectivity, which could be
In this situation, monitoring the packet loss over an extended period will isolate the problem.
example is the performance problem associated with large delay. This may be attributable to an
drop of packets. We can identify the source of the packet delay from a route-tracing procedure and
probe for the packet discards at that node. Refer to [Rose and McCloghrie, 1995] for a detailed
of the performance degradation cases in various components and media.

As in fault management, problems could occur at multiple locations simultaneously. These
be reported to the central management system as multiple independent events although they may
correlated. For example, an excessive drop in packets in one of the links may switch the traffic to
alternate route. This could cause an overload in that link, which will be reported as an alarm. A
sophisticated approach using the correlation technology is again required here, which we will
in Section 11.4.

11.3.4 Performance Statistics

Performance statistics are used in tuning a network, validating of SLA, which will be covered
Section 11.9, analyzing use trends, planning facilities, and functional accounting. Data are
by means of an RMON probe and RMON MIB for statistics. Statistics, to be accurate, require
amounts of data sampling, which create overhead traffic on the network and thus impact its
mance. One of the enormous benefits of using RMON probe for collecting statistics is that it can
done locally without degrading the overall performance of the network. An RMON MIB contains
history and statistics groups (see Section 8.4) for various media and can be used efficiently to collect
relevant data and store them for current or future analysis.

One application of the results obtained from performance statistics is to tune the network for
performance. For example, two segments of the network may be connected by a gateway and
intersegment traffic could produce excessive performance delay. Error statistics on dropped packets
the gateway interfaces would manifest this problem. The solution to resolve this problem is to
the bandwidth of the gateway by either increasing its capacity or by adding a second gateway
the segments. Of course, adding the extra gateway could cause configuration-related problems
hence reconfiguration of traffic may be needed.

Various error statistics at different layers are gathered to measure the quality of service and
performance improvement, if needed. Some of the other performance parameters that can be
by monitoring network statistics are bandwidth of links, utilization of links, and controlling
to-average ratio of inherently bursty data traffic. In addition, traffic utilization can be

redistributing the load during the day, with essential traffic occupying the busy hours and non-essential traffic the slack hours, with the latter being store and forward.

An important performance criterion in real-time traffic in broadband service is the latency or delay caused by dispersion in large bandwidth signal. This affects the quality of service due to performance degradation.

Another important statistic, especially in real-time broadband services, is the variation-in-network delay, otherwise known as **jitter**. This impacts the quality of service (QoS) guaranteed to the customer by the SLA. For example, in a cable modem, a managed object *docsQoSServiceClassMaxJitter* (see the DOCSIS Quality of Service MIB in Table 13.3) is used to monitor the jitter.

Another performance application is validation of SLA between the service user and the service provider. The SLA may require limiting input to the service provider network. If the packet rate is tending toward the threshold of SLA, one may have to control the bandwidth of the access to the service provider network. This can be achieved by implementing application interfaces that use algorithms such as the leaky bucket or the token bucket [Tanenbaum, 1996]. The leaky bucket algorithm limits the maximum output data rate, and the token bucket algorithm controls its average value. Combining the two, we can tune the peak-to-average ratio of the output. Some ATM switches have such interfaces built into them, which are easily tunable. This would be desirable if a service provider's pricing is based on peak data rate usage instead of average rate.

Performance statistics are also used to project traffic use trends on traffic use and to plan future resource requirements. Statistical data on traffic are collected and periodic reports are generated to study use trends and project needs. Thus, trend analysis is helpful for future resource planning.

Statistics can be gathered, as we saw in Section 9.2, on the network load created by various users and applications. These can be used to do functional accounting so that the cost of operation of a network can be charged to the users of the network or at least be justified.

14 EVENT CORRELATION TECHNIQUES

We have illustrated some simple methods to diagnose and isolate the source of a problem in fault and performance management. When a centralized NMS receives a trap or a notification, it is called **receiving an event**. A single problem source may cause multiple symptoms, and each symptom detected is reported as an independent event to the management system. Obviously, we do not want to treat each event independently and act to resolve it. Thus, it is important that the management system correlates all these events and isolates the root cause of the problem. The techniques used for accomplishing this are called **event correlation techniques**.

There are several correlation techniques used to isolate and localize fault in networks. All are based on (1) detecting and filtering of events, (2) correlating observed events to isolate and localize the fault either topologically or functionally, and (3) identifying the cause of the problem. In all three cases, there is intelligence or reasoning behind the methods. The reasoning methods distinguish one technique from another.

We will discuss six approaches to correlation techniques. They are (1) rule-based reasoning, (2) model-based reasoning, (3) case-based reasoning, (4) codebook, (5) state transition graph model, and (6) finite state machine model. See Lewis [1999] for a detailed comparison of the various methods.

Rule-Based Reasoning

Rule-based reasoning (RBR) is the earliest form of correlation technique. It is also known by many other names such as rule-based expert system, expert system, production system, and blackboard system. It has a knowledge base, working memory, and an inference engine, as shown in Figure 11.7 [Cronk *et al.*, 1988; Lewis, 1994]. The three levels representing the three components are the knowledge level, the data level, and the control level, respectively. Cronk *et al.* are also a good source for review of network applications of RBR. The knowledge base contains expert knowledge as to (1) definition of a problem in the network and (2) action that needs to be taken if a par-ticular condition occurs. The knowledge base information is rule-based in the form of **if–then** or **condition–action**, containing rules that indicate which operations are to be performed when. The working memory contains—as working memory elements—the topological and state information of the network being monitored. When the network goes into a faulty state, it is recognized by the working memory. The inference engine, in cooperation with the knowledge base, compares the current state with the left side of the rule-base and finds the closest match to output the right side of the rule. The knowledge base then executes an action on the working memory element.

In Figure 11.7, the rule-based paradigm is interactive between the three components and is iterative. There are several strategies for the rule-based paradigm. A specific strategy is implemented in the inference engine. Choosing a specific rule, an action is performed on the working memory element, which could then initiate another event. This process continues until the correct state is achieved in the working memory.

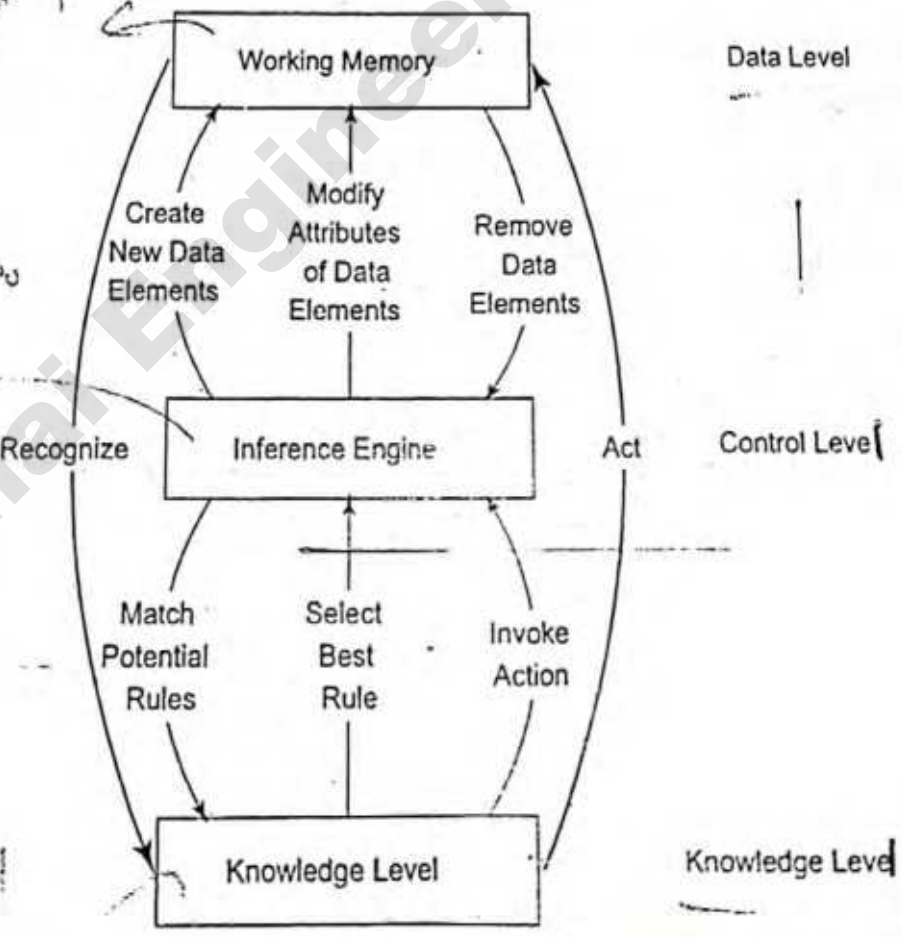


Figure 11.7 Basic Rule-Based Reasoning Paradigm

Rules are made up in the knowledge base from the expertise of the experts in the field. The rule is an exact match and the action is very specific. If the antecedent in the rule does not match, the paradigm breaks and it is called "brittle." However, it can be fixed by adding more rules, which would increase the database size and degrade the performance, referred to as **knowledge acquisition bottleneck**. There is an exponential growth in size as the number of working memory elements grows.

In addition, the action is specific, which could cause unwanted behavior. For example, we can define the alarm condition for packet loss as follows:

```
If packet loss < 10%              alarm green
If packet loss => 10% < 15%       alarm yellow
If packet loss => 15%             alarm red
```

The left side conditions are the working memory elements, which if detected would execute the appropriate rule defined in the rule-base. As we can see, this could cause the alarm condition to flip back and forth in boundary cases. An application of fuzzy logic is used to remedy this problem [Lewis, 1994], but it is harder to implement.

The RBR is used in Hewlett-Packard OpenView Element Management Framework [Hajela, 1996]. Figure 11.8 is an adaptation of the scenario from [Hajela, 1996] to illustrate an implementation of RBR. It shows a four-layer network. Backbone Router A links to Router B. Hub C, connected to Router B, has four servers, D1 through D4, in its LAN. Without a correlation engine, failure in the interface of Router A will generate an alarm. This fault then propagates to Router B, Hub C, and finally to Servers D1 through D4. It is important to realize that there is a time delay involved in the generation of alarms. In general, propagation of faults and time delay associated with them need to be recognized as such in fault management.

Four correlation rules are specified in Figure 11.9. Rule 0 has no condition associated with it. Rules 1–3 are conditional rules. In order to allow for the propagation time, a correlation window of 20 seconds is set.

The inference engine at the control level interprets the above rules and takes the actions shown in Figure 11.10.

In the above example, it can be observed that only alarm A is sent and others are ignored as long as they arrive within the correlation window. The alarms could even be generated out of sequence.

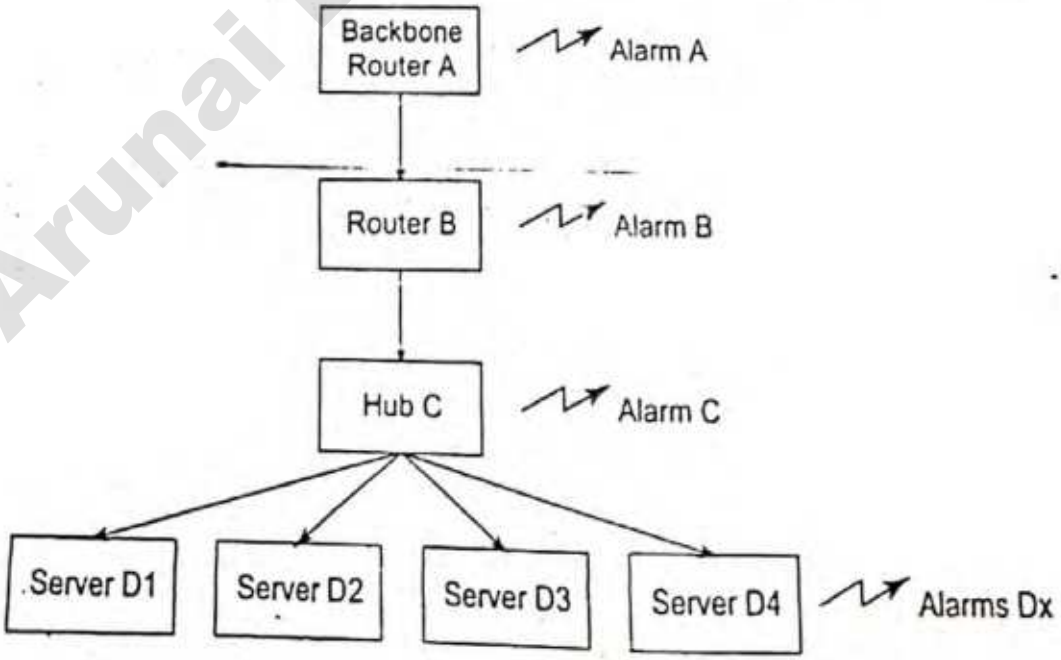


Figure 11.8 RBR-Based Correlation Example Scenario

Rule 0:	Alarm A :	Send root cause Alarm A	
Rule 1	Alarm B	If Alarm A present	Related to A and ignore
Rule 2	Alarm C	If Alarm B present	Related to B and ignore
Rule 3	Alarm Dx	If Alarm C present	Related to C and ignore

Correlation window: 20 seconds.

Figure 11.9 Rules Specifications for the Example in Figure 11.8

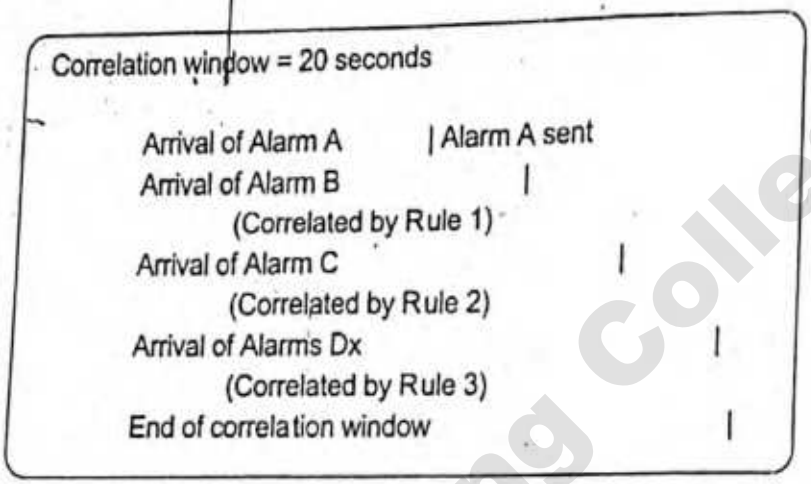


Figure 11.10 Control Actions for the RBR Example of Figure 11.8

However, because of the specification of correlation window size, the inference engine waits befo sending alarms B, C, and Dx.

Several commercial systems have been built using RBR. Some examples are Computer Associat TNG and Tivoli TME.

11.4.2 Model-Based Reasoning

An event correlator based on model-based reasoning is built on an object-oriented model associate with each managed object. A model is a representation of the component it models. The model, in th traditional object-oriented representation, has attributes, relations to other models, and behaviors. Th relationship between objects is reflected in a similar relationship between models.

Let us picture a network of hubs that are connected to a router, as shown in the left half Figure 11.11. The right half shows the corresponding model in the event correlator in the NMS. Th NMS pings every hub and the router (really a router interface to the backbone network) periodically check whether each component is working. We can associate communication between the NMS and managed component as between a model (software object) in the NMS correlator and its counterpart managed object. Thus, in our example, the model of each hub periodically pings its hub and the mod of the router pings the router. As long as all the components are working, no additional operation needed.

If Hub1 fails, it is recognized by the H1 model. Let us assume that the H1 model is programmed wait for lack of response in three consecutive pings. After three pings with no response, the H1 mod suspects a failure of Hub1. However, before it declares a failure and displays an alarm, it analyses its r lation to other models and recognizes that it should query the router model. If the router model respon that the router is working, only then the Hub1 alarm is triggered. If the router model responds that

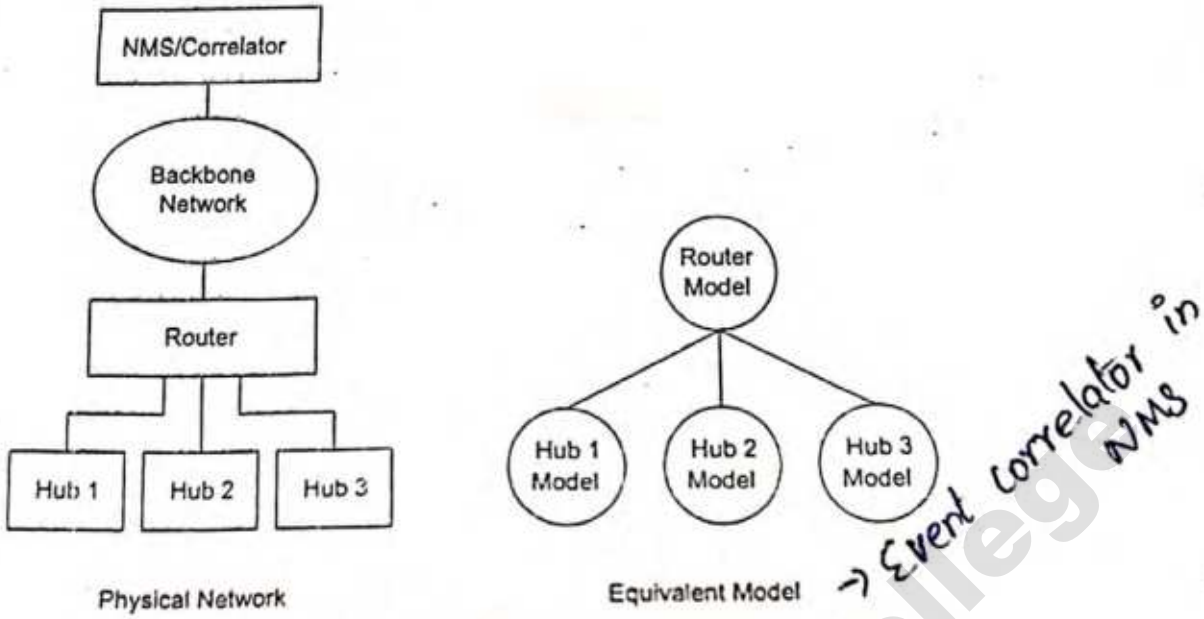


Figure 11.11 Model-Based Reasoning Event Correlator

is not receiving a response from the router, then the Hub1 model deduces that the problem is with the router and not Hub1. At least, it cannot definitively determine a Hub1 failure as long as it cannot communicate with Hub1 because of the router failure.

The above example is modeled after [Lewis, 1999], who presents an interesting scenario of a classroom with teacher. Outside the classroom is a computer network with a router and workstations. Each student is a model (software mirror) of the workstation. The teacher is a model of the router. Each student communicates with his or her real-world counterpart, which is the workstation outside the classroom. The teacher communicates with the router. If a student fails to communicate with his or her workstation, he or she queries the teacher as to whether the teacher could communicate with the teacher's router. Depending on the yes or no answer of the teacher, the student declares a "fail" (yes) or "no-fail" (no) condition, respectively. Model-based reasoning is implemented in Cabletron Spectrum.

11.4.3 Case-Based Reasoning

Case-based reasoning (CBR) overcomes many of the deficiencies of RBR. In RBR, the unit of knowledge is a rule; whereas in CBR, the unit of knowledge is a case [Lewis, 1995]. The intuition of CBR is that situations repeat themselves in the real world; and that what was done in one situation is applicable to others in a similar, but not necessarily identical, situation. Thus, when we try to resolve a trouble, we start with the case that we have experienced before [Kolodner, 1997; Lewis, 1995]. Kolodner treats CBR from an information management viewpoint, and Lewis applies it specifically to network management.

The general CBR architecture is shown in Figure 11.12 [Lewis, 1995]. It consists of four modules: input, retrieve, adapt, and process, along with a case library. The CBR approach uses the knowledge gained before and extends it to the current situation. The former episodes are stored in a case library. If the current situation, as received by the input module, matches one that is present in the case library (as compared by the retrieve module), it is applied. If it does not, the closest situation is chosen by the adapt module, and adapted to the current episode to resolve the problem. The process module takes the appropriate action(s). Once the problem is resolved, the newly adapted case is added to the case library.

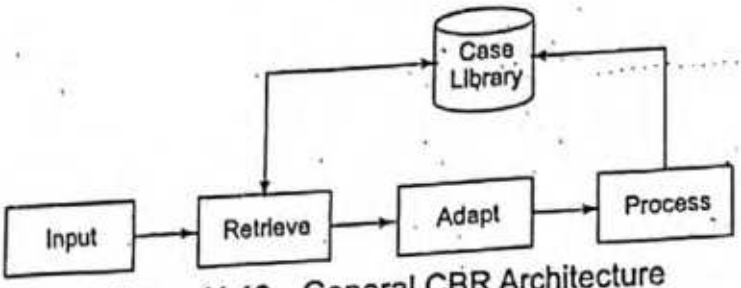


Figure 11.12 General CBR Architecture

Lewis also describes the application of CBR in a trouble-tracking system, CRITTER [Lewis, 1996]. The CRITTER application has evolved into a CBR application for network management named SpectroRx built by Cabletron. When a trouble ticket is created on a network problem, it is compared to similar cases in the case library containing previous trouble tickets with resolutions. The current trouble is resolved by adapting the previous case in one of three ways: (1) parameterized adaptation, (2) abstraction/respecialization adaptation, and (3) critic-based adaptation. The resolved trouble ticket is then added to the case library. We will use the examples given in the reference to illustrate the three adaptation methods.

Parameterized Adaptation. Parameterized adaptation is used when a similar case exists in the case library, but the parameters may have to be scaled to resolve the current situation. Consider the current trouble with file_transfer_throughput, which matches the following trouble ticket in the case library, shown in Figure 11.13.

In the parameterized adaptation of the trouble ticket shown in Figure 11.14, variable F has been modified to F′ and the relationship between network load adjustment variable A′ and F′ remain the same as between A and F. In the default situation, where there is an exact match, F′ and A′ are F and A.

Abstraction/Respecialization Adaptation. Figure 11.15 shows three trouble tickets. The first two are two cases from the case library that matched the current problem we have been discussing. The first option adjusts the network load, and the second option adjusts the bandwidth of the network. The user or the system has the option of adapting either of the two based on restrictions to be placed on adjusting the workload or adjusting the bandwidth. Let us choose the option of not restricting the network load, which implies that we have to increase the bandwidth. We can add this as additional data to the trouble ticket that chooses the bandwidth option and create a new trouble ticket, which is shown as the third trouble ticket in Figure 11.15. This is now added to the case library.

This CBR adaptation is referred to as abstraction/respecialization adaptation. Choosing to adjust bandwidth and not load is a policy decision, which we will discuss in Section 11.8.

Critic-Based Adaptation. The third adaptation, critic-based adaptation, is one where a critic or a craft

Trouble: file_transfer_throughput=F
Additional data: none
Resolution: A=f(F), adjust_network_load=A
Resolution status: good

Figure 11.13 Matching Trouble Ticket for the CBR Example

Trouble: file_transfer_throughput=F'
Additional data: none
Resolution: A'=f(F'), adjust_network_load=A'
Resolution status: good

Figure 11.14 Parameterized Adaptation for the CBR Example

Trouble: file_transfer_throughput=F
Additional data: none
Resolution: A=f(F), adjust_network_load=A
Resolution status: good

Trouble: file_transfer_throughput=F
Additional data: none
Resolution: B=g(F), adjust_network_bandwidth=B
Resolution status: good

Trouble: file_transfer_throughput=F
Additional data: adjust_network_load=no
Resolution: B=g(F), adjust_network_bandwidth=B
Resolution status: good

Figure 11.15 Abstraction/Respecialization Adaptation for the CBR Example

Trouble: file_transfer_throughput=F
Additional data: network_load=N
Resolution: A=f(F,N), adjust_network_load=A
Resolution status: good

Figure 11.16 Critic-Based Adaptation for the CBR Example

person decides to add, remove, reorder, or replace an existing solution. Figure 11.16 shows an example where the network_load has been added as an additional parameter in adjusting the network load, and resolution A is a function of two variables, F and N. This is added as a new case to the case library.

CBR-Based CRITTER. The architecture of CRITTER is shown in Figure 11.17 [Lewis, 1996]. It is integrated with the NMS, Spectrum. The core modules of CRITTER are the four basic modules of the CBR system shown in Figure 11.12: input, retrieve, adapt, and process. There is a fifth additional module, propose, which displays potential solutions found by the reasoning module and allows the user to inspect and manually adapt these solutions.

The input module receives its input from the fault detection module of Spectrum. The process module updates the ticket library with the new experience. The retrieve module uses determinators to retrieve a group of tickets from the library that are similar to an outstanding ticket. The initial set of determination rules is based on expertise knowledge and is built into the determinators module. The application technique is the strategy used by the adapt module. User-based adaptation is the interface module for the user to propose critic-based adaptation.

Comparing RBR and CBR [Kolodner, 1997] distinguishes the differences between RBR and CBR. In RBR, the retrieval is done on exact match, whereas in CBR the match is done on a partial basis. RBR

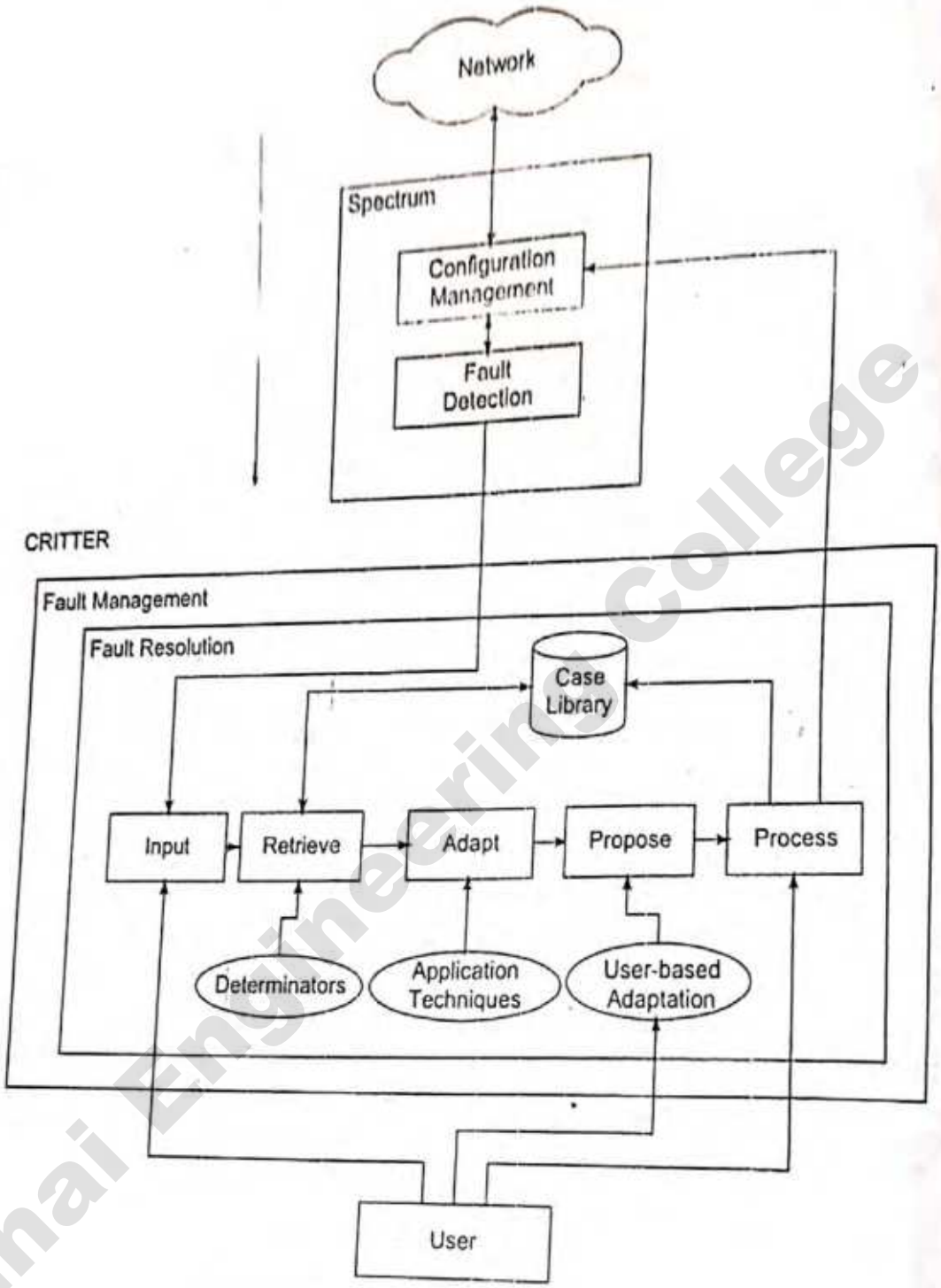


Figure 11.17 CRITTER Architecture

is applied to an iterative cycle of microevents. CBR is applied as a total solution to the trouble and adapted to the situation on hand.

11.4.4 Codebook Correlation Model

Algorithms have been developed to correlate events that are generated in networks based on model of the network and the behavior of network components. Because they are based on algorithms, cla are made that they do not require expert knowledge to associate the events with problems. Although is true, we still need expert knowledge in selecting the right kinds of input that are to be fed to the c relator to develop an efficient system.

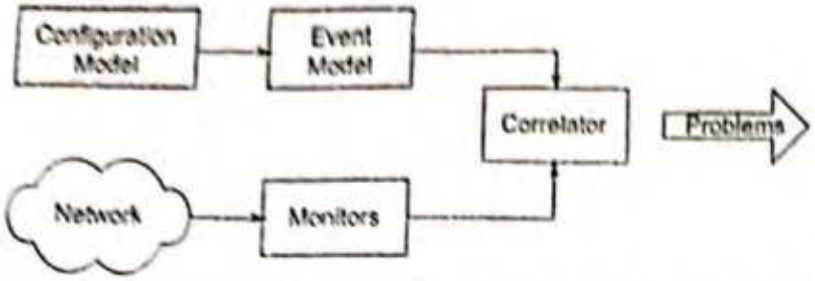


Figure 11.18 Generic Architecture of an Event Correlation System

Figure 11.18 [Kliger *et al.*, 1995] shows the architecture of a model-based event correlation system. We should caution that the "model-based event correlation" should not be confused with the term "model-based reasoning approach" that we discussed in Section 11.4.2. As the heading states, we will refer to this as **codebook correlation.**

Monitors capture alarm events and input them to the correlator. The configuration model contains the configuration of the network. The event model represents the various events and their causal relationships (we will soon define the causality relationship). The correlator correlates the alarm events with the event model and determines the common problems that caused the alarm event.

One of the correlation algorithms based on generic modeling is a coding approach to event correlation [Kliger *et al.*, 1995]. In this approach, problem events are viewed as messages generated by a system and "encoded" in sets of alarms that they cause. The function of the correlator is to "decode" those problem messages to identify the problems. Thus, the coding technique comprises two phases. In the first phase, called the **codebook selection phase**, problems to be monitored are identified and the symptoms or alarms that each of them generates are associated with the problem. (As we stated at the beginning of this approach, this is where expert knowledge is needed.) This produces a problem–symptom matrix. In the second phase, the correlator compares the stream of alarm events with the codebook and identifies the problem.

In order to generate the codebook matrix of problem–symptom, let us first consider a causality graph, which represents symptom events caused by other events. An example of such a causality graph is shown in Figure 11.19. Each node in the graph represents an event. Nodes are connected by directed edges, with edges starting at a causing event and terminating at a resulting event. For example, event E1 causes events E4 and E5. Notice that events E1, E2, and E3 have the directed edges only going out from them and none coming into them. We can identify these nodes as problem nodes and the rest as symptom nodes, as they all have at least one directed edge pointing inward. With problems labeled as Ps and symptoms as Ss, the newly labeled causality graph of Figure 11.19 is shown in Figure 11.20. There are three problem nodes, P1, P2, and P3, and four symptom nodes S1, S2, S3, and S4. We have eliminated

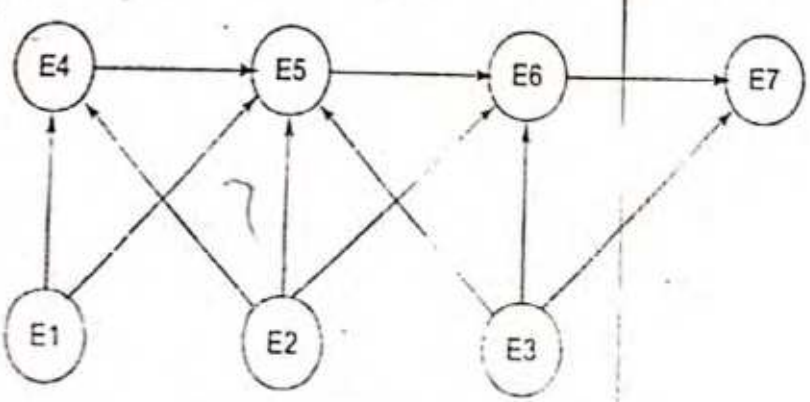


Figure 11.19 Causality Graph

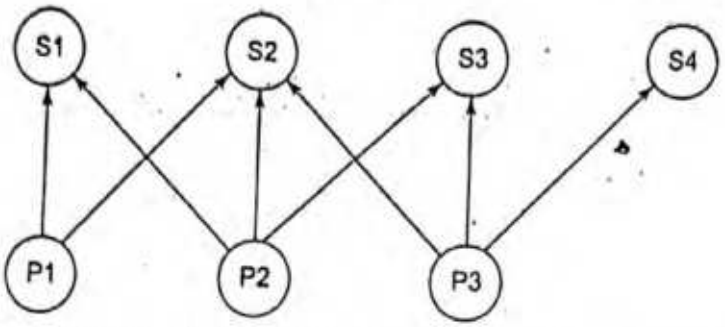


Figure 11.20 Labeled Causality Graph for Figure 11.19

those directed arrows where one symptom causes another symptom, as it does not add any additional information to the overall causality graph.

We can now generate a codebook of problem–symptom matrix for the causality graph of Figure 11.20 (we will drop the qualifier "labeled" from now on). This is shown in Figure 11.21 with three columns as problems and four rows as symptoms.

In general, the number of symptoms will exceed the number of problems and hence, the codebook can be reduced to a minimal set of symptoms needed to uniquely identify the problems. It is easy to show that two rows are adequate to uniquely identify the three problems in the codebook shown in Figure 11.21. We will keep row S1 and try to eliminate subsequent rows, one at a time. At each step, we want to make sure that the remaining codebook distinguishes between the problems. You can prove to yourself that eliminating rows S2 and S3 does not preserve the uniqueness, whereas eliminating either S2 and S4 does. The reduced codebook, called the correlation matrix, is shown in Figure 11.22.

Drawing the causality graph based on the correlation matrix of Figure 11.20, we derive the correlation graph shown in Figure 11.23, which is called the correlation graph.

We will apply the above knowledge to a more general situation of the causality graph shown in Figure 11.24 [Kliger *et al.*, 1995]. Figure 11.24(a) depicts the causality graph of 11 events. Figure 11.24(b) shows the equivalent problem–symptom causality graph. Nodes 1, 2, and 11 show only outgoing directed arrows and are hence identified as problems and the rest of the nodes as symptoms.

We will now reduce the causality graph to a correlation graph. Symptoms 3, 4, and 5 form a cycle of causal equivalence and can be replaced by a single symptom, 3. Symptoms 7 and 10 are caused, respectively, by

	P1	P2	P3
S1	1	1	0
S2	1	1	1
S3	0	1	1
S4	0	0	1

Figure 11.21 Codebook for Figure 11.20

	P1	P2	P3
S1	1	1	0
S3	0	1	1

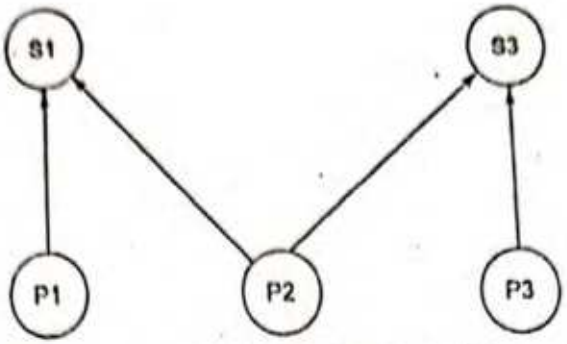
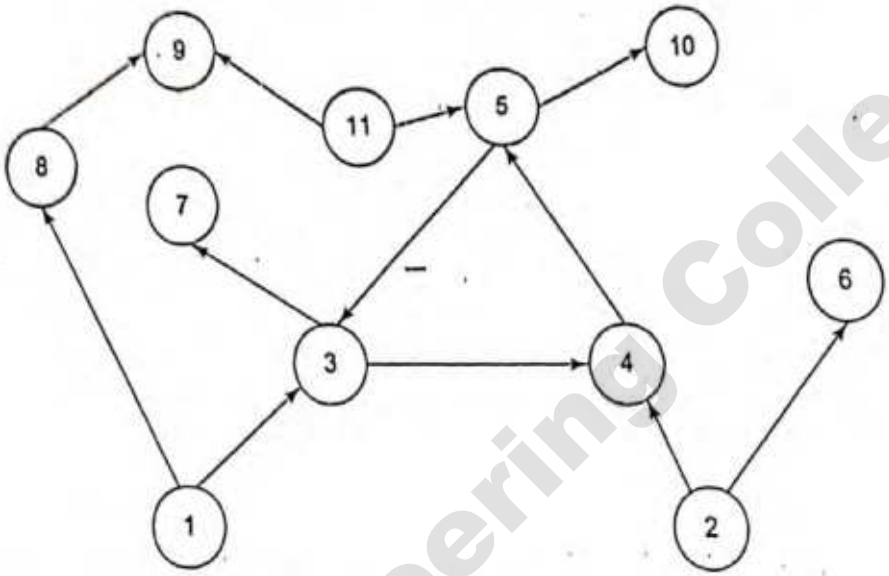
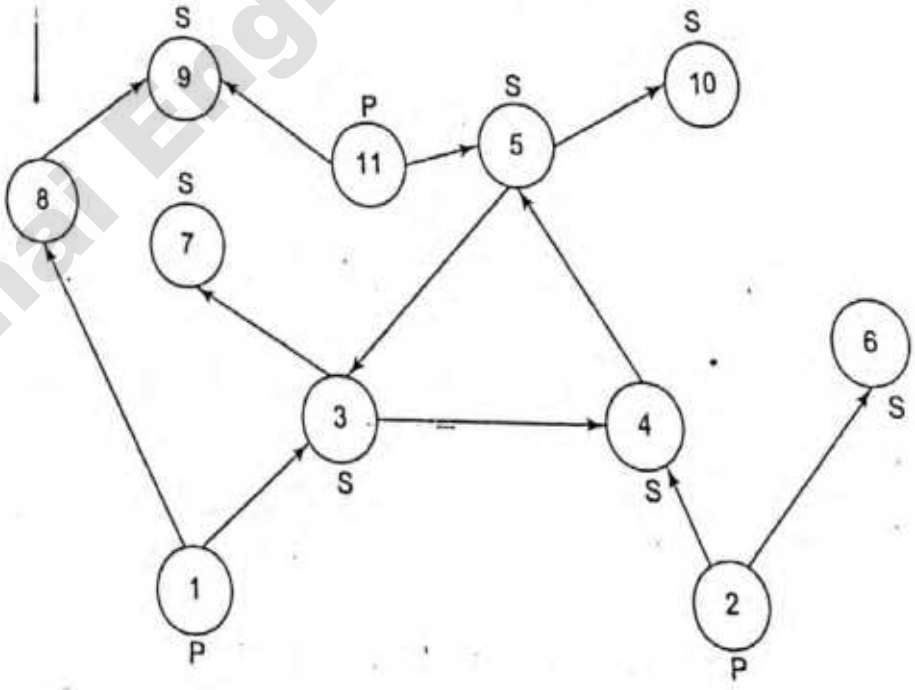


Figure 11.23 Correlation Graph for Figure 11.20



(a) Event Causality Graph



(b) Problem–Symptom Causality Graph

Figure 11.24 Generalized Causality Graph

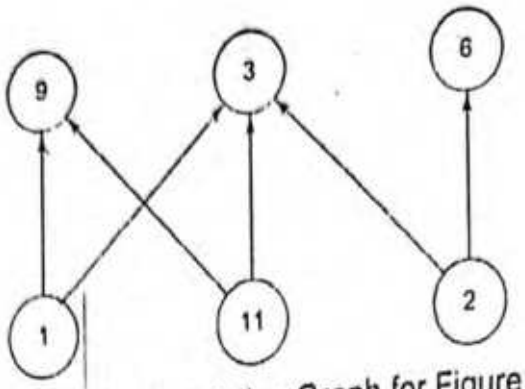


Figure 11.25 Correlation Graph for Figure 11.24

	P1	P2	P11
S3	1	1	1
S6	0	1	0
S9	1	0	1

Figure 11.26 Correlation Matrix for Figure 11.24

symptoms 3 and 5 and hence can be ignored. Likewise, symptom 8 can be eliminated as it is an intermed ate symptom node between problem node 1 and symptom node 9, which is also directly related to proble node 11. We thus arrive at the correlation graph shown in Figure 11.25 and the correlation matrix shown Figure 11.26. Notice that in the particular example the model is unable to distinguish between problem 1 and 11 as they produce identical symptoms in the correlation graph based on the event model.

Further refinements can be made in the codebook approach to event correlation in terms of tolerance to spurious noises and probability relationship in the causality graph. We have derived the correlatio matrix to be the minimal causal matrix. Thus, each column in the code matrix is differentiated from other columns by at least one bit (i.e., value in one cell). From coding theory, this corresponds to a Ham ming distance of one. Any spurious noise in the event detection could change one of the bits and thus a codeword would identify a pair of problems. This could be avoided by increasing the Hamming distance to two or more, which would increase the number of symptoms in the correlation matrix. Also, the rela tionship between a problem and symptoms could be defined in terms of probability of occurrence, and the correlation matrix would be a probabilistic matrix.

The codebook correlation technique has been implemented in InCharge system developed by System Management ARTS (SMARTS) [Yemini *et al.*, 1996].

11.4.5 State Transition Graph Model

A state transition graph model is used by Seagate's NerveCenter correlation system. This could be used as a stand-alone system or integrated with an NMS, which HP OpenView and some other vendors have done.

A simple state diagram with two states for a ping/response process is shown in Figure 11.27. The two states are *ping node* and *receive response*. When an NMS sends a ping, it transitions from the *ping node* state to the *receive response* state. When it receives a response, it transitions back to the *ping node* state. As you know by now, this method is how the health of all the components is monitored by the NMS.

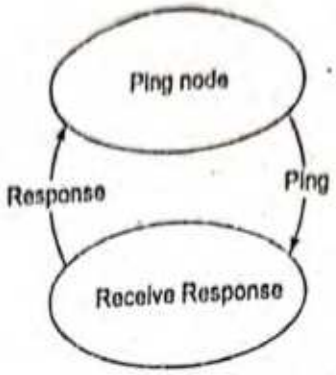


Figure 11.27 State Transition Diagram for Ping/Response

It is best to illustrate with an example of how a state transition diagram could be used to correlate events in the network. Let us choose the same example as in model-based reasoning, Figure 11.11. An NMS is pinging the hubs that are accessed via a router. Let us follow through the scenario of the NMS pinging a hub. When the hub is working and the connectivity to the NMS is good, a response is received for each ping sent, say every minute, by the NMS. This is represented by the top two states, *ping hub* and *receive response*, on the left side of Figure 11.28. Let us now consider the situation when a response for a ping is not received before the next ping is ready to be sent. NMS typically expects a response in 300 milliseconds (we are not pinging some obscure host in a foreign country!). An action is taken by the NMS and the state transitions from *receive response* to *pinged twice* (referred to as *ground state* by NerveCenter). It is possible that a response is received for the second ping and in that situation the state transitions back to the normal *ping hub* state.

However, if there is no response for the second ping, NMS pings a third time. The state transition is now *pinged three times*. The response for this ping will cause a transition to the *ping hub* state. However, let us consider the situation of no-response for the third ping. Let us assume that the NMS is configured to ping three times before it declares that there is a communication failure between it and the hub. Without any correlation, an alarm will be triggered and the icon representing the hub would turn red.

However, the hub may actually be working and the workstations on it may all be communicating with each other. From the topology database, the correlator in the NMS is aware that the path to the hub is via the router. Hence, on failure of the third ping, an action is taken and the system transitions to the *ping router* state. The router is pinged and the system transitions to the *receive response from router* state.

There are two possible outcomes now. The connectivity to the router is lost and no response is received from the router. The system takes no action, which is indicated by the closed loop in the *ping router* state. (How does the router icon turn red in this case?)

The second possibility is that a response is received from the router. This means that the connection to the hub is lost. Now, the correlator in the NMS triggers an alarm that turns the hub icon red.

We notice that in the scenario of a router connectivity failure, only the router icon turns red and none of the hubs connected to it turn red, thus identifying the root cause of the problem.

11.4.6 Finite State Machine Model

Another model-based fault detection scheme uses the communicating finite state machine [Miller, 1991]. The main claim of this process is that it is a passive testing system. It is assumed that an observer agent is present in each node and reports abnormality to a central point. We can visualize the node ob-

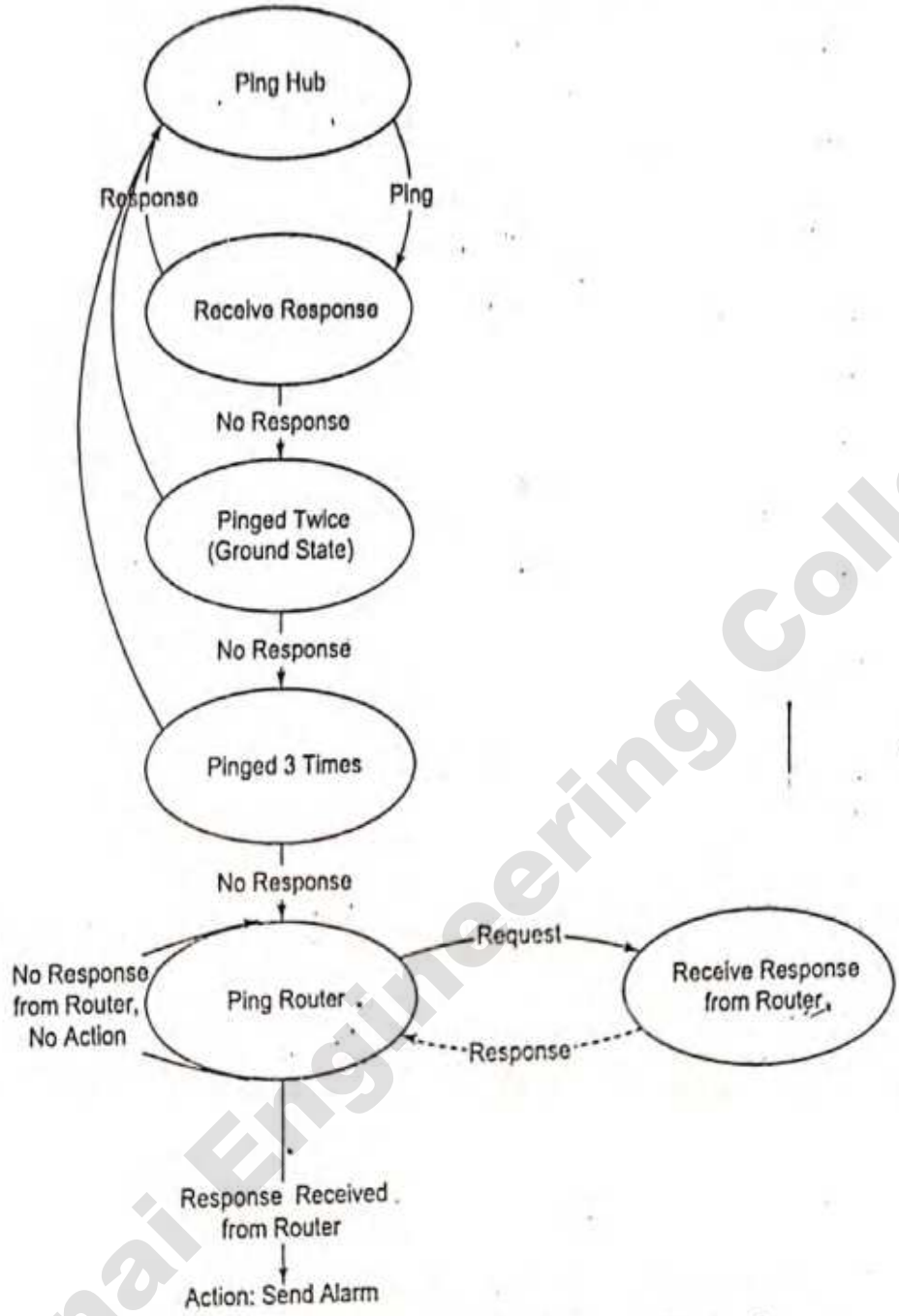


Figure 11.28 State Transition Graph Example

server as a Web agent and the central point as the Web server. An application on the server correlates the events. A failure in a node or a link is indicated by the state machine associated with the component entering an illegal state.

A simple communicating finite state machine for a client–server system is shown in Figure 11.29. It presents communication between a client and server via a communication channel. For simplicity, both the client and the server are assumed to have two states each. The client, which is in send request state, sends a request message to the server, and transitions to receive the response state. The server is currently in the receive request state. The server receives the request and transitions to the send response state. After processing the request, it sends the response and transitions back to the receive request state. The client then receives the response from the server and transitions to the send request state.

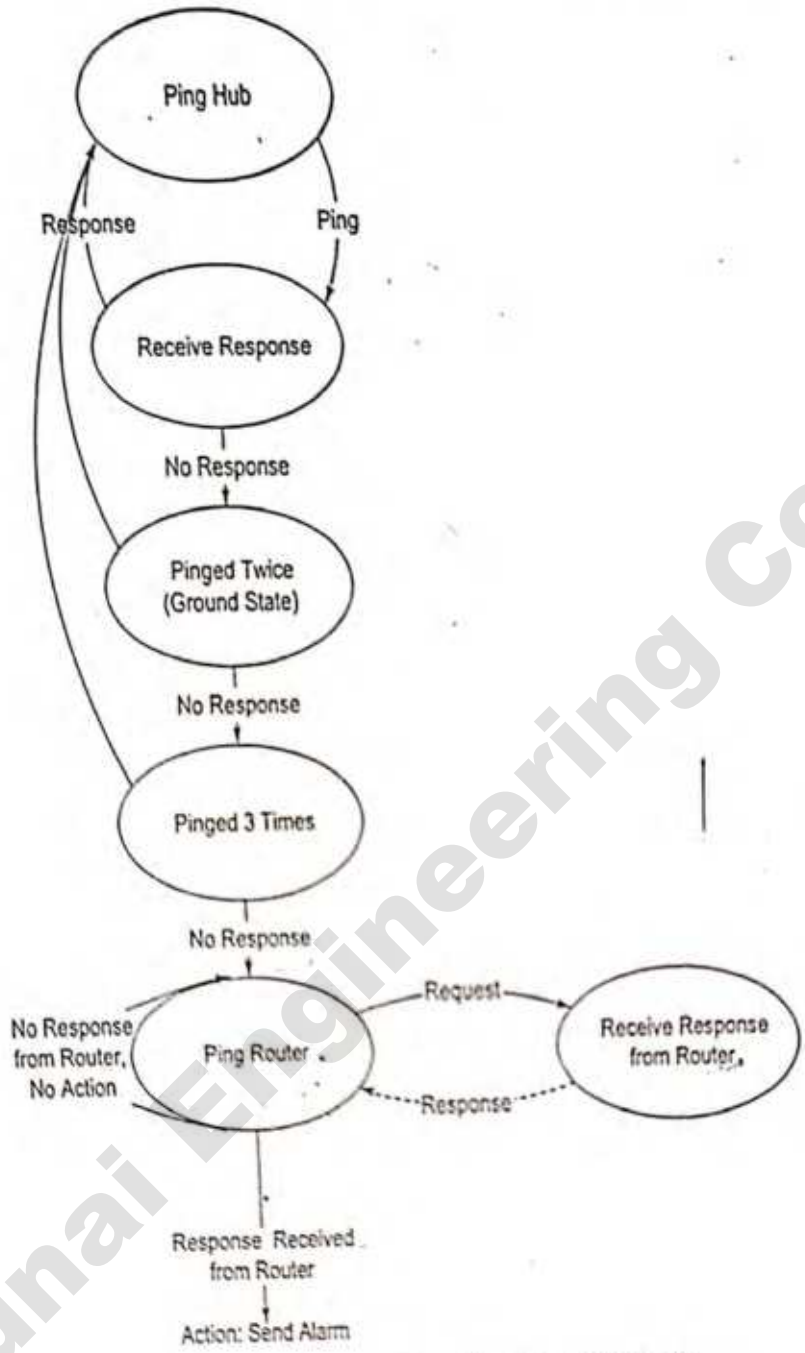


Figure 11.28 State Transition Graph Example

server as a Web agent and the central point as the Web server. An application on the server correlates the events. A failure in a node or a link is indicated by the state machine associated with the component entering an illegal state.

A simple communicating finite state machine for a client–server system is shown in Figure 11.29. It presents communication between a client and server via a communication channel. For simplicity, both the client and the server are assumed to have two states each. The client, which is in send request state, sends a request message to the server, and transitions to receive the response state. The server is currently in the receive request state. The server receives the request and transitions to the send response state. After processing the request, it sends the response and transitions back to the receive request state. The client then receives the response from the server and transitions to the send request state.

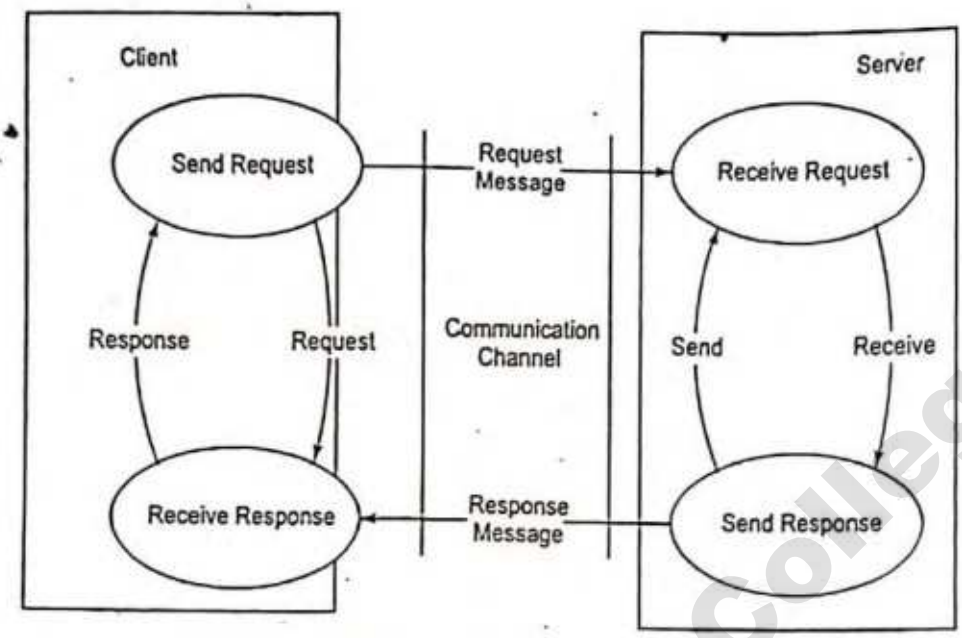


Figure 11.29 Communicating Finite State Machine

If either the client or the server enters an illegal state during the transitions, the system has encountered a fault. For example, after sending a response, if the server does not transition to receive a request state, it is in a failed state. A message is sent to a central location under a fault condition either by the component itself or by the one communicating with the failed component. This is a passive detection scheme similar to the trap mechanism.

We can observe a similarity between the finite state machine model and the state transition graph model with regard to state transitions. However, the main difference is that the former is a passive system and the latter is an active one.

11.5 SECURITY MANAGEMENT

Security management is both a technical and an administrative issue in information management. It involves securing access to the network and information flowing in the network, access to data stored in the network, and manipulating the data that are stored and flowing across the network. The scope of network and access to it not only covers enterprise intranet network, but also the Internet that it is connected to.

Another area of great concern in secure communication is communication with mobile stations. There was an embarrassing case of a voice conversation from the car-phone of a politician being intercepted by a third party traveling in an automobile. Of course, this was an analog signal. However, this could also happen in the case of a mobile digital station such as a hand-held stock trading device. An intruder could intercept messages and alter trade transactions either to benefit by it or to hurt the person sending or receiving them.

In Chapter 7 we covered several of the security issues associated with SNMP management as part of SNMPv3 specifications and discussed possible security threats. Four types of security threats to network management were identified: modification of information, masquerade, message stream modification, and disclosure. They are applicable to security in the implementation of security subsystems in the agent (authoritative engine) and in the manager (non-authoritative engine). The SNMPv3 security subsystem

is the User-Based Security Model (USM). It has two modules—an authentication module and a privacy module. The former addresses data integrity and data origin, the latter is concerned with data confidentiality, message timeliness and limited message protection. The basic concepts discussed in Chapter 7 are part of generalized security management in data communications.

Security management goes beyond the realm of SNMP management. In this section, we will address policies and procedures, measures to prevent security breaches, and network protection from software attacks. Policies and procedures should cover preventive measures, steps to be taken during the occurrence of a security breach, and post-incident measures. Because the Internet is so pervasive and everybody's network is part of it, all government and private organizations in the world are concerned with security and privacy of information traversing it.

In this introductory textbook, we will not be going into the depth of security management that it deserves. For additional information, you are advised to pursue the innumerable references available on the subject [Cooper *et al.*, 1995, Kaufman *et al.*, 1995, Leinwand and Conroy, 1996; RFC 2196; Wack and Carnahan, 1994].

Policies and Procedures

The IETF workgroup that generated RFC 2196 defines a security policy as "a formal statement of the rules by which people who are given access to an organization's technology and information assets must abide." Corporate policy should address both access and security breaches. Access policy is concerned with who has access to what information and from what source. SNMP management addressed this in terms of a community access policy for network management information. An example of access policy in an enterprise network could be that all employees have full access to the network. However, not everyone should have access to all corporate information, and thus accounts are established for appropriate employees to have access to appropriate hosts and applications in those hosts. These policies should be written so that all employees are fully aware of them.

However, illegal entry into systems and accessing of networks must be protected against. The policies and procedures for site security management on the Internet are dealt with in detail elsewhere [RFC 2196, NIST, 1994]. The National Computer Security Center (NCSC) has published what is known as the Orange Book, which defines a rating scheme for computers. It is based on the security design features of the computer. The issues for corporate site security using the intranet are the same as for the Internet and are applicable to them equally. It is a framework for setting security policies and procedures.

The basic guide to setting up policies and procedures is:

1. Identify what you are trying to protect
2. Determine what you are trying to protect it from
3. Determine how likely the threats are
4. Implement measures, which will protect your assets in a cost-effective manner
5. Review the process continuously and make improvements to each item if a weakness is found

The assets that need to be protected should be listed including hardware, software, data, documentation, supplies, and people who have responsibility for all of the above. The classic threats are from unauthorized access to resources and/or information, unintended and/or unauthorized disclosure of information, and denial of service. Denial of service is a serious attack on the network. The network is brought to a state in which it can no longer carry legitimate users' data. This is done either by attacking the routers or by flooding the network with extraneous traffic.

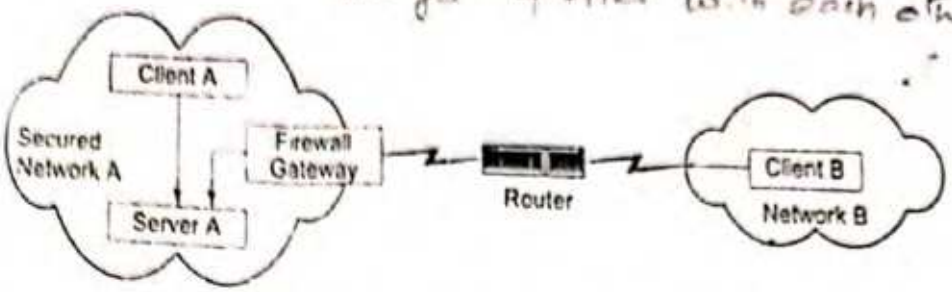


Figure 11.30 Secure Communication Network

11.5.2 Resources to Prevent Security Breaches

We addressed the policies and procedures in the last section. In this section, we will discuss various security breaches that are attempted to access data and systems, and the resources available to protect them.

Figure 11.30 shows a secure communication network, which is actually a misnomer. There is no fully secure system in the real world; there are only systems which are hard and time-consuming to break into, as we shall describe. Figure 11.30 shows two networks communicating with each other via a WAN, which has just one router. Server A and Client A shown in Network A are communicating with each other; and Client B in Network B is also communicating (or trying to communicate) with Server A in Network A.

Let us look at the security breach points in this scenario. Hosts in Network B may not have the privilege to access Network A. The firewall gateway shown in Figure 11.30 is used to screen traffic going in and out of secure Network A. Even if Network B has access permission to Network A, some intruder, for example one who has access to the router in the path, may intercept the message. The contents of the message, as well as source and destination identifications, can be monitored and manipulated, which are security breaches.

Security breaches can occur in the Internet and intranet environment in numerous ways. In most corporate environments, security is limited to user identification and password. Even the password is not changed often enough. This is the extent of authentication. Authorization is limited to the establishment of accounts, i.e., who can log into an application on a host. Besides normal activities of breach, we have to protect against special situations, such as when a disgruntled employee could embed virus programs in company programs and products.

11.5.3 Firewalls

The main purpose of a firewall is to protect a network from external attacks. It monitors and controls traffic into and out of a secure network. It can be implemented in a router, or a gateway, or a special host. A firewall is normally located at the gateway to a network, but it may also be implemented at host access points.

There are numerous benefits in implementing a firewall to a network. It reduces the risk of access to hosts from an external network by filtering insecure services. It can provide controlled access to the network in that only specified hosts or network segments could access some hosts. Since security protection from external threats is centralized and transparent, it reduces the annoyance to internal users while controlling the external users. A firewall could also be used to protect the privacy of a corporation. F

example, services such as the utility *finger*, which provides information about employees to outsiders, can be prevented from accessing the network.

When the security policy of a company is implemented in a firewall, it is a concatenation of a higher-level access service policy, where a total service is filtered out. For example, the dial-in service can be totally denied at the service policy level, and the firewall can filter out selected services, such as the utility *finger*, which is used to obtain information on personnel.

Firewalls use packet filtering or application-level gateways as the two primary techniques of controlling undesired traffic.

Packet Filters. Packet filtering is the ability to filter packets based on protocol-specific criteria. It is done at the OSI data link, network, and transport layers. Packet filters are implemented in some commercial routers, called *screening routers* or *packet-filtering routers*. We will use the generic term of packet-filtering routers here. Although routers do not look at the transport layers, some vendors have implemented this additional feature to sell them as firewall routers. The filtering is done on the following parameters: source IP address, destination IP address, source TCP/UDP port, and destination TCP/IP port. The filtering is implemented in each port of the router and can be programmed independently.

Packet-filtering routers can either drop packets or redirect them to specific hosts for further screening, as shown in Figure 11.31. Some of the packets never reach the local network as they are trashed. For example, all packets from network segment a.b.c.0 are programmed to be rejected, as well as File Transfer Protocol (FTP) packets from *d.e.f.0:21* (note that Port 21 is a standard FTP port). The SMTP (email) and FTP packets are redirected to their respective gateways for further screening. It may be observed from the figure that the firewall is asymmetric. All incoming SMTP and FTP packets are parsed to check whether they should be dropped or forwarded. However, outgoing SMTP and FTP packets have already been screened by the gateways and do not have to be checked by the packet-filtering router.

A packet-filtering firewall works well when the rules to be implemented are simple. However, the more rules we introduce, the more difficult it is to implement. The rules have to be implemented in the right order or they may produce adverse effects. Testing and debugging are also difficult in packet filtering [Chapman, 1992].

Application-Level Gateway. An application-level gateway is used to overcome some of the problems identified in packet filtering. Figure 11.32 shows the application gateway architecture. Firewalls F1 and F2 will only forward if data are to or from the application gateway. Thus, the secured LAN is a gateway LAN. The application gateway behaves differently for each application, and filtering is handled by the proxy services in the application gateway. For example, for FTP service, the file is stored first in the application gateway and then forwarded. For TELNET service, the application gateway verifies the

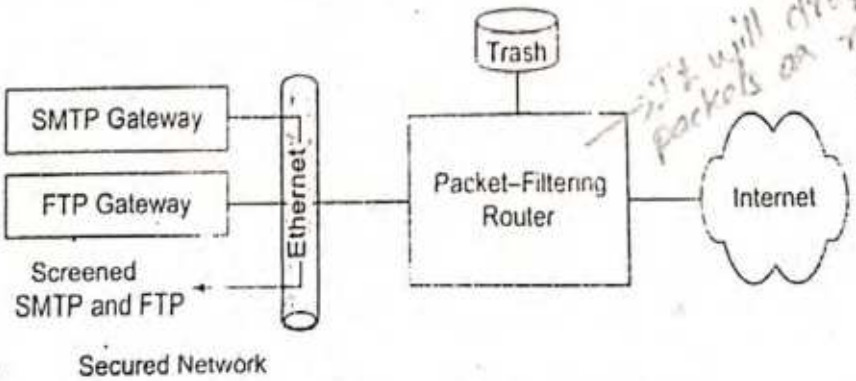


Figure 11.31 Packet-Filtering Router

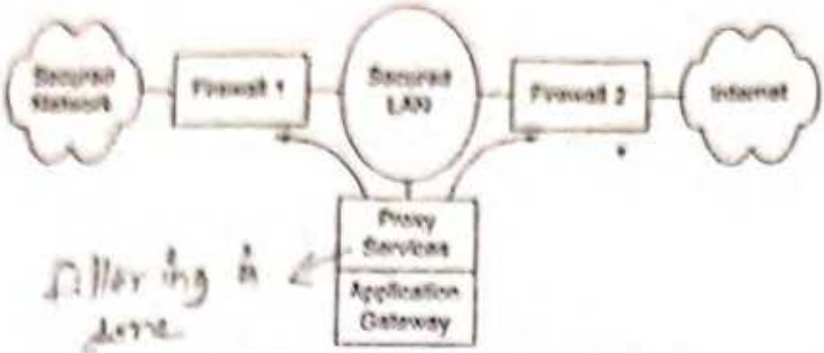


Figure 11.32 Application-Level Gateway

authentication of the foreign host, the legitimacy to communicate with the local host, and then makes the connection between the gateway and the local host. It keeps a log of all transactions.

Firewalls protect a secure site by checking addresses (such as IP address), transport parameters (such as FTP, NNTP), and applications. However, how do we protect access from an external source based on the user, who is using a false identification? Moreover, how do we protect against an intruder manipulating the data while they are traversing the network between the source and the destination? These concerns are addressed by secure communication.

11.5.4 Cryptography

For secure communication, we need to ensure integrity protection and authentication validation. **Integrity protection** makes sure that the information has not been tampered with as it traverses between the source and the destination. **Authentication validation** validates the originator identification. In other words, when Ian receives a message that identifies it coming from Rita, is it really Rita who sent the message? These two important aspects address the four security threats—modification of information, masquerade, message stream modification, and disclosure—mentioned at the beginning of Section 11.5. Besides the actual message, control and protocol handshakes need to be secure.

There are hardware solutions to authentication. However, it is not a complete solution, since the information could be intercepted and tampered with as it traverses from the source to the destination, including the user identification and password.

The technology that is best suited to achieving secure communication is software based. Its foundation lies in cryptography. Hashing or message digest, and digital signature, which we will address soon, are built on top of it to achieve integrity protection and source authentication.

Cryptographic Communication. Cryptography means secret (crypto) writing (graphy). It deals with techniques of transmitting information, for example a letter from a sender to a receiver without any intermediary being able to decipher it. You may view this as the information (letter) being translated to a special language that only the sender and receiver can interpret. Now, cryptography should also detect if somebody was able to intercept the information. Again extending our analogy, if the letter written in a secret language were to be mailed in a sealed (we mean really sealed) envelope, if somebody tampers with it, the receiver would detect it.

The basic model of cryptographic communication is shown in Figure 11.33. The input message, called **plaintext**, is encrypted by the encryption module using a **secret (encryption) key**. The encrypted message is called **ciphertext**, which traverses through an unsecure communication channel, the Internet

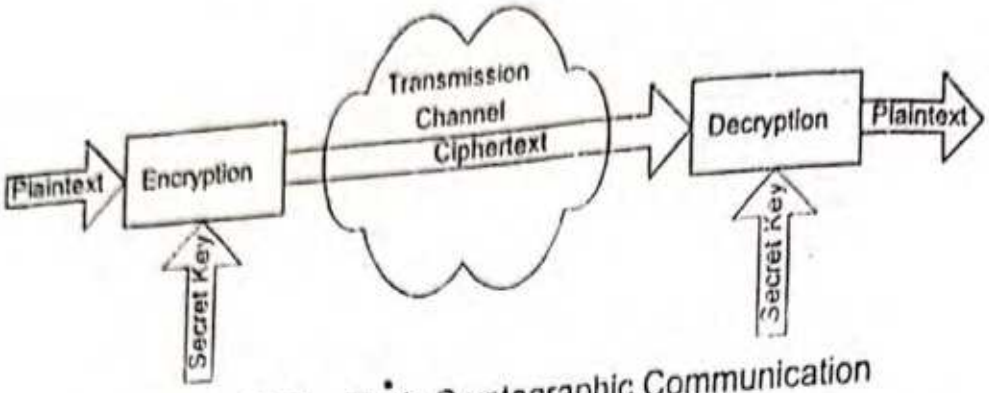


Figure 11.33 Basic Cryptographic Communication

for example. The ciphertext is unintelligible information. At the receiving end, the decryption modu deciphers the message with a decryption key to retrieve the plaintext.

The first known example of cryptography is the Caesar cipher. In this scheme, each letter is replace by another letter, which is three letters later in the alphabet (i.e., key of 3). Thus, the plaintext, *netwo management*, will read as *qhwzrun pdqdjhphqw* in ciphertext. Of course, the receiver knew ahead time the secret key (3) for successfully decrypting the message back to the plaintext *network manag ment* by moving each letter back three positions.

Secret Key Cryptography. The Caesar cipher was later enhanced by the makers of Ovaltine a distributed as Captain Midnight Secret Decoder rings. Each letter is replaced by another letter *n* lette later in the alphabet (i.e., key of *n*). Of course, the sender and the receiver have to agree ahead on t secret key for successful communication. It is the same key that is used for encryption and decryptic and is called **secret key cryptography**. The encryption and decryption modules can be implemented either hardware or software.

- It is not hard to decode the above ciphertext by an intruder. It would only take a maximum of 2 attempts to decipher since there are 26 letters in the alphabet. Another encryption scheme, *monoa phabetic cipher*, is to replace each letter uniquely with another letter that is randomly chosen. Now, th maximum number of attempts for the intruder to decipher has been increased 26! (26! = 26 · 25 · 24 ...1). However, it really does not take that many attempts as there are patterns in a language.

Obviously, the key is the key (no pun intended) to the security of messages. Another aspect of the ke is the convenience of using it. We will illustrate our scenario with Ian and Rita (Ian for initiator and Rit for responder so that it is easy to remember) as users at the two ends of a "secure" communication link Ian and Rita could share a key, their "secret key," for accomplishing secure communication. However, i Ian wants to communicate with Ted (for third party), they both need to share a secret key. Soon, Ian ha to remember one secret key for each person with whom he wants to communicate, which obviously i impractical. It is hard enough to remember your own passwords, if you have several of them, and whic systems they go with.

Two standard algorithms implement secret key cryptography. They are Data Encryption Standar (DES) and International Data Encryption Algorithm (IDEA) [Kaufman *et al.*, 1995]. They both dea with 64-bit message blocks and create the same size ciphertext. DES uses a 56-bit key and IDEA use a 128-bit key. DES is designed for efficient hardware implementation and consequently has a poo performance if implemented in software. In contrast to that, IDEA functions efficiently in softwar implementation.

Both DES and IDEA are based on the same principle of encryption. The bits in the plaintext block are rearranged using a predetermined algorithm and the secret key several times. While decrypting, the process is repeated in the reverse order for DES and is a bit more complicated for IDEA.

A message that is longer than the block length is divided into 64-bit message blocks. There are several algorithms to break the message. One of the more popular ones is the cipher block chaining (CBC) method. We learned the use of it in USM in SNMPv3 in Section 7.7. There, the message was broken using CBC and then encrypted using DES. Performing such an operation on the message, even on identical plaintext blocks, would result in dissimilar ciphertext blocks.

Public Key Cryptography. We observed that each user has to have a secret key for every user that he/she/it (a program) wants to communicate with. Public key cryptography [Diffe and Hellman, 1976; Kaufman *et al.*, 1995] overcomes the difficulty of having too many keys for using cryptography. Secret key cryptography is symmetric in that the same key is used for encryption and decryption, but public key is asymmetric with a *public key* and a *private key* (not secret key, remember secret key is symmetric and private key is not). In Figure 11.34, the public key of Ian is the key that Rita, Ted, and everybody else (that Ian wants to communicate with) would know and use to encrypt messages to Ian. The private key, which only Ian knows, is the key that Ian would use to decrypt the messages. With this scheme, there is secure communication between Ian and his communicators on a one-to-one basis. Rita's message to Ian can be read only by Ian and not by anyone else who has his public key, since the public key cannot be used to decrypt the message.

We can compare the use of asymmetric public and private keys in cryptography to a mailbox (or a bank deposit box) with two openings. There is a mail slot to drop the mail and a collection door to take out mail. Suppose it is a private mailbox in a club and has restricted access to members only. All members can open the mail slot with a public key given by the administration to drop their mail, possibly containing comments on a sensitive issue of the club. Any member's mail cannot be accessed by other members since the public key only lets the members open the mail slot to drop mail and not the collection door. The administrator with a private key can open the collection door and access the mail of all the members. Of course, this asymmetric example has more to do with access than cryptography. But, you get the idea!

The Diffe-Hellman public key algorithm is the oldest public key algorithm. It is a hybrid of secret and public key. The commonly used public key cryptography algorithm is RSA, named after its inventors, Rivest *et al.* [1978]. It does both encryption and decryption as well as digital signatures. Both the message length and the key length are variable. The commonly used key length is 512 bits. The block

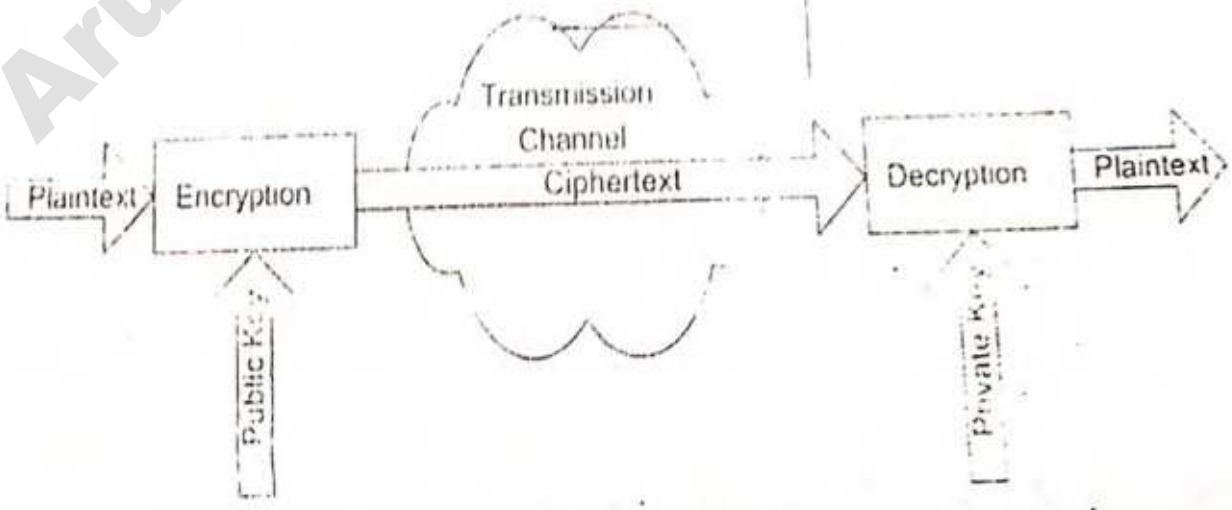


Figure 11.34 Public-Key Cryptographic Communication

size of the plaintext, which is variable, should be less than the key size. The ciphertext is always the length of the key. RSA is less efficient than either of the secret key algorithms, DES or IDEA. Hence, in practice, RSA is used to first encrypt the secret key. The message is then transmitted in one of the secret key algorithms.

Message Digest. Any telecommunications engineer is familiar with the cyclic redundancy check (CRC) detection of errors in digital transmission. This involves calculating a check sum based on the data in the frame or packet at the sending end and transmitting it along with the data. The CRC, also known as *checksum*, is computed at the receiving end and is matched against the received checksum to ensure that the packet is not corrupted. An analogous principle is used in validating the integrity of the message. In order to ensure that the message has not been tampered with between the sender and the receiver, a cryptographic CRC is added with the message. This is derived using a cryptographic hash algorithm, called *message digest* (MD). There are several versions, one of the most common being MD5. We covered the use of MD5 while discussing the authentication protocol in SNMPv3 in Chapter 7. We will look at the use of it in digital signature in the next subsection.

There are different implementations of MD5. In particular, the MD5 utility is used in FreeBSD 2.x (md5sum under LINUX). The utility takes as input a message of arbitrary length producing output consisting of a 128-bit message digest of the input. An example of MD5 utility use is shown in Figure 11.35.

As we can see from the example, the message digest for the string that we entered was generated based on the data received from standard input (from the screen). The FreeBSD version also has a test mode that can be turned on by specifying "-x" as a parameter, as shown in Figure 11.36.

A second algorithm used to obtain a hash or message digest is the Secure Hash Standard (SHS). This has been proposed by the National Institute for Standards and Technology (NIST). It is similar to MD5, but can handle a maximum message length of 2^{64} bits in contrast with MD5, which can handle

```
$ md5
The quick brown fox jumped over the lazy dog
^D
d8e8fca2dc0f896fd7cb4cb0031ba249
```

Figure 11.35 Example of an MD5 Message Digest

```
$ md5 -x
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5
("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345 6789")
= d174ab98d277d9f5a5611c2c9f419d9f
MD5
("12345678901234567890123456789012345678901234567890123456789012345678
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Figure 11.36 MD5 FreeBSD Version Test Mode Output

an unlimited input length of 32-byte chunks. The SHS produces an output of 160-bit, whereas the MD5 output is 128-bit long.

Some significant features of the message digest are worth mentioning. First, there is a one-to-one relationship between the input and the output messages. Thus, the input is uniquely mapped to an output digest. It is interesting to observe that even a one-bit difference in a block of 512 bits could produce a message digest that looks vastly different. In addition, the output messages are completely uncorrelated. Thus, any pattern in the input will not be recognized at the output.

Another feature of message digest is that the output digest is of constant length for a given algorithm with chosen parameters, irrespective of the input message length. In this respect, it is very similar to CRC in that CRC-32 is exactly 32 bits long. We saw in SNMPv3 that the *authKey* generated by the MD5 algorithm is exactly 16-octet long.

Lastly, the generation of a message digest is a one-way function. Given a message, we can generate a unique message digest. However, given a message digest, there is no way the original message could be generated. Thus, if a password were transmitted from a client to a server, this would protect against somebody eavesdropping and deciphering the password. This could also be used for storing the password file in a host without any human being able to decipher it.

We know that the generation of a message digest is a one-way function. We also know that no two messages could produce identical message digests. Could these two combinations ensure that the message is not tampered in transit by an unauthorized person? The answer is no. This is because if the interceptor knows which algorithm is being used, he or she could modify the message (assuming that he/she decrypts the message), generate a new message digest, and send it along with the modified message. If Ian sent the message to Rita, and Ted modified the message as per the above scenario while in transit, Rita would not know the difference. Additional protection is needed to guard against such a threat, which is achieved by attaching a digital signature to the message.

Digital Signature. In public key cryptography, or even in secret key cryptography, if Rita receives a message claiming that it is from Ian, there is no guarantee as to who sent the message. For example, somebody other than Ian who knows Rita's public key could send a message identifying himself or herself as Ian. Rita could not be absolutely sure who sent it. To overcome this problem, a digital signature can be used. Signed public-key cryptographic communication is shown in Figure 11.37.

The digital signature works in the reverse direction from that of public key cryptography. Ian can create a digital signature using his private key (marked "S" in parentheses in Figure 11.37) and Rita could validate it by reading it using Ian's public key. The digital signature depends on the message and the key. Let us consider that Ian is sending a message by email to Rita. A digital signature, which is a message digest, is generated using any hash algorithm with the combined inputs of the plaintext message and the private key of Ian. The digital signature is concatenated with the plaintext message and is encrypted using Rita's public key (marked "R" in parentheses). At the receiving end, the incoming ciphertext message is decrypted by Rita using her private key. She then generates a message digest with the combined input of the plaintext message and Ian's public key, and compares it with the digital signature received. If they match, she concludes that the message has not been tampered with. Further, she is assured that the message is from Ian, as she used Ian's public key to authenticate the source of the message.

Notice that only the originator can create the digital signature with his or her private key and others can look at it with the originator's public key and validate it, but cannot create it. A real-world analogy to digital signature is check writing. The bank can validate the signature as to its originality, but it is hard to duplicate a signature (at least manually) of the person who signed the check.

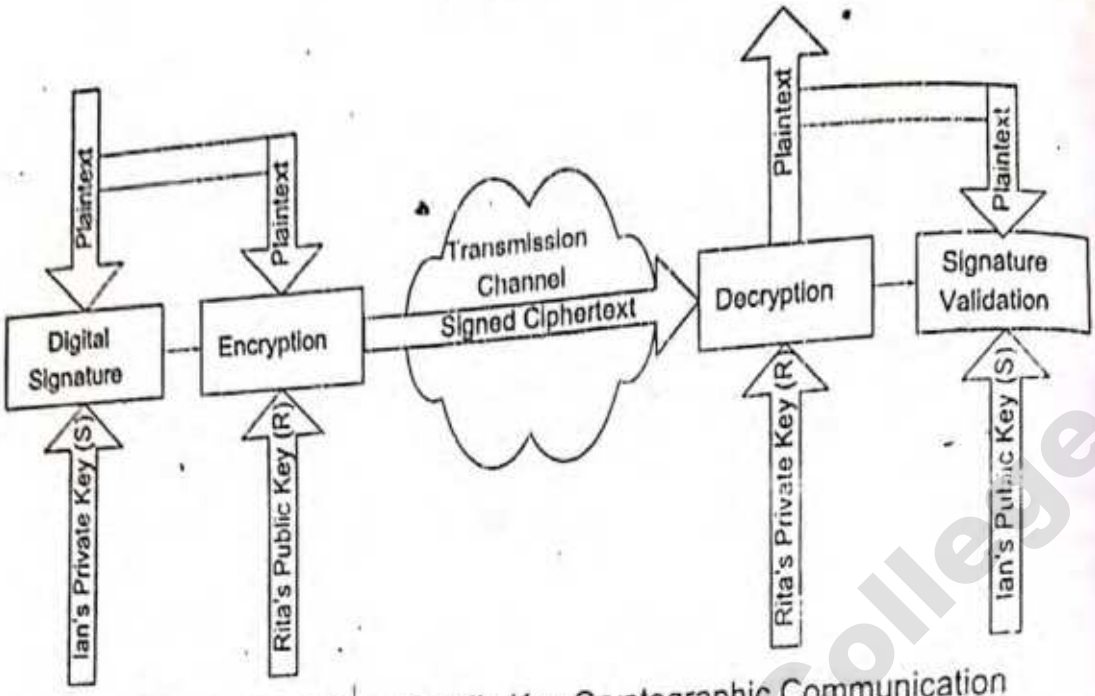


Figure 11.37 Signed Public-Key Cryptographic Communication

Digital signature is valuable in electronic commerce. Suppose Rita wants to place an order w company ABC for buying their router product. She places the order over the Internet with her dig signature attached to it. The digital signature using a public key protects both ABC and Rita regard the validity of the order and who ordered it. It is even better than using secret key cryptography, since the latter case, Rita could change her mind and allege that company ABC generated the order using secret key that they have been using. In public key cryptography, she could not do that.

11.5.5 Authentication and Authorization

Authentication is the verification of the user's identification, and **authorization** is the access privile to the information. On the Internet without security, the user's identification and password, which used for authentication, can easily be captured by an intruder snooping on the LAN or WAN. There several secure mechanisms for authentication, depending on complexity and sensitivity. Authorization use the services could be a simple read, write, read–write, or no-access for a particular service. The pri lege of using the service could be for an indefinite period, or a finite period, or just for one-time use.

There are two main classes of systems, which are of interest to us in the implementation of authentication scheme. The first is the client–server environment in which there is a request–respo communication between the client and the server. The client initiates a request for service to the ser The server responds with the results of the service performed. The communication is essentially tw way communication. In this environment besides authentication (and of course, an integrity chec authorization also needs to be addressed.

The second class of service is a one-way communication environment, such as email or e-comme transaction. The message transmitted by the source is received by the receiver after a considerable d lay—sometimes days if an intermediate server holds up the transaction for a long time. In such a c both the authentication and an integrity check need to be performed at the receiving end.

We will address client–server authentication systems in the next section and the one-way mess authentication and integrity protection system in Section 11.5.7.

11.5.6 Client–Server Authentication Systems

We will consider four types of client–server environments and the implementation of authentication function in each: host/user environment, a ticket-granting system, an authentication system, and authentication using cryptographic function.

Host/User Authentication. We have the traditional host and user validation for authentication, both of which are not very secure. They are also not convenient to use. Host authentication involves certain hosts to be validated by the server providing the service. The host names are administered by the server administrator. The server recognizes the host by the host address. If Server S is authorized to serve a client Host C, then anybody who has an account in C could access Server S. The server maintains the list of users associated with Host C and allows access to the user. If John Smith is one of the users in C, and John wants to access the server from another Workstation W, the workstation W has to be authenticated as a client of S. If not, John is out of luck. Further, his name has to be added to the list of users in W to access S. To make the environment flexible, every client with every possible user is added to the server negating the secure access feature!

Let us consider user authentication, which is done by the user providing identification and a password. The main problem with the password is that it is detected easily by eavesdropping, say using a network probe. To protect against the threat of eavesdropping, the security is enhanced by encrypting the password before transmission. Commercial systems are available that generate a one-time password associated with a password server that validates it when presented by the service-providing host. The user uses a unique key each time to obtain the password, such as in the ticket-granting system (next section).

Ticket-Granting System. We will explain the ticket-granting system with the most popular example of Kerberos, which was a system developed by MIT as part of their Project Athena. Figure 11.38 shows the ticket-granting system with Kerberos. Kerberos consists of an authentication server and a ticket-granting server. The user logs into a client workstation and sends a login request to the authentication server. After verifying that the user is on the access control list, the authentication server gives an encrypted ticket-granting ticket to the client. The client workstation requests a password from the user, which it uses to decrypt the message from the authentication server. The client then interacts with the

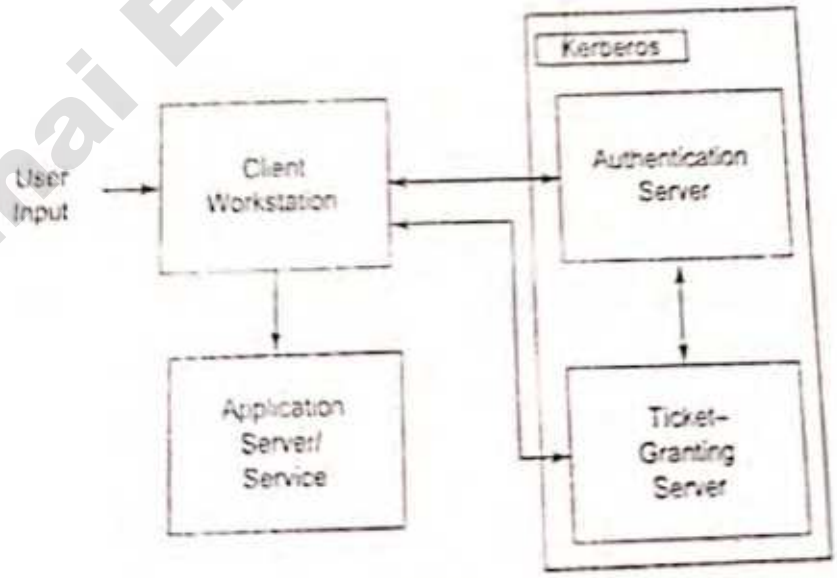


Figure 11.38 Ticket-Granting System

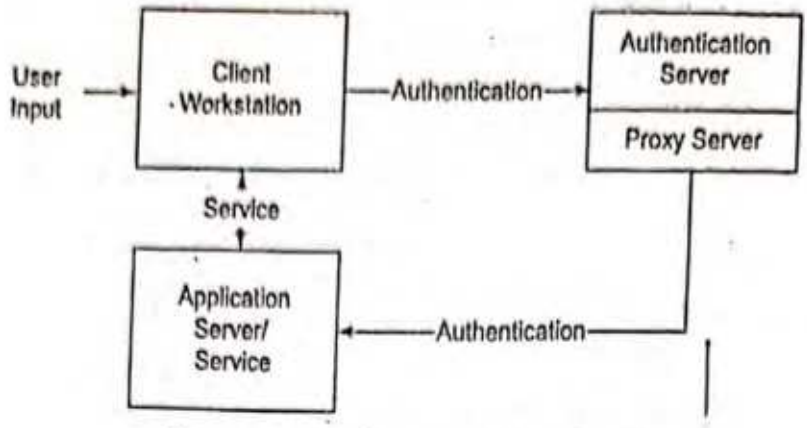


Figure 11.39 Authentication Server

ticket-granting server and obtains a service-granting ticket and a session key to use the application server. The client workstation then requests service from the application server giving the service-granting ticket and the session key. The application server, after validation of the ticket and the session key, provides service to the user. Of course, this processing happens in the background. It is transparent to the user, whose only interaction is with the client workstation requesting application service.

Authentication Server System. An authentication server system, shown in Figure 11.39, is somewhat similar to the ticket-granting system except that there is no ticket granted. No login identification and password pair is sent out of the client workstation. The user authenticates to a central authentication server, which has jurisdiction over a domain of servers. The central authentication server, after validation of the user, acts as a proxy agent to the client and authenticates the user to the application server. This is transparent to the user, and the client proceeds to communicate with the application server. This is the architecture of Novell LAN.

Authentication Using Cryptographic Functions. Cryptographic authentication uses cryptographic functions. The sender can encrypt an authentication request to the receiver, who decrypts the message to validate the identification of the user. Algorithms and keys are used to encrypt and decrypt messages, which we will address now.

11.5.7 Message Transfer Security

The one-way message transfer system is non-interactive. For example, if Rita receives an email from a person who claims to be Ian, she needs to authenticate Ian as the originator of the message, as well as to ensure that nobody has tampered with the message. This could also be the situation in the case where Ian sends a sell order from his mobile station in his car. Ted could intercept the message and alter the number of shares or the price. We will treat all these under the category of secure mail systems.

There are three secure mail systems—privacy-enhanced mail (PEM), pretty good privacy (PGP), and X.400-based mail system [Kaufman *et al.*, 1995]. All three schemes are variations of the signed public-key cryptographic communication discussed in Section 11.5.4 and shown in Figure 11.37. We will describe PEM and PGP in this section. X-400 is a set of specifications for an email system defined by the ITU Standards Committee and adopted by OSI. It is a framework rather than implementation-ready specification. We will also review SNMPv3 secure communication that we covered in Chapter 7 as it bears a close resemblance to the message transfer security.

Privacy-Enhanced Mail (PEM). Privacy-enhanced mail (PEM) was developed by IETF, and specifications are documented in RFC 1421–RFC 1424. It is intended to provide PEM using end-to-end cryptography between originator and recipient processes [RFC 1421]. The PEM provides privacy enhancement services (what else!), which are defined as (1) confidentiality, (2) authentication, (3) message integrity assurance, and (4) non-repudiation of origin. The cryptographic key, called the *data encryption key* (DEK), could be either a secret key or a public key based on the specific implementation and is thus flexible. However, the originating and terminating ends must have common agreement (obviously!).

Figure 11.40 shows three PEM processes defined by IETF: MIC-CLEAR, MIC-ONLY, and ENCRYPTED based on message integrity and encryption scheme. Only the originating end is shown. In all three procedures, reverse procedures are used to extract the message and validate the originator ID and message integrity. The differences between the three procedures are dependent on the extent of cryptography used and message encoding. The **message integrity code** (MIC) is generated as discussed in Section 11.5.4 on digital signature and included as part of email in all three procedures.

The specification provides two types of keys—a **data-encrypting key** (DEK) and an **interexchange key** (IK). The DEK is a random number generated on a per message basis. The DEK is used to encrypt the message text and also to generate an MIC, if needed. The IK, which is a long-range key agreed upon between the sender and the receiver, is used to encrypt DEK for transmission within the message. The IK is either a public or a secret key based on the type of cryptographic exchange used.

If an asymmetric public key is used to encrypt the message, then the sender cannot repudiate ownership of the message. Legal evidence of message transactions is stored in the data, which are used in applications such as e-commerce. Another common characteristic of these procedures is the first step in converting the user-supplied plaintext to a canonical message text representation, defined as equivalent to the inter-SMTP representation of message text. The final output in each procedure is used as the text portion of the email in the electronic mail system.

Figure 11.40(a) shows the MIC-CLEAR procedure and is the simplest of the three. The MIC generated is concatenated with the SMTP text and is inserted as the text portion in the email.

In the MIC-ONLY procedure, shown in Figure 11.40(b), the SMTP text is encoded into a printable character set. The printable character set consists of a limited set of characters that is assured to be present at all sites and thus make the intermediate sites transparent to the message. The MIC is concatenated with the encoded message and is fed to the email system.

Figure 11.40(c) is the most sophisticated of the three procedures. The SMTP text is padded, if needed, and encrypted. A public key is the best choice here, because it guarantees the originator ID. The encrypted message, encrypted MIC, and the DEK are all encoded in printable code to pass through the mail system as ordinary text. They are concatenated and fed to the email system.

Pretty Good Privacy (PGP). Pretty good privacy (PGP) is a secure mail package developed by Phil Zimmerman that is available in the public domain. Figure 11.41 shows the various modules in the PGP process at the originating end. The reverse process occurs at the receiving end and is not shown in the figure. PGP is a package in the sense that it does not reinvent the wheel. It defines a clever procedure that utilizes various available modules to perform the functions needed to transmit a secure message, such as email.

The signature generation module uses MD5 to generate a hash code of the message and encrypts it with the sender's private key using an RSA algorithm. Either IDEA or RSA is employed to generate the encrypted message. IDEA is more efficient than RSA, but secret key maintenance is necessary in contrast to RSA's use of a public key. The encrypted message is compressed using ZIP. The signature is

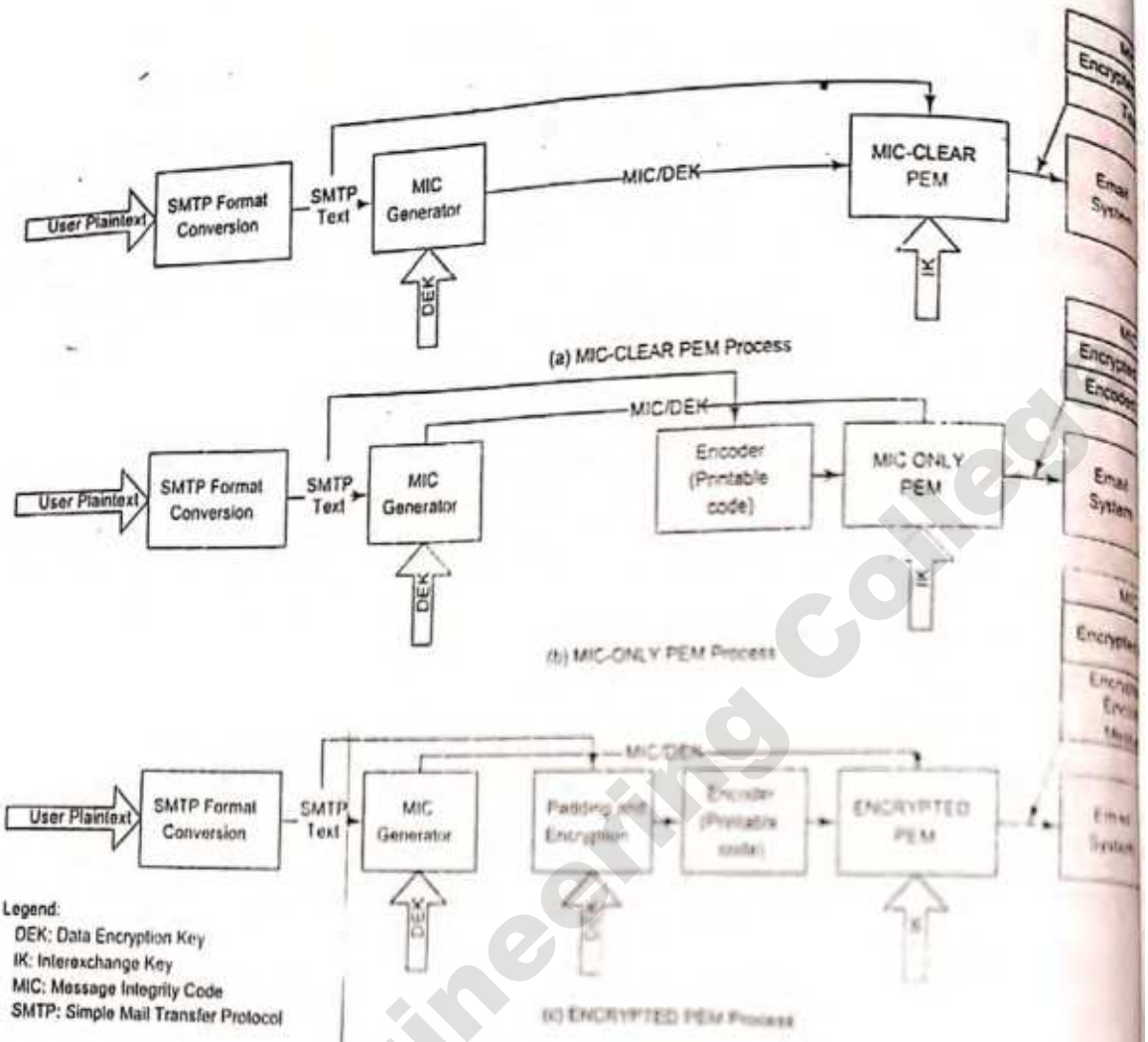


Figure 11.40 PEM Processes

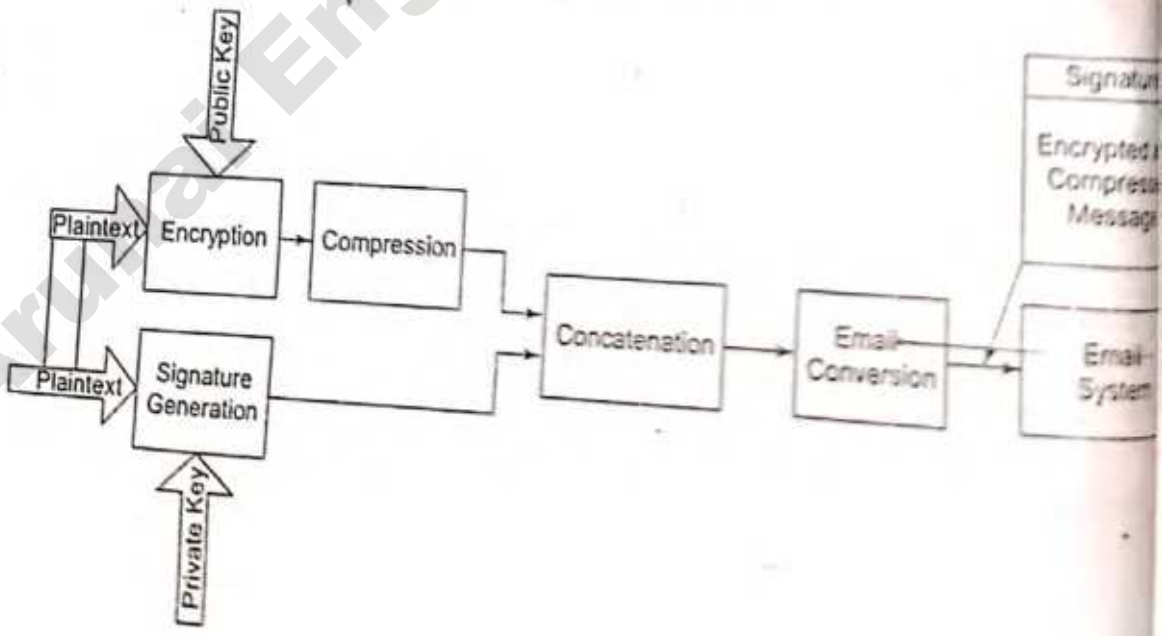


Figure 11.41 PGP Process

concatenated with the encrypted message and converted to ASCII format using the Radix-64 conversion module to make it compatible with the email system.

PGP is similar to ENCRYPTED PEM with additional compression capability. The main difference between PGP and PEM is how the public key is administered. In PGP, it is up to the owner. In PEM, it is formally done by a certification authority (the Internet Policy Certification Authority (PCA) Registration Authority). In practice, PGP is used more than PEM. Both PGP and PEM provide more than a secure mail service. We can send any message or file.

SNMPv3 Security. We dealt with secure transmission in SNMPv3 in Chapter 7. Although an NMS–management agent behaves like a client–server system, the security features are similar to the message transfer cryptography. We will compare the processes studied in Section 7.7 to message transfer cryptography. Figure 11.42 shows a conceptualized representation of Figure 7.13 for an outgoing message.

In an NMS, the user password and authoritative SNMP engine ID (network management agent ID) are used to generate an authentication key by the USM. This is equivalent to the DEK in PEM or the private key in PGP.

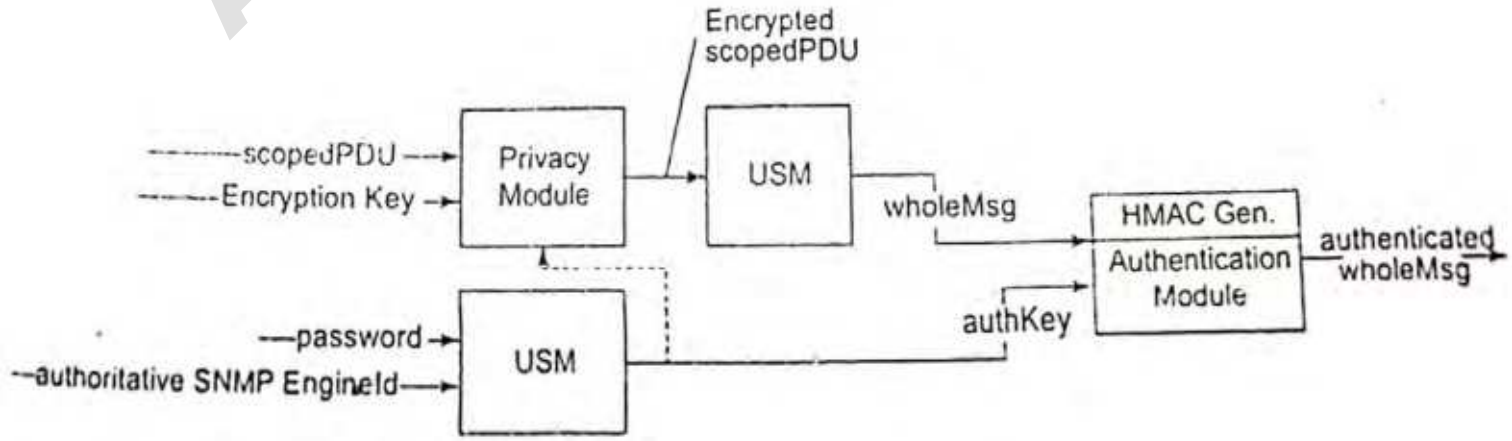
Either the authentication key or preferably a different encryption key is used to generate an encrypted *scopedPDU* by the privacy module. This is similar (but not identical) to encryption of the message in PEM and PGP.

The USM module prepares the whole message with the encrypted *scopedPDU* and other parameters. The authentication key and the whole message are used as inputs to generate HMAC, which is equivalent to the signature in PEM and PGP. The authentication module combines the signature and the whole message to output the authenticated whole message. In an incoming message, the authentication module is provided the whole message, authentication key, and the HMAC as input to validate the authentication.

11.5.8 Network Protection from Virus Attacks

In the current Internet environment, we cannot leave the subject of security without mentioning the undesired and unexpected virus attack on networks and hosts. It is usually a program that, when executed, causes harm by making copies and inserting them into other programs. It contaminates a network by importing an infected program from outside sources, either online or via disks.

The impact of virus infection manifests itself in many ways. Among the serious ones are preventing access to your hard disk by infecting the boot track, compromising your processor (an outside source controlling your computer), flooding your network with extraneous traffic that prevents your hosts from using it, etc.



Generally, viruses are recognized by patterns and virus checkers do just that. Apart from the common sense of preventive measures, it is wise to have the latest virus checkers installed on all your hosts. It should be scheduled to run periodically. It also checks the inputs and outputs of the processor for possible virus infection.

11.6 ACCOUNTING MANAGEMENT

Accounting management is probably the least developed function of network management application. We have discussed the gathering of statistics using RMON probes in Chapter 8 and in Section 11.3.4. Accounting management could also include the use of individual hosts, administrative segments, and external traffic.

Accounting of individual hosts is useful for identifying some hidden costs. For example, the library function in universities and large corporations consumes significant resources and may need to be accounted for functionally. This can be done by using the RMON statistics on hosts.

The cost of operations for an information management services department is based on the service that it provides to the rest of the organization. For planning and budget purposes, this may need to be broken into administrative group costs. The network needs to be configured so that all traffic generated by a department can be gathered from monitoring segments dedicated to that department.

External traffic for an institution is handled by service providers. The tariff is negotiated with the service provider based on the volume of traffic and traffic patterns, such as peak traffic and average traffic. Internal validation of the service provider's billing is a good management practice.

11.7 REPORT MANAGEMENT

We have elected to treat report management as a special category, although it is not assigned a special functionality in the OSI classification. Reports for various application functions—configuration, fault, performance, security, and accounting—could normally be addressed in those sections. The reasons for us to deal with reports as a special category are the following. A well-run network operations center goes unnoticed. Attention is paid normally only when there is a crisis or apparent poor service. It is important to generate, analyze, and distribute various reports to the appropriate groups, even when the network is running smoothly.

We can classify such reports into three categories: (1) planning and management reports, (2) system reports, and (3) user reports.

Planning and management reports keep the upper management appraised as to the status of network and system operations. It is also helpful for planning purposes. Budgeting needs to be done for capital and operational expenses. Table 11.1 lists some of the planning and management reports under different categories. Since the information management services department's main product is service, it is important to keep the management apprised of how the quality of service meets the SLA (more on Section 11.9). Reports on this category include network availability, systems availability, problem reports, service response to problem reports, and customer satisfaction. Trends in traffic should address traffic patterns and volume of traffic in the internal network, as well as external traffic. Information technology is constantly evolving and hence management should be kept apprised of upcoming technology and the plan for migration to new technology. Finally, for budgeting purposes, the cost of operations by function, use, and personnel needs to be presented.

Generally, viruses are recognized by patterns and virus checkers do just that. Apart from the common sense of preventive measures, it is wise to have the latest virus checkers installed on all your hosts. It should be scheduled to run periodically. It also checks the inputs and outputs of the processor for possible virus infection.

11.6 ACCOUNTING MANAGEMENT

Accounting management is probably the least developed function of network management application. We have discussed the gathering of statistics using RMON probes in Chapter 8 and in Section 11.3.4. Accounting management could also include the use of individual hosts, administrative segments, and external traffic.

Accounting of individual hosts is useful for identifying some hidden costs. For example, the library function in universities and large corporations consumes significant resources and may need to be accounted for functionally. This can be done by using the RMON statistics on hosts.

The cost of operations for an information management services department is based on the service that it provides to the rest of the organization. For planning and budget purposes, this may need to be broken into administrative group costs. The network needs to be configured so that all traffic generated by a department can be gathered from monitoring segments dedicated to that department.

External traffic for an institution is handled by service providers. The tariff is negotiated with the service provider based on the volume of traffic and traffic patterns, such as peak traffic and average traffic. Internal validation of the service provider's billing is a good management practice.

11.7 REPORT MANAGEMENT

We have elected to treat report management as a special category, although it is not assigned a special functionality in the OSI classification. Reports for various application functions—configuration, fault, performance, security, and accounting—could normally be addressed in those sections. The reasons for us to deal with reports as a special category are the following. A well-run network operations center goes unnoticed. Attention is paid normally only when there is a crisis or apparent poor service. It is important to generate, analyze, and distribute various reports to the appropriate groups, even when the network is running smoothly.

We can classify such reports into three categories: (1) planning and management reports, (2) system reports, and (3) user reports.

Planning and management reports keep the upper management appraised as to the status of network and system operations. It is also helpful for planning purposes. Budgeting needs to be done for capital and operational expenses. Table 11.1 lists some of the planning and management reports under different categories. Since the information management services department's main product is service, it is important to keep the management apprised of how the quality of service meets the SLA (more on Section 11.9). Reports on this category include network availability, systems availability, problem reports, service response to problem reports, and customer satisfaction. Trends in traffic should address traffic patterns and volume of traffic in the internal network, as well as external traffic. Information technology is constantly evolving and hence management should be kept apprised of upcoming technology and the plan for migration to new technology. Finally, for budgeting purposes, the cost of operations by function, use, and personnel needs to be presented.

Table 11.1 Planning and Management Reports

CATEGORY	REPORTS
Quality of service/service level agreement	Network availability
	Systems availability
	Problem reports
	Service response
	Customer satisfaction
Traffic trends	Traffic patterns
	Analysis of internal traffic volume
	Analysis of external traffic volume
Technology trends	Current status
	Technology migration projection
Cost of operations	Function
	Use
	Personnel

Table 11.2 System Reports

CATEGORY	REPORTS
Traffic	Traffic load—internal
	Traffic load—external
Failures	Network failures
	System failures
Performance	Network
	Servers
	Applications

The day-to-day functioning of engineering and operations requires operation-oriented reports. Traffic, failure, and performance are the important categories, as shown in Table 11.2. A pattern analysis of these reports will be helpful in tuning the network for optimum results.

Users are partners in network services and should be kept informed as to how well any SLA is being met. Some service objectives are met by joint efforts of the users and the information management services department. Table 11.3 shows some typical user reports. The SLA normally includes network availability, system availability, traffic load, and performance. In addition, users may require special reports. For example, the administration may want reports on payroll or personnel.

11.8 POLICY-BASED MANAGEMENT

We discussed network and system management tools in the last chapter. In this chapter, we covered the application tools and technology geared toward network and system management. For these to be

Table 11.3 User Reports

CATEGORY	REPORTS
Service level agreement	Network availability
	System availability
	Traffic load
	Performance
User-specific reports	User-defined reports

successfully deployed in an operational environment, we need to define a policy and preferably bu into the system, i.e., implement policy management. For example, network operations center pe may observe an alarm on the NMS, at which time they need to know what action they should tak depends on what component failed, severity or criticality of the failure, when the failure happene In addition, they need to know who should be informed and how, and that depends on when the occurred and what SLAs have been contracted with the user. We illustrated this with an example o in Section 11.4.3, where a policy restraint was used to increase the bandwidth as opposed to re load in resolving a trouble ticket.

As we mentioned in Section 11.5.1 on security management, policy plays an equally important greater, role as the technical area. Without policy establishment and enforcement, security manage is not of much use.

Our focus here is not the administrative side of the subject, although it is important, but wi technical aspects of policy implementation in network management. Figure 11.43 is policy ma ment architecture proposed by [Lewis, 1996] for network management. It consists of a domain spa objects, a rule space consisting of rules, a policy driver that controls action to be performed, and a space that implements the actions and attributes of the network being controlled.

The objects in the domain space are events such as alarms in fault management, packet loss in formance, and authentication failure in security management. The objects have attributes. For exa attributes of alarms are severity, type of device, location of device, etc. Attributes of packet loss c

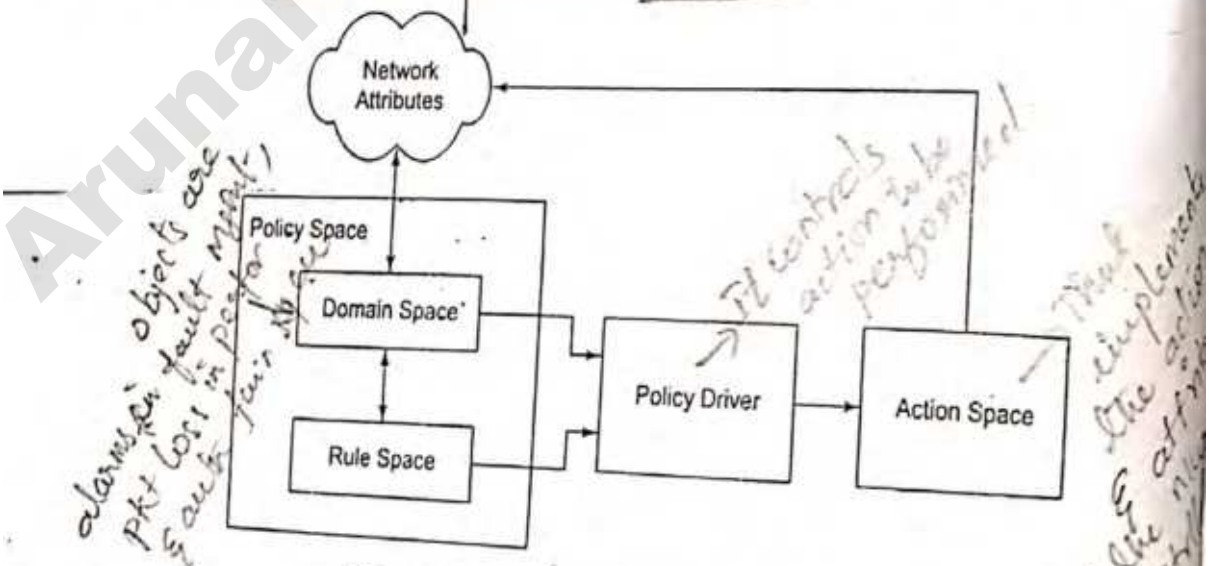


Figure 11.43 Policy Management Architecture

the layer at which packets are lost, the percentage loss, etc. Rules in the rule space define the possible actions that could be taken under various object conditions. It is the same as in RBR, with if–then, condition–action. The policy driver is the control mechanism, which is similar to the inference engine. Thus, the objects in the domain space and the set of rules in the rule space are combined for a policy decision that is made by the policy driver. It is worth bringing the distinction between a rule and policy here. In the operations center, a rule could be that all network failures should be reported to the engineering group. A statement that this applies to the operations personnel at the network management desk makes the rule a policy. Because the responsibility is assigned to specific individuals, failing to do so will be blamed on the person on duty at that time. The action space executes the right-hand side of the rules by changing the attributes of the network and/or executing an external action. In resolving the throughput problem using the CBR technique in Section 11.4.3, we discussed the options regarding the actions to be taken. Whether network load should be controlled or more bandwidth be allocated is a policy decision. This can be implemented as an RBR in the policy rule space. An example of external action could be to page engineering for a severe network failure.

11.9 SERVICE LEVEL MANAGEMENT

Let us keep building a superstructure of telecommunications management to bring us up to date on the technology. We addressed policy management in the last section that ensures the optimal and enterprise-wide consistent use of the network and system management systems. However, the establishment of corporate policy does not stop at the best and consistent use of management tools. The network, systems, and business applications that run on them are there to serve customers, and customer satisfaction is essential for the success of the business [Adams and Willetts, 1996]. Hence, policy management should be driven by service level management, which is the second to the top layer in the TMN model shown in Figure 11.11.

We have illustrated implementing service level management in Chapter 10 on TMN with operations systems. An operations system, in general, does an exclusive or special-purpose function. With the availability of element management and NMSs, it is time for the arrival of a generalized service level management. Service level management is defined as the process of (1) identifying services and characteristics associated with them, (2) negotiating an SLA, (3) deploying agents to monitor and control the performance of network, systems, and application components, and (4) producing service level reports [Lewis, 1999]. Lewis compares the definition of service level management to quality of service (QoS) management defined by the Object Modeling Group (OMG).

The characteristics associated with services are service parameters, service levels, component parameters, and component-to-service mappings. A service parameter is an index into the performance of a service—for example, the availability of a business application for a customer. The business application depends upon various underlying components—for example, network devices, systems, and applications on the systems. Thus, there is a one-to-many mapping between the service parameter and the underlying component parameters. The availability of the business application in the SLA can be defined in terms of the availability of these underlying components. In this case, the availability service parameter is a function of the availability component parameter.

An SLA is a contract between the service provider and the customer, specifying the services to be provided and the quality of those services that the service provider promises to meet. The pricing for the service depends on the QoS commitment.

The objective of service level management is to ensure customer satisfaction by meeting or exceeding the commitments made in the SLA and to guide policy management. In addition, it provides input to the business management system.

Summary

We have learned in this chapter how to apply all the knowledge we have gained in the book to practical situations. We have dealt with the five categories of OSI application functions, namely configuration, fault, performance, security, and accounting.

Configuration management involves, in addition to setting and resetting the parameters of network components, provisioning of the network and inventory management. Operation systems perform the latter functions. Network topology management is concerned with discovery and mapping of the network for operations that can be used to monitor them from a centralized operations center.

Fault detection consists of fault detection and fault isolation. Similarly, performance degradation involves detection and isolation. We dealt with these subjects in a simplistic manner in the early part of the chapter and in a more complex manner in the latter part. We discussed the emerging topic of correlation technology and the various correlation techniques that have been implemented in systems. They correlate events or alarms, which arrive from multiple sources, and determine the root cause of the problem. A knowledge base built upon heuristic experience, as well as algorithmic procedures, is used in such systems, either for the selection of inputs or for reasoning.

While we addressed the issues of performance management, we discussed the performance metrics and the important role of performance statistics in network management.

Security management played a small part in SNMP management, but plays an extremely sensitive and critical role in overall network management. We have dealt with this in detail in this chapter. We covered the importance of policies and procedures. We looked at the various means of how information can be accessed, tampered with, or destroyed. These are done by unauthorized and perverted personnel. We also learned how to protect, if not completely at least partially, against such attacks. In this context, we discussed various authentication and authorization procedures. There are sophisticated cryptographic methods to transport information across unsecured channels to ensure secure communication. We talked about secret and public keys in cryptography to accomplish this. We briefly addressed the issue of how to protect our networks and systems against the growing menace of virus attacks.

From a business management viewpoint, we discussed the methods of using the statistical data gathered from the network to generate accounting applications. Reports play an essential role in the management of information services. We described the three classes of reports: planning and management, system, and user. We gave examples of the types of reports that are useful in each class.

Many of the network and service management decisions are policy based. We discussed how this could be built into the system that would help personnel who are expected to implement those policies.

We brought this chapter to a conclusion discussing service level management. Service level management helps satisfy customer needs. A service level agreement between the customer and the service provider defines the needs of the customer and the commitments of the service provider.

Exercises

1. You are asked to do a study of the use pattern of 24,000 workstations in an academic institution. Make the following assumptions for your study:
   You are pinging each station periodically. The message size in both directions is 128 bytes long. The NMS you are using to do the study is on a 10-Mbps LAN, which functions with 30% efficiency. What would be the frequency of your ping if you were not to exceed 5% overhead?

Network Management Application Protocols: Common Management Information Service Element and File Transfer Access and Management

Irrespective of the actual protocols used to provide an end-to-end integrity of the data, the application specific information exchanged for management falls into two classes. The interactive class of application is exchanged using CMISE and file transfer class using FTAM. This chapter presents a detailed discussion of the features offered by CMISE with examples. A brief review of the features of FTAM is given for completion. As there is no specific use of FTAM in TMN environment today, only a cursory look is provided.

4.1. INTRODUCTION

The previous chapter presented the communication requirements for the Q3 and X TMN interfaces. These requirements provide the infrastructure for transferring management information between the communicating systems acting in the managing and managed roles. Chapter 2 discussed the various network management application functions. These applications, as discussed in the earlier chapters, are categorized into interactive and file-oriented applications.

Applications that exchange information according to a request/reply paradigm belong to the interactive class. The reply may be an acknowledgment or a response with the requested information. In some cases a reply may not be present. Information exchange is often bursty in this class of application. In the literature sometimes this class is referred to as transaction-oriented applications. The term "transaction" usually implies properties such as atomicity, isolation, and concurrency of the requested operation across multiple systems. Mechanisms such as roll back are part of the definition of a transaction. While management interactions may be distributed and require support for the above properties, this is not included in the scope of this chapter.

The file-oriented class corresponds to applications that require transfer of files between the systems. In some cases, combined capabilities of both the classes may be required.

For example, to perform software download exchanges using the request/reply paradigm will be required to identify and define when to activate the new software or a patch to an existing one. The transfer itself may use the file-oriented protocol, a more efficient mechanism.

This chapter discusses the two application service elements specified by TMN standards: Common Management Information Service Element (CMISE) is used as the application layer component for the interactive class and File Transfer Access and Management (FTAM) for the file-oriented class. These two application layer services are distinguished from another class of services such as directory. These provide supporting services for TMN and do not exchange network management application information.

4.2. COMMON MANAGEMENT INFORMATION SERVICE ELEMENT

The communication reference model discussed in Chapter 1 identified that at the application layer, various building blocks called Application Service Elements (ASE) are defined and used in combination to meet the needs of an application. CMISE is an application service element that defines a common structure for exchanging management information. As will be seen later, the services and protocol structure are general, making it suitable for managing various resources—telecommunications and data communications network resources and applications such as directory.

In compliance with the general framework of any OSI application service element, CMISE is composed of two parts. The service definitions[1] are used by the service user in requesting and responding to the requests. The service definitions shield the user of the services from the changes to the protocol.[2] The service definitions are defined in ITU Recommendation X.710|ISO 9595.[3] The protocol specification is presented in X.711| ISO 9596-1. These services are referred to as common because the definitions are applicable to different functional areas. As an example, a service to report an event can be used in fault management to report an alarm and in performance management to report the collected traffic data.

4.2.1. Model

Systems Management architecture was discussed in Chapter 1 using Figure 1.1. The model for describing CMISE is a further refinement of this architecture. Even though CMISE specifies an external interface between two systems, it is impossible to describe the model as well as the services without discussing the model of the managed resources. CMISE and information modeling addressed in the next chapter are very closely intertwined, and it is a difficult choice to determine which topic is to be presented first. The approach chosen here is to present them in the chronological order of standardization. In order

[1]From a programming perspective, service definitions have been used by groups (NMF) to specify application programming interfaces (APIs).

[2]Even though this is the rationale behind the split in terms of service and protocol definitions, the tight coupling between the service and protocol definitions in CMISE makes it difficult to imagine no change to service specification if the protocol has to be modified.

[3]The service and protocol documents in use today were first published in 1991. Corrections and clarifications to remove ambiguities were identified by implementations in the last few years. These changes have now been integrated, and a revised version was approved in 1998.

to facilitate understanding of this chapter, information on modeling the managed resources is provided.

The model of CMISE is split into two aspects: command interface to the managed resources and asynchronous reports from managed resources. The former is referred to as *operations* and the latter is known as *notifications*. From the perspective of request/reply paradigm both interfaces are considered remote operations. However, the semantics added with CMISE is to separate the request/reply (response) in terms of operations sent to the managed resources from a management system and operations sent by the managed system. In order to avoid confusion with generic definition of an operation, the phrase *management operation* is used to refer to operations initiated by the management system and *management notifications* is used for operations initiated by the managed system.

Figures 4.1 and 4.2 show the model of CMISE with the managed resources located as branches of a tree. Information models define management views of a resource. A managed object represents the manageable properties of the resource. Managed objects with the same properties are instances of a managed object class. Examples of MO classes are network element, log, and alarm record. The resource being managed may have additional information in the context of the service it provides in addition to the management information. In the example of the managed object class representing a network element, call processing aspects may not be modeled as they are not used for management.

The CMISE model is defined in terms of management operations performed by the resource and management notifications emitted by the resource. These managed objects and their properties form a repository referred to as *Management Information Base*. The tree structure shown in the figures is a direct result of how the managed objects are referenced or named. A hierarchical naming structure using containment relationship is defined. The

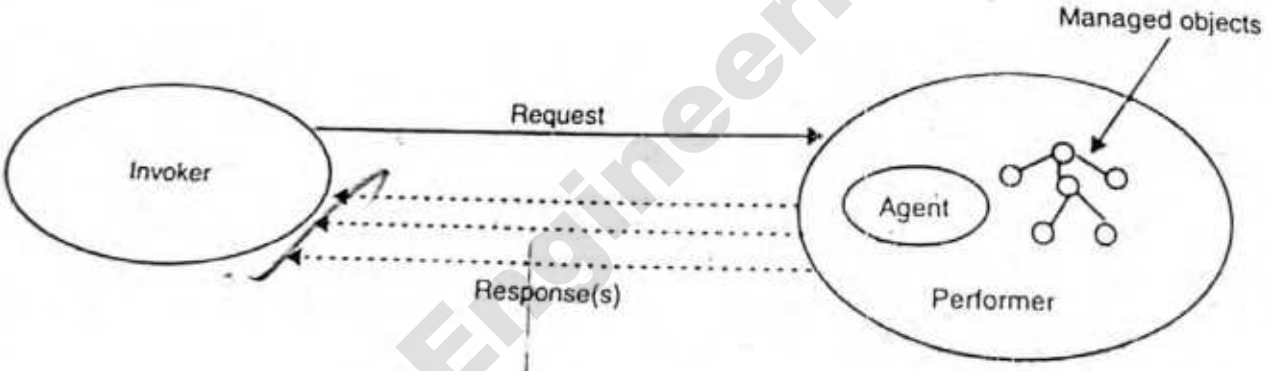


Figure 4.1 Management operations model.

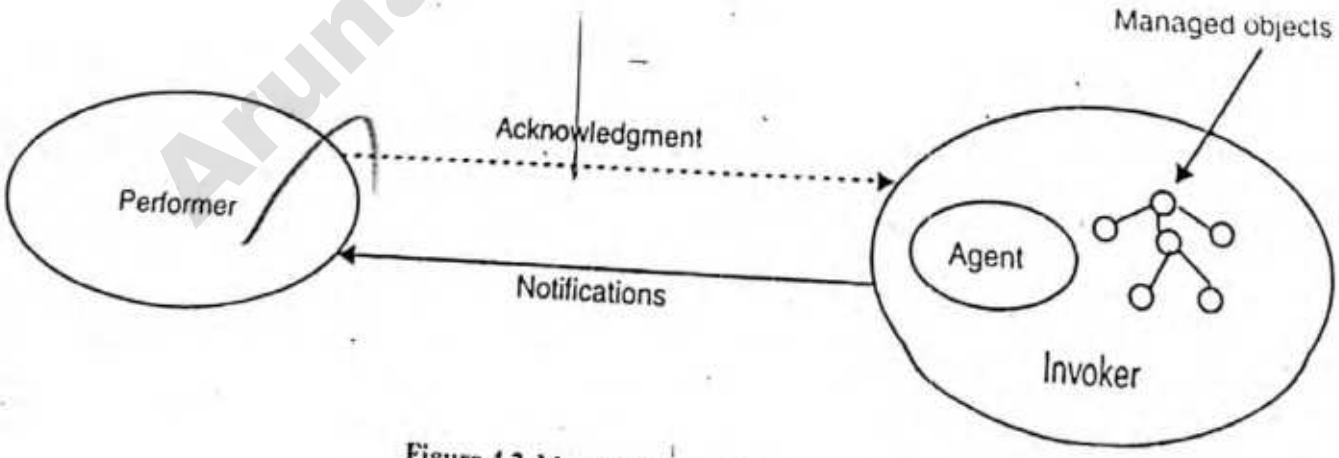


Figure 4.2 Management notifications model.

name of a managed object lower in the tree is specified in terms of the managed object immediately above it. This structure lends itself to effectively use capabilities offered by CMISE services described later.

Before describing the model, let us consider the two terms shown in the figure—*invoker* and *performer*. The context for these terms is any remote operation irrespective of the application. A system in the invoker role issues a request (invokes a remote operation) to a remote system. The system in the performer role responds to this request. The invoker and performer should not be confused with manager and agent. The invoker and performer roles depend on the management context. In Figure 4.1, the invoker role is assumed by a managing system and the performer role by the agent system. In Figure 4.2, where the requested operation is a management notification, the invoker and performer roles are reversed. The agent shown in both figures represents the functionality in the system for providing the external interface.

The model for *management operation* is specified in terms of a request issued by a managing system in the invoker role. The agent system in the performer role receives the request, and the result of the operation (successful or error) is returned in the response. In order to indicate that a response may not always be required, the return arrow is shown as a dashed line. The model also shows in the performer side two concepts—*agent* and a set of *managed objects* structured as a tree from naming perspective. This is because the request may be directed on either a single or multiple objects. Each object that performs the request returns a response. Concepts such as *encapsulation* from object-oriented paradigm are applied in developing this model. The managed object is responsible for maintaining the integrity after the requested operation is completed. The agent, on the other hand, provides a coordinating role. As an example, if a request is sent to include multiple objects, the agent performs actions such as name resolution to identify the specific objects for performing the operation and (if required) synchronizes the request across multiple objects. In addition, agent is responsible for providing the external communication. The last function is applicable for both aspects—responses to management operations and management notifications.

Management notification shown in Figure 4.2 corresponds to external communication of the notifications emitted by the managed object. The notifications are a remote operation invoked by the agent system. As the notification may or may not be acknowledged (response to the request containing the notification[4]) it is again shown using a dashed line.

4.2.2. Service Definitions

Table 4.1 lists the services defined by CMISE. The service column contains the name of the service primitive. Describing the services in terms of service primitives and parameters associated with each primitive is a common practice at all layers of the OSI Reference Model. The services corresponding to various remote operations in management are specified using "M" (stands for management) as the first character. The service primitives in OSI standards fall into two classes—*confirmed* and *unconfirmed*. The confirmed services imply that a responses is required as opposed to the opposite case of unconfirmed. It should be noted that the type of the service primitive in a specific layer does not imply the same type is required in all layers. The confirmation or otherwise depends on the context in which

[4]It is confusing for a casual reader that notifications are considered an operation request since conventionally an operation request is equated to a command. The general request/reply methodology is customized for management.

TABLE 4.1 CMISE Services

Service	Type	Description
M-EVENT-REPORT	confirmed/ unconfirmed	Report an occurrence of an event to another open system
M-GET	confirmed	Retrieve attribute(s) and their values from managed objects
M-SET	confirmed/ unconfirmed	Modify attribute(s) values of managed object(s)
M-ACTION	confirmed/ unconfirmed	Request an open system to perform an action on managed object(s)
M-CREATE	confirmed	Request an open system to create a new managed object Only one instance can be created per request
M-DELETE	confirmed	Request an open system to delete managed object(s)
M-CANCEL-GET	confirmed	Request to cancel a previously invoked M-GET service

the service is defined. Consider the case where from management perspective, reading the properties of a managed object is a confirmed service. A response is expected with the values for the requested properties. In the context of the presentation layer, there is no semantics associated with the data. The presentation layer is only concerned with transforming the data to a representation that the communicating parties understand and is oblivious to whether data in one direction requires a response or not. Within the context of the presentation layer, it is data being sent in both directions, having no understanding of the request/response semantics. In Table 4.1, the type column indicates whether the service is confirmed or unconfirmed or both.

Services that may be used in either confirmed or unconfirmed mode are M-EVENT-REPORT, M-SET, and M-ACTION. If requested in the unconfirmed mode no response is generated by the performer. Before describing the services in the next section, let us briefly expand the concept of managed object class and instance mentioned earlier. This is necessary to understand the table and the parameters of the service.

CMISE services, as mentioned earlier, are used with managed objects and their properties. Even though the next chapter describes how to develop information models for use with CMISE, it is necessary to at least identify the various components to understand the services. The properties are defined in terms of zero or more attributes, events generated, and actions performed. The attributes may or may not be modifiable. The behaviour of a managed object, though very important, is not explicitly used in the service definition. The effects manifest in different ways (for example, a change in the value of a state attribute as a result of resource failure).

The description column indicates that in some cases the reference to managed object includes the possibility of multiple objects. Wherever the letter "s" is shown in the bracket, these services may be requested such that the operation is performed on an individual object or multiple objects. The mechanism for selecting multiple objects will be described in the section on scoping.

The following sections define the parameters associated with each service. The first column identifies the parameter name. The second column indicates if the parameter is present in the request. This automatically implies that the receiving system is informed of the parameter with the same status. The response column specifies the status of the parameter as a result of receiving and/or performing the request. Errors may be returned either because the request contained invalid information or a problem was encountered when performing

the requested operation. The status of this column also applies for the sending system when it receives the response as a confirmation. The status of each parameter is indicated as M, U, C, or -. M indicates that the parameter is always present in every request for this service. U indicates it is a user's option to include this parameter. If the parameter marked U is not present, this is not considered an error. Two classes of user options are used with these parameters. In some cases the parameter is truly optional for an application. In other cases the optionality is determined by the application using the service (for example, an alarm report defined using the M-EVENT-REPORT service). This will be further explained later. The conventions available for an OSI service definition does not provide a notation to distinguish the two cases. C is used to indicate that the presence of the parameter is determined by a condition. The condition is explained as part of the service definition. Some of the parameters in the response column have an equal sign after the status notation. This is used to indicate that the value in the response must equal the value in the request. The status of "-" indicates that the corresponding parameter is not applicable. As with any OSI service definition, the parameters shown in the various tables for each CMIS service identify the information to be provided by the user of the service. This does not imply that each parameter translates to a field in the protocol data unit exchanged between the managing and managed system.

4.2.2.1. Event Report Service. The Event Report Service is used to report an occurrence of an event (sometimes referred to as a notification from the object centric view) emitted by a managed object to another open system.[5] Actually, the event is emitted by the resource and is reflected to an external system, which in this case is a management system, as an event report. It should be noted that a notification from a resource does not always imply an even report will be emitted. It will be seen later that it is possible to configure the type of events to be reported to specific destinations. Table 4.2 describes the parameters associated with this service. Often an event report is equated to alarm erroneously. The service parameters described below may be used with any event irrespective of the function (e.g., alarm surveillance, performance monitoring, status reporting).

Let us now consider each of the above parameters to understand the semantics. *Invoke identifier* is used in all CMISE services to identify a specific request. It is very similar to sequence number in lower layer protocols. Because the same value must be present in the response, correlation can be done between multiple requests and responses. The use of this parameter enables the managing and managed systems to issue requests without waiting for a response. Even though there are outstanding requests, as long as the invoke identifier is not reused (prior to receiving the response), multiple requests can be issued.

The mode parameter in the request column specifies if the event report is to be sent requesting confirmation. If the event report is confirmed, this implies an acknowledgment is requested from the receiving system. This parameter is not applicable in the response.

The parameters *managed object class* and *instance* together identify the resource emitting the notification. The class defines the type of resource, and the instance identifies the specific entity emitting the notification. For example, suppose an equipment model is defined with an object class circuit pack to represent the various cards in a system. An instance of a circuit pack is uniquely identified so that it can be distinguished from other instances of

[5]The term *open system* indicates that the interface offered by the receiving or sending system uses open standard protocols. Specifically this term has been used in conjunction with the OSI Reference Model.

TABLE 4.2 M-Event-Report Service Parameters

Parameter Name	Req/Ind	Rsp/Conf
Invoke identifier	M	M(=)
Mode	M	-
Managed object class	M	U
Managed object instance	M	U
Event type	M	C(=)
Event time	U	-
Event information	U	-
Current time	-	U
Event reply	-	C
Errors	-	C

M—Mandatory
U—User option
C—Condition

cards. Different types of cards (for example, a controller card, line cards, video card) in a system may be considered to be of type circuit pack. However, an instance of a line card will be distinguished from another line card or a controller card using a name that is unique.

The *event type* parameter identifies the type of event. Examples of event types are communication alarm, state change, and object creation. The event type must correlate with the managed object class. An invalid event type error is generated (assuming confirmed mode is used) if the event type is not defined for that class. Event time indicates the time of occurrence of the event. This parameter is optional. The event information, though identified as U, falls into a different class of optionality than the event time. The presence of this parameter is determined by the type of the event. For example, when an alarm is emitted, let us assume that the severity of the alarm and the potential cause for the alarm are required. The event information associated with the definition of the alarm will make this parameter mandatory. The user in this case is the alarm reporting application instead of the end user. In the response, event type is conditional and if present the value must be the same as in the request. The condition for the presence of event type is determined by whether the parameter event reply is included in the response. If the response is merely an acknowledgment for the receipt of the report, then event reply and consequently event type are not required.

The current time in the response is sent optionally to time stamp the response.

The event report service does not include the semantics of the actions or steps to be taken by the system receiving the report. For example, the receiving system may choose to generate an audible alarm or log the event. An acknowledgment, if sent by the receiving system, implies that the receiver has taken the appropriate action. As will be seen from the discussion of the parameters, an event report may be issued requesting acknowledgment. If duplicate acknowledgments are received for the same report, then the managing system cannot distinguish them except in the following cases: time stamp, if present in the acknowledgments (note current time is optional); a mechanism exists in the managing system to correlate the event report sent as request with acknowledgments.

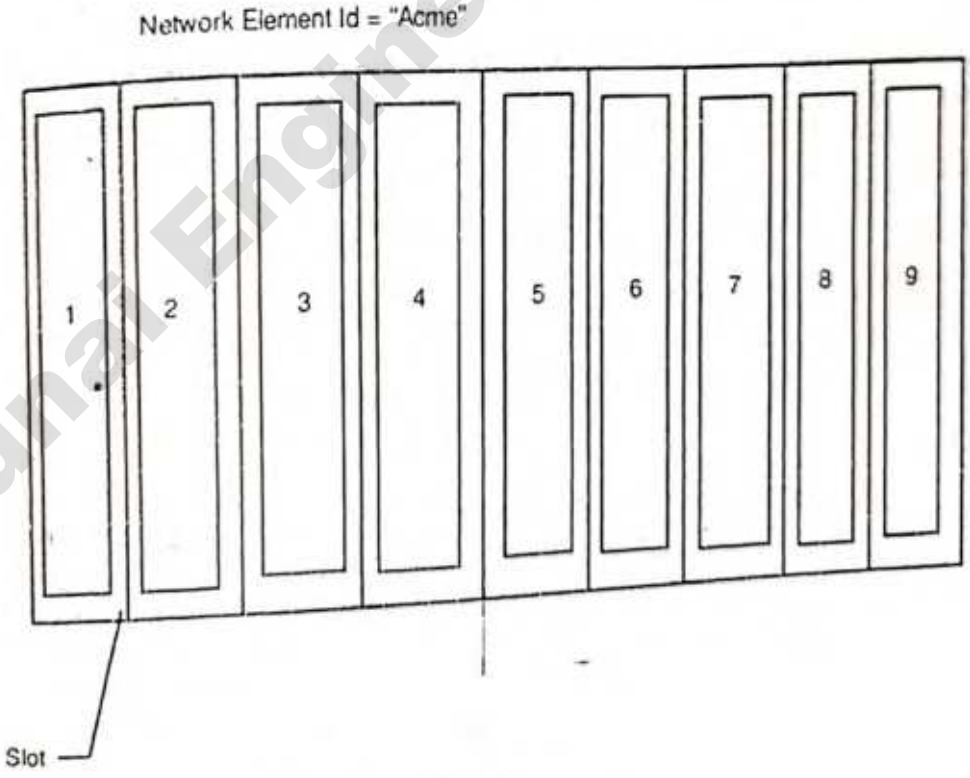
The response may be either an acknowledgment or an error. Various generic and service specific errors are defined in the standard. These are discussed in the section on errors.

An example use of this service is shown using the example in Figure 4.3. Consider a host digital terminal with a E1 card connected to a switch, a controller card, a card to perform the time slot interchange, and a card connected to the distribution side. The network element (HDT) consists of slots where the cards are plugged in. Assume that a managed object called *equipment holder* is used to model the slot and *circuit pack* models the cards.

Assume that a failure of the time slot interchange card (TSIC) occurs and there is no backup available. This will result in an equipment alarm being emitted with a probable cause value of "replaceable unit problem." As there is no backup available, the loss of this card will impact call processing and hence the severity of the alarm is critical. The parameter values for event report service are shown in Table 4.3. It is assumed that event report is unconfirmed and therefore no acknowledgment is required.

The circuit pack and slot (modeled as equipment holder) are named by assigning value to the attribute called *equipment Id*. The managed object instance corresponding to TSIC card is named relative to slot 4 containing it. The combined information of the slot value and the name of the card uniquely identifies this circuit pack from another card in the HDT.

4.2.2.2. Get Service. This confirmed service enables the invoking open system to retrieve the attribute value(s) of one or more managed objects (MOs) from the managing system. This service by definition is a confirmed service where the request is not complete until a response is received (this is to be contrasted with the set and action services defined later). The parameters associated with this service are shown in Table 4.4. As will be seen later in



1. E1 card
2. Protecting E1 card
3. Controller card
4. Time slot interchange card
5. Protecting TSI card
6. Power supply card
7. Protecting power supply card
8. Transport card
9. Protecting transport card

Figure 4.3 Model of a host digital terminal.

TABLE 4.3 Example Use of M-Event-Report Service

Parameter Name	Value
Invoke identifier	7
Mode	unconfirmed
Managed object class	circuit pack
Managed object instance	equipment Id = 4 equipment Id = "TSICA"
Event type	equipment alarm
Event time	1997-7-27:18:32
Event information	
Probable cause	replaceable unit problem
Perceived severity	critical

Table 4.4 Parameters of M-GET Service

Parameter Name	Req/Ind	Rsp/Cnf
Invoke identifier	M	M
Linked identifier	-	C
Mode	M	-
Base object class	M	-
Base object instance	M	-
Scope	U	-
Filter	U	-
Access control	U	-
Synchronization	U	-
Attribute identifier list	U	-
Managed object class	-	C
Managed object instance	-	C
Attribute list	-	C
Current time	-	U
Errors	-	C

the section on scoping, if the request is directed to multiple objects, then each selected object returns a separate reply.

In the event report service the request column identifies the parameters supplied by the managed system and the response column corresponds to the managing system. For all other services described here, the roles of the request and response columns are reversed. The request parameters are supplied by the managing system, and the response parameters are provided by the managed system.

The semantics of invoke identifier is the same for all services. The two parameters *base object class* and *instance* together identify the managed object. The names of the parameters in the request to reference the managed object are different from the event report service. The reason for this relates to the ability to request that the operation be performed on multiple objects. The term "base" is used to signify that the managed object in the request is the start of a search tree to select multiple objects. The simplest form of

using this service is to request the get operation on one object, namely that identified as the base object.

The parameters *scope* and *filter* are discussed later in the section on scoping and filtering, respectively. In summary, these parameters allow the management system to specify criteria for selecting multiple objects from one request. The synchronization parameter is associated with the selection of multiple objects. The operation on multiple objects may be performed as an atomic[6] operation. In this case an operation is not executed if it cannot be performed on any selected object (actually the resource). The synchronization is applied across multiple objects and not across attributes of a single object. If the latter is required, this should be specified as part of the information model.

The access control parameter is present in services initiated by the manager. It should also be noted that the response does not support this parameter. The access control field is a placeholder in that details are not defined within CMIS. Depending on the mechanism chosen, appropriate security information is included. The parameter is an end-user option and mostly determined by the TMN interface type and application. Many of the applications exchanging information on an X interface are using this parameter. The managed system uses this information to determine if the requester has the permission to invoke this operation. A specific error has been defined to indicate denial of access.

The attribute identifier list includes the names of the attributes whose values are to be retrieved. The list may be absent as indicated by the status of "U." When the list of attributes is not present or empty, then values for all attributes of that object(s) are returned. If an attribute is not present in an object, a suitable error is returned.

The response to the get request may be one of the following—successful return of the values for the requested attributes, rejection of the request, and an error response with partial success and error information. If a request includes retrieving multiple attributes some of which are not present in the object, then the response includes values and errors.

The managed object class and instance parameters are applicable only in the response column. The condition status for these parameters is related to whether the request should be performed on multiple objects. If scope for the request is only the base object, then there is no requirement for the response to include identification of the object. If the operation is performed on multiple objects as a result of one request, then the response must include the managed object class and instance parameters.

The correlation between the request and response depends on whether multiple objects are selected to perform the operation. In the case of a single object, the correlation is achieved using the invoke identifier parameter. The value of the parameter in the response equals the value in the request. The tables where requests on multiple objects are issued, the response column for invoke identifier is M and not M (=). It must be noted that if the request does not include the scope parameter, then the value of invoke identifier in the response must equal that in the request. The section on scoping describes how the request and responses are correlated when multiple objects are selected.

The error parameter includes both generic errors applicable to all operations and errors specific to the get operation. The section on errors discusses the details.

[6] *Atomicity* is a property well-defined in a distributed transaction processing environment and corresponds to how the different activities across multiple systems are coordinated. In this case even if the objects are distributed, viewed from the managing system all objects are managed through the same agent. Distribution if implemented is not visible.

Taking the example of the Time Slot Interchange card, Tables 4.5a and b illustrate the use of M-GET service. The attribute list includes affected object list, alarm status, administrative state, and operational state. Assume that an alarm was emitted because of TSIC failure, as shown in the example of event report service.

Because the request in the example addressed a single object, the response is not required to include the name of the managed object (circuit pack referenced in the request). Correlating the request with response is achieved by examining the value of the invoke identifier. The attribute list includes the identifier and value(s) for the requested attributes. The affected object list includes the list of objects that will be affected as a result of failure of the circuit pack. The two objects shown in this example correspond to the line terminations used to provide the clock source for the HDT. The values for the other attributes are self-explanatory. Because of the card failure the operability of the card is lost as shown by the value "disabled." As explained in Chapter 2, the administrative state is controllable by the managing system. Prior to the failure the card was providing service and thus the value "unlocked." In some implementations, it is possible to lock the administrative state as a result of failure.

4.2.2.3. Set Service. This service enables the invoking open system to request modifications of the attribute value(s) of one or more MOs in another open system. This service may be used as confirmed or unconfirmed service. If request is sent unconfirmed, the result of the operation can be determined to be successful or otherwise only by retrieving the

TABLE 4.5a Example M-GET Service Request

Parameter Name	Value
Invoke identifier	4
Base object class	circuit pack
Base object instance	equipment Id = 4 equipment Id = "TSICA"
Attribute identifier list	affected object list, alarm status, replaceable, administrative state, operational state

TABLE 4.5b Example M-GET Service Response

Parameter Name	Value
Invoke identifier	4
Attribute list	affected object list = {ds1 LineTTPId = 1, ds1 LineTTPId = 2}, alarm status = critical, replaceable = yes, administrative state = unlocked operational state = disabled

values of the attributes. The parameters associated with this service are shown in Table 4.6. If the request for modification is directed to multiple objects, then each selected object returns a separate reply.

Many of the parameters in Table 4.6 have the same semantics discussed with M-GET service. The mandatory parameter *modification list* specifies the requested modification to the values of the attribute. The modification list consists of a set of three components. The service is defined to allow modification of multiple attribute values. The components are the identifier for the attribute, the new value(s), and the type of modification. The new value(s) is not always required and is determined by the type of modification. The type of modification is one of the following: replace a value with the new value supplied, add one or more values to an existing set, remove one or more values from an existing set, and set the value to the default specified for the object. The modify operators to add or remove values are applicable only if the attribute is defined to include multiple values at the same time. The affected object list attribute in Table 4.5a for the get service is an example where add and remove values are appropriate modify operations. The value for the attribute is not required when the modification is to set the value to the defined default. Depending on the type of attribute the default may be a single value or a set of values. If no modification operator is supplied, the default is to replace with the supplied value.

Continuing the example of the circuit pack, let us assume that the manager requests that the value of the administrative state be set to locked. Table 4.7 illustrates the parameter names and values for this request. The parameter modification list has one element in the set. The three components of the set are {administrative state, locked, replace}. In this example the modification operator may not be included because replace is the default. Assume that confirmation on performing the operation is requested.

Because the mode is confirmed, a response will be sent by the managed system. The successful response may be either an acknowledgment with invoke identifier value 5 and no other parameters or in addition to the invoke identifier, may also include the value being set to as a result of the operation. This value may be the same as the one in the request

TABLE 4.6 Parameters of M-SET Service

Parameter Name	Req/Ind	Rsp/Cnf
Invoke identifier	M	M
Linked identifier	-	C
Mode	M	-
Base object class	M	-
Base object instance	M	-
Scope	U	-
Filter	U	-
Access control	U	-
Synchronization	U	-
Managed object class	-	C
Managed object instance	-	C
Modification list	M	-
Attribute list	-	U
Current time	-	U
Errors	-	C

TABLE 4.7 Example Use of M-SET Service

Parameter Name	Value
Invoke identifier	5
Mode	confirmed
Base object class	circuit pack
Base object instance	{equipmentId = 4, equipmentId = "TSICA"}
Modification list	{
Attribute Id	administrative state
Value	locked
Modify operator	replace }

or different. The latter case of returning a value that is different from the requested value is determined by the information model. For example, if a modem speed is set to a value that is somewhat different from the actual value it supports, it is possible for an implementation to set it to a value closest to the requested value and send a response with that value. If unable to perform the replace operation, an error response is generated.

4.2.2.4. Action Service. This service enables the invoking open system to request another open system to perform an action on one or more MOs in another open system. The action to be performed is defined as part of the information model for the resources. The parameters of this service are listed in Table 4.8.

As noted above, services except action and event report are defined similar to the database operations. The definitions of action and event report are not complete at CMIS level. Action provides a method to model aspects of the resource that are not defined in terms of attributes and read/write operations on them.[7]

Similar to the set operation, action requests may be issued in a confirmed or unconfirmed mode. The action type specifies the mode to be used in the request. Because the actions are specific to managed resources, at the generic level of CMIS the details of action information are not specified. Except for action type, action information, and action reply, other parameters have the same semantics described earlier. The action type is mandatory and must be appropriate for the managed object class. If action type is not defined for that managed object class, an invalid action type error is generated assuming confirmed mode is selected.

Unlike the other services, action service permits multiple responses for a request on a single object. This is useful, for example, in requesting that a test be performed on a resource. If the test has multiple steps or takes time to complete then intermediate responses on the progress of the test may be desirable.[8] In this case several responses may be associated with one request. If multiple objects are candidates to perform the request, then similar to get and set services individual responses are required for each selected managed object. In both cases the correlation is done using the linked identifier value. This mechanism is explained later in the section on *scoping*.

The action type is required under the condition action reply is present. The validity of the information in the response parameter action reply is determined by the value of the

[7]The choice of using a set versus action is in many cases a debatable issue because changes to the attributes may be done within either operation. Guidelines have been provided in the next chapter on this topic.

[8]Another approach is to acknowledge the request, and the results are later issued as notifications.

TABLE 4.9 Example Use of M-ACTION Service

Parameter Name	Value
Invoke identifier	6
Mode	unconfirmed
Base object class	circuit pack
Base object instance	{equipmentId = 4, circuitpackId = "TSICA"}
Action type	reset

TABLE 4.10 Parameters of M-CREATE Service

Parameter Name	Req/Ind	Rsp/Cnf
Invoke identifier	M	M(=)
Managed object class	M	C
Managed object instance	U	C
Superior object instance	U	-
Access control	U	-
Reference object instance	U	-
Attribute list	U	C
Current time	-	U
Errors	-	C

The managed object class parameter is mandatory as this defines the schema to be used in creating an instance. The *instance* refers to how the object is to be named. The different cases for assigning the name are: the managing system provides a name by populating the managed object instance parameter and allows the agent to select the name. The latter can again be done in two ways. A partial name can be provided in the superior object instance name. This option permits the managing system to specify that the new object must be contained in the object identified in the superior object instance field. Given the containing object, the agent may allocate a suitable name if no further information is contained in the attribute list for naming. In the second case the agent can assign the name according to its internal operations.

Let us consider the example of the circuit pack to discuss the various cases:

Case 1: The request does not specify managed object instance, superior object instance, or the value for the attribute equipmentId. Agent assigns the name relative to the slot in which the circuit pack is contained and informs the manager of the name.

Case 2: The request specifies the managed object instance field to be {equipment Id = 4, equipment Id = "TSICA" }. No further parameter is required as the complete name is provided. If the managed object instance parameter is supplied and a different superior object instance (other than equipmentId = 4) is provided, this will cause an error to be generated. To prevent this situation, the standard does not allow use of both fields in one request.

Case 3: The request specifies the value (equipmentId = 4) in the superior object instance field. This refers to the name of the slot according to the naming rules specified by the information model. The attribute list includes the value "TSICA" for

equipmentId. The combination of these two parameters are used to define the name of the new object.

Case 4: This case is very similar to case 3 except that the attribute list does not include the value for equipmentId. The managing system using its internal definitions allocates a value for this attribute thus forming the name of the new object.

While it is common in most information models to allow all the preceding four cases, there exist applications where the manager is not permitted to provide the name. One such application is the Trouble administration function for the X interface. The manager requests a trouble report be created for the facility leased from a service provider. In this case, the name of the trouble report is generated by the agent conforming to various policies defined for the environment. Another example where agent defines the name is when the naming depends on the physical architecture of the equipment.

In addition to these choices for naming a newly created object, it is also possible to create an object as a copy of another object except for the name. This is done by providing the name of the object to be copied from in the reference object name parameter. In order to successfully create a new object as a copy of another object, the class of the new object should match that of the reference object. It is also possible to override the values of the attributes in the reference object when creating the new object. This is achieved by providing the values of those attributes in the attribute list.

Irrespective of whether the new object is a copy of another object (so that override of values may be specified) or not, the attribute list parameter is used to specify the values of the attributes when the object is created. The attribute list is not always required. Specifically this is true in two cases. One case is the copy method described earlier. In the second case, default values may be available for the attributes as part of the schema or the agent alone can provide the values based on information not visible to the manager.

As this is a confirmed service, a response (success or error) is always generated. The name of the newly created object is required if it was not included in the request. However, if the parameter was present in the request (case 1), then it is not required to include this parameter in the response. Not all the cases are unambiguous relative to the condition stated in the standard, specifically cases 3 and 4. In case 3, even though the managed object instance parameter is not provided, the name is included in the request as components of two parameters. A safe approach for the sender is to include it in the response for cases 3 and 4. The receiver should be always capable of receiving the parameter in all cases.

The response to the request (assuming all values were provided including the name) may include only the invoke identifier. If the attribute list is present, then all attribute identifiers and their values must be included. In other words, let us assume that the request only provided some attribute values and allowed the rest to be assigned either using defined default values or by the agent. It is not acceptable in the standard to return only the values assigned by the agent and the manager to infer that all the values provided in the request are present in the newly created object.[10]

Consider an example where the managing system requests the creation of a circuit pack object. Assume that the invoke identifier for the request is 8 and the manager has not

[10]While this makes the manager implementations simple in not having to construct the attributes with what was in the request and what is in the response, resolve any discrepancies, etc., this is an unnecessary restriction. The manager should remember the values supplied and only need information that could not be provided in the request.

provided the name as it depends on the physical architecture. Table 4.11 lists the parameters and values associated with the response to this create request.

It should be noted that equipmentId = "TSICA" occurs twice here, once as part of the name and a second time as an attribute value. This is the result of the requirement in the standard that a complete list of all attribute values are to be included if this parameter is present.

TABLE 4.11 Example Use of M-CREATE Service Response

Parameter Name	Value
Invoke identifier	8
Managed object class	circuit pack
Managed object instance	{equipmentId = 4, equipmentId = "TSICA"}
Attribute list	{
	equipmentId = "TSICA,"
	adminstrativeState = unlocked,
	operationalState = enabled,
	alarmStatus = cleared,
	replaceable = yes,
	affectedObjectList =
	{ds1LineTTPId = 1,dis1LineTTPId = 2},
	lineCircuitAddress = 1}

4.2.2.6. Delete Service. This service is used to request that the managed open system deletes one or more MOs. Table 4.12 lists the parameters for this request. Similar to the create service, this service is a confirmed service.

If the request indicates that multiple objects are to be deleted, then a separate response is sent as discussed for the other services above. There are no new parameters or conditions for the parameters (when C is the status) beyond what has been described earlier. A simple

TABLE 4.12 Parameters of M-DELETE Service

Parameter Name	Req/Ind	Rsp/Cnf
Invoke identifier	M	M
Linked identifier	-	C
Base object class	M	-
Base object instance	M	-
Scope	U	-
Filter	U	-
Access control	U	-
Synchronization	U	-
Managed object class	-	C
Managed object instance	-	C
Current time	-	U
Errors	-	C

request to delete the circuit pack instance shown in the create example is provided in Table 4.13. The response in this case is an acknowledgment (assuming successful deletion) with the invoke identifier.

TABLE 4.13 Example Use of M-DELETE Service Request

Parameter Name	Value
Invoke identifier	10
Managed object class	circuit pack
Managed object instance	{equipmentId = 4, circuitpackId = "TSICA"}

4.2.2.7. Cancel Get Service. This confirmed service is used to cancel a previously requested M-GET service for which the complete response has not been received. This service is meaningful to use in the following cases. If the get request was addressed to multiple objects (intentionally or erroneously), the invoker may wish to cancel the request. This may be necessary because the request is taking too long to complete, or the required information is received and therefore additional responses are not required. Table 4.14 lists the parameters of this service.

The request includes an invoke identifier and the invoke identifier of the previously invoked get request. If the request is still outstanding in the managed system, then successful cancellation takes place. If the request is already completed by the time the cancel request is received, then an error is returned. At the time of sending the request, the invoker may assume that not all results are received. However, the internal operations to obtain the data may have been completed in the managed system even though they have not all been sent across the interface.

It should be noted that a generic service to cancel a previously issued request is available only with the get request. This is because a get operation is not a destructive operation. Canceling a read of the data does not change any behaviour of the resource. However, requests to modify values and perform actions are destructive operations and the side effect of undoing the request requires further consideration. The semantics of the operation in terms of how the resource behaviour changes as a result of the operation determine the appropriate undo request. These effects are outside the generic scope of CMISE and are to be included with information models.

TABLE 4.14 Parameters of M-CANCEL-GET Service

Parameter Name	Req/Ind	Rsp/Cnf
Invoke identifier	M	M(=)
Get invoke identifier	M	-
Errors	-	C

4.2.3. Errors

When a service is requested in a confirmed mode, a response is sent indicating success or failure in completing the request. Several errors are defined in CMIS, and they fall into the following categories. Some of the errors are general and applicable to all services. The category called management operation implies these errors are common across all ser-

vices belonging to the management operations model shown in Figure 4.1. Some of the error values noted for operations may not be applicable to cancel get service. Table 4.15 lists the names of the errors, a brief description, and the category. When an error value is appropriate only for a specific service(s), they are identified in the category column. The table provides illustrative examples of these categories and is not an all-inclusive list. The description column straddles between the phrases "service" and "operation." Even though the services are discussed in this subsection, the errors are better described in some cases in terms of the result of performing the requested operation (this includes both management operation and notification).

General errors such as "no such object class" and "no such attribute" may be used to mean more than one actual cause. Possible cases are discussed later. Further analysis beyond the error value will be required to determine the actual reason for error.

The errors are returned only as a response to the request (assuming an error occurred). If an error is present in the response (for example, the value of the managed object class in a response is not recognized), no errors are issued by the receiving system. The response may only be rejected. This prevents the possible chaining of responses to an endless loop by responding to a response.

In Table 4.15, some of the errors, even though listed in CMIS, are detected by the user of CMIS. For example, consider the case of invalid argument value. The event report and action services defined previously leave the details of event and action information unspecified. The parameters populating these two fields are dependent on the event/action type. In the example of equipment alarm from circuit pack, the event information such as perceived severity is defined by the Alarm reporting function that uses CMIS services. Suppose the values defined for the parameter "perceived severity" are critical, major, minor, and warning. A value "unknown" is out of the range. The error is therefore detected by the user of the service.

In addition to the generic errors, a mechanism to define context specific errors either for a specific function or a specific resource is included. This error is a signal to indicate further specialization may be available in a particular context. Consider the case of circuit pack. Assume that the resource is defined to perform diagnostics as a result of a reset request, and in performing the reset request, an error is encountered. Instead of sending a response that indicates "processing failure" (something happened when reset was performed) with no further information, augmenting with "Startup diagnostics failed" allows the managing system to determine the next step more easily.

4.2.4. Scoping Feature

The discussions earlier for M-GET, M-SET, M-ACTION, and M-DELETE services stated that a request may be issued to address multiple objects. The parameter scope shown in the tables is used to capture this functionality. If the parameter is not present, the default value in the request is directed to the single object identified in the base object instance.

In order to understand how multiple objects are selected as candidates for performing the request, let us take a digression and discuss briefly how the managed objects are logically[11] arranged in the information base. Figure 4.4 is an example of a tree of managed objects in a network element.

[11]The term *logical* is used to denote that this structure is not necessarily the database structure used in an implementation. Conceptually, the objects form a hierarchy stemming from the naming rules to be discussed in the next section.

TABLE 4.1[12] Examples of Error values

Error	Category	Description
Duplication invocation	General	Invocation identifier has been reused prior to the completion of a previous request with the same identifier. Because the manager assumes that messages are in order and acknowledgment is not received at the application level regarding the same request, to assume the error in transmission is not expected. Therefore this is an application level error for reusing the same sequence number.
Invalid argument value	Notification and action operation	Event or action information (depending on the service) is not valid. For example, the values of some or all parameters may be out of the defined range.
Unrecognized operation	General	The requested operation is not one of the CMIS defined services (e.g., event report, get, set, etc.)
Unrecognized event type	Notification	Event type is not defined for the managed object class referenced in the request.
Processing failure	General	An error has occurred when processing the request. This error value without further augmenting does not provide enough information. See description below.
Access denied	Management operations	Access privileges for the requested operation are not available. Depending on the service (operation), access is denied to reading the attributes, accessing the object itself, modifying the attributes or requesting an action be performed. This error is usually considered in the context of security support.
No such object class	General	The managed (base) object class referenced in the service request is not known to the receiving system. This error is used in management operations if the request refers to a class not recognized by the managed system. In the case of event report service, the managing system does not recognize the name of the managed object class. Note that in some cases the error may be because the revisions of the definitions used by the sending and receiving may not be the same.
No such attribute	Create, get, and set operations	The attribute is not recognized by the managed system. This does not mean the attribute does not exist. The object class (create) or the object instance selected for the operation does not include this attribute. If, for example, an attribute is defined for a class to be present as an option, a particular instance may have been instantiated without this option, or the attribute may not be applicable for that class.
Synchronization not supported	Management operations except create	This error is applicable only when the request is directed to select multiple objects. The type of synchronization (atomic or best effort) requested is not supported.
No such invoke identifier	Cancel get	The cancel of the previously issued get operation cannot be completed. This scenario is possible in cases where the specified invocation identifier for the request being canceled is incorrect or the previous request has been completed and therefore it is not available for cancel.
Invalid operator	Set	The modify operator specified is not applicable for the attribute specified in the request. For example, if an attribute has a single integer value and the request is to add another value, this is an invalid operation.

[12]Management of the invocation identifiers is required to assure that duplicates are not used. CMISE standard itself does not define a mechanism. The simple approach used in some implementations is to increment a value for each request until it becomes a very large number. If communication is lost between the systems, the outstanding invocation identifiers are not remembered.

TABLE 4.15 Examples of Error values *(cont.)*

Error	Category	Description
Class Instance conflict	Management operations	As seen by the parameters for all service requests, the presence of the managed object class is required. Except for create, instance name is also required. This error indicates that there is a mismatch between the name of the object and the value of the class specified in the request. For example, if the request contains the class "fabric" and the name refers to a circuit pack, this is an appropriate error.

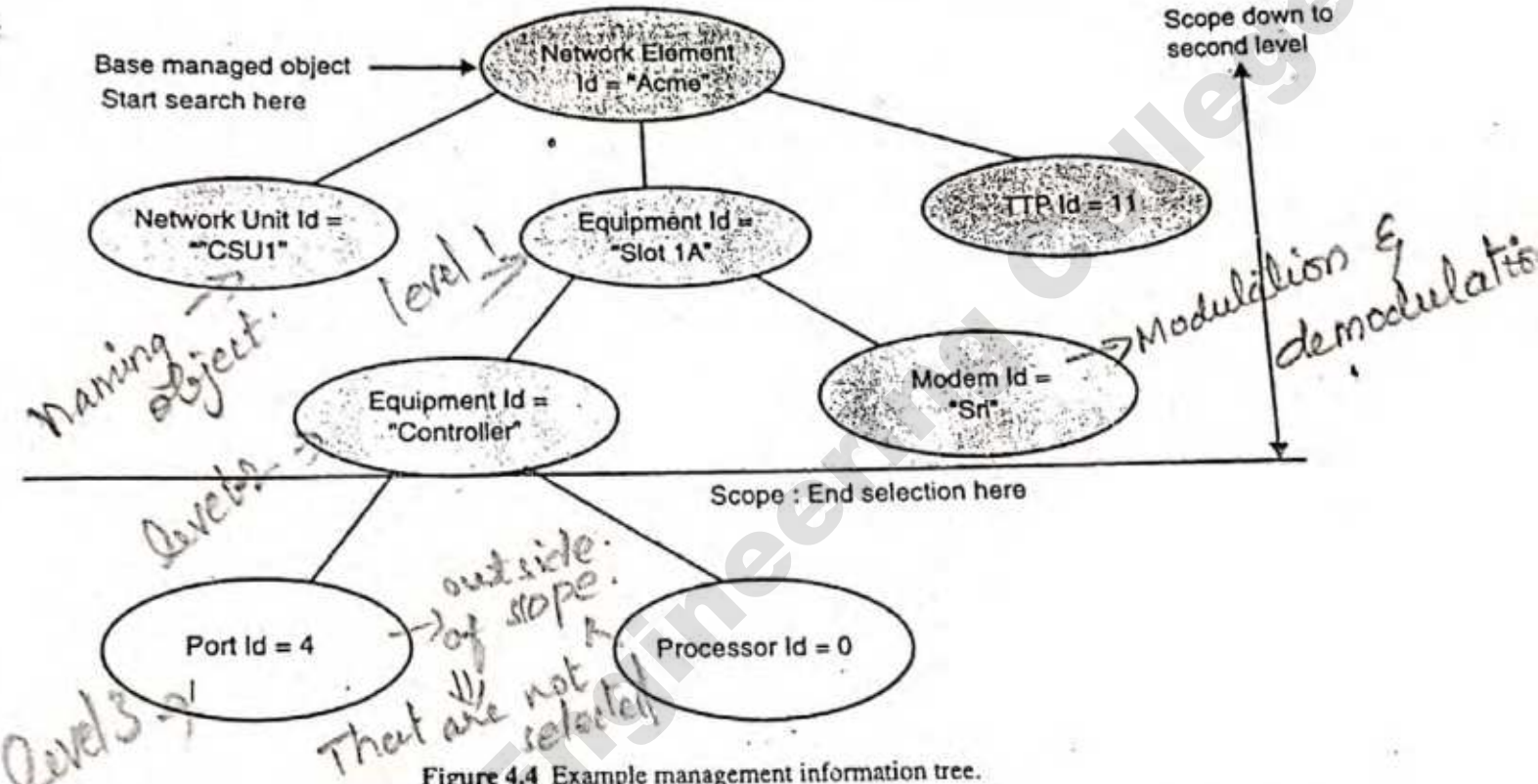


Figure 4.4 Example management information tree.

The objects in the figure are shown as ellipses. The value of the attribute used for naming the object is shown inside the text. In this example, the network element named as "Acme" contains one slot. The slot can hold two cards; each one of which may be managed independently. One card is a controller card, and the other card performs modulation/demodulation of the transported signal. Another object considered as part of the network element is a subscriber unit located close to the customer.[13] Let us assume that interface between the two cards is not visible to the managing system. The termination point 11 corresponds to where a signal (DS1/E1) from a switch is terminated. Using this tree every object gets a unique name.

The scope parameter in the above example has a value of 2. This implies that all objects relative to the base object (in this case the network element) that are within two levels of the hierarchy are candidates to perform the requested operation. Objects that are outside of this scope are not selected. If there is no further criteria to eliminate any of the object, then everyone within this scope will perform the operation. All the objects that are selected within

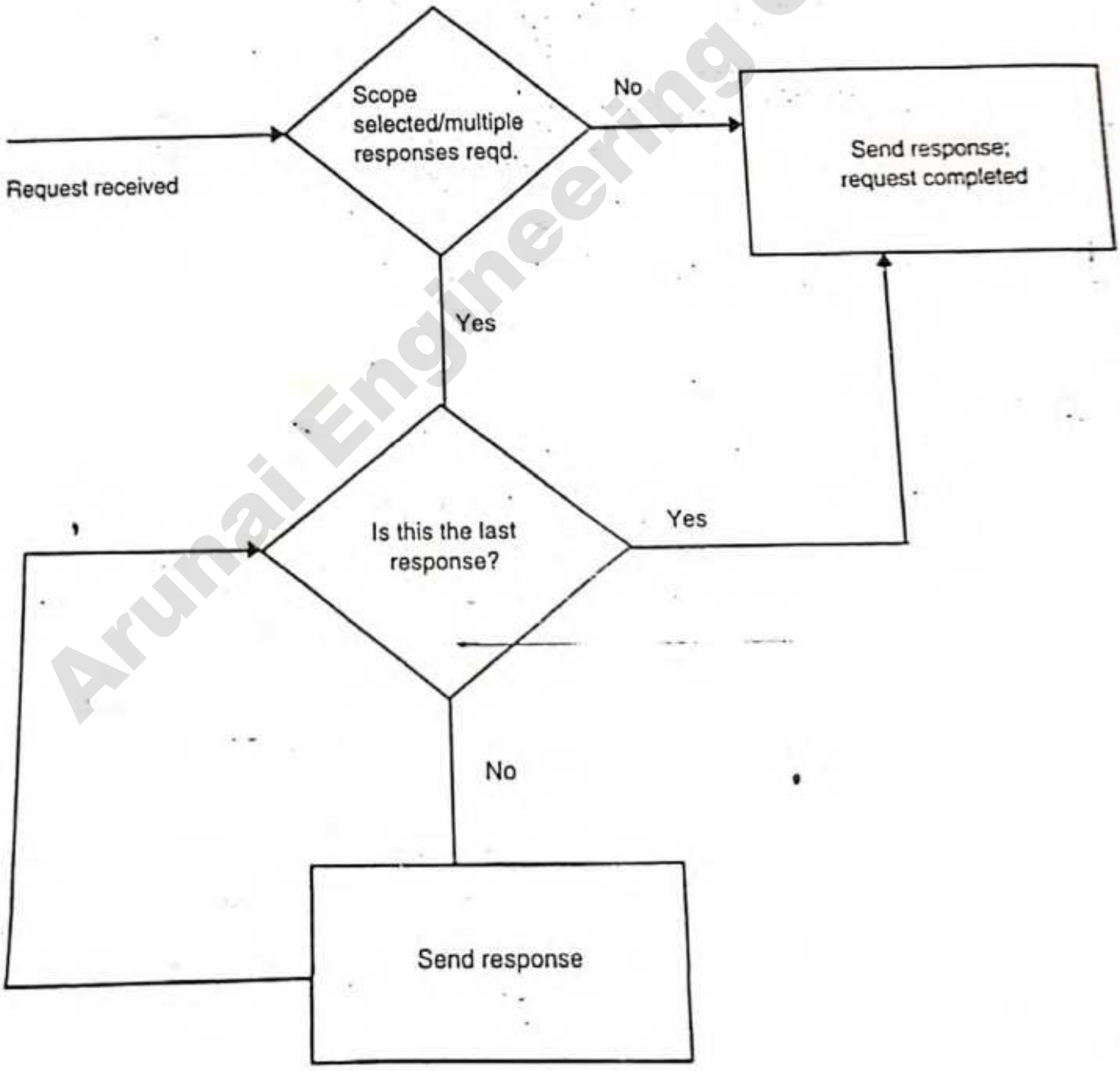
[13]This example is a case where the network element is geographically distributed between the central office and near the customer location.

the scope of 2 relative to the base (starting node for the search) are shown as shaded ellipses in the figure.

The base object in the request may start at any point in the tree. If a managing system wants to retrieve the names of all the cards contained in slot 1A, then the base object should be set at the object representing the specific slot. Different values may be specified for the scope parameter. These values allow the selection of objects at a specific level (for example, second level only), base and all objects within a specific level (the previous example of up to level 2), and the entire subtree (in the previous example all objects including the nonshaded ones).

Once the scope for selecting the objects is established, let us consider how the responses are returned and correlated. It was mentioned earlier that an individual response is returned for every selected object. Consider that the manager uses the scope shown in Figure 4.4 and issues a get request. Let us assume a simple case where attribute identifiers are not explicitly included in the request. As stated in the M-GET service description, this implies all attributes pertaining to each of the selected objects are to be returned.

Figure 4.5 illustrates the process for selecting the multiple objects and responding to the request. This process is further illustrated in Figure 4.6 using the interactions between the managing and managed system.



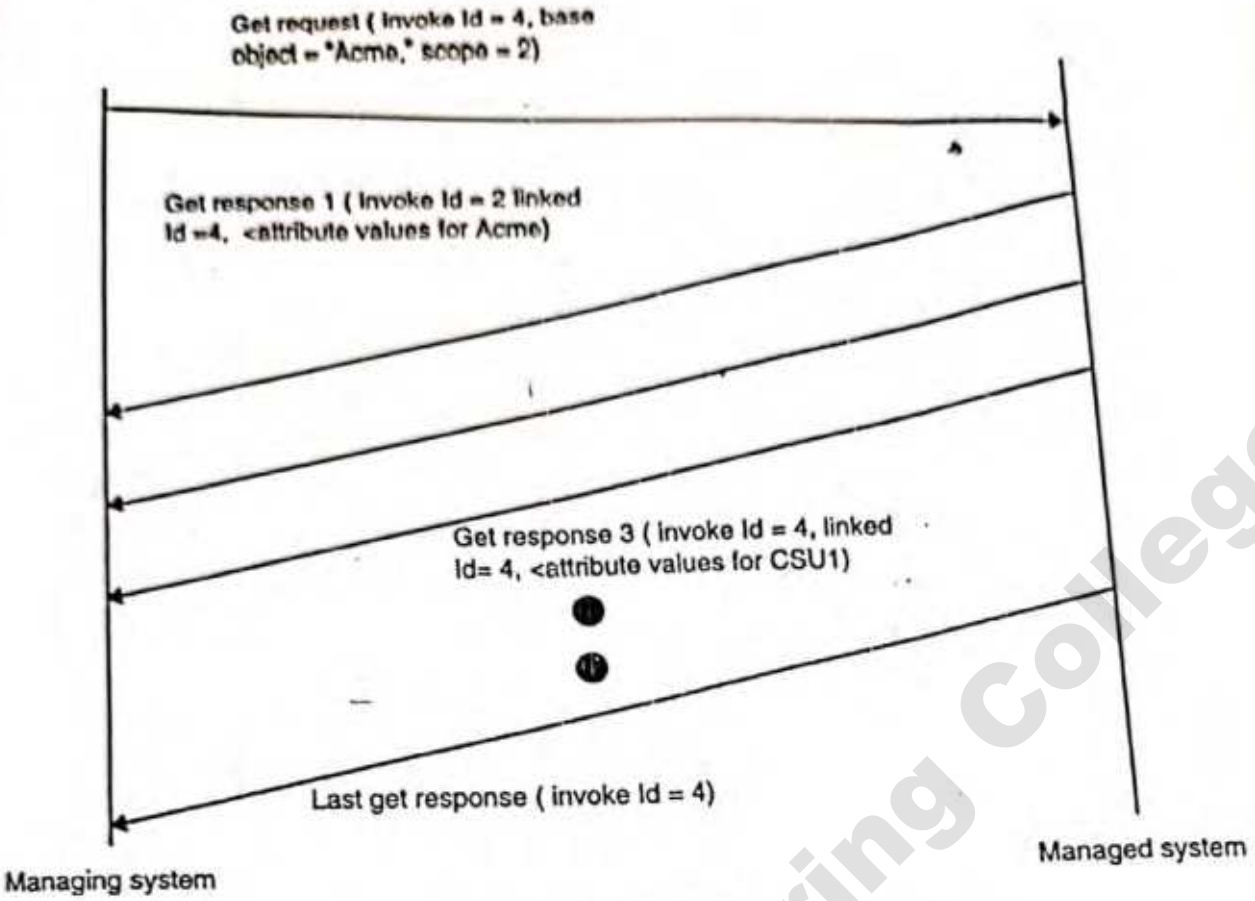


Figure 4.6 Correlation of multiple responses to request.

Figure 4.5 is not specific to selecting multiple objects via scoping. As pointed out earlier one service where multiple responses are possible without selecting multiple objects is an action. The requirement for multiple responses in this case depends on the action definition. In contrast, if scoping parameter is specified in M-GET, M-SET, M-ACTION, or M-DELETE service, and the value is such that multiple objects are selected, then it is required to send multiple responses. In Figure 4.5, the process description as seen from the system receiving and responding to request is described.

During the discussions of the service parameters, the use of invoke identifier to correlate the request and responses was identified. The value of the invoke identifier supplied in the request is returned in the response. This allows, for example, the managing system to issue requests without waiting for a response. The correlation can be done whenever the response is received by using the value of the invoke identifier. Figure 4.6 describes how this correlation is performed by the managing system. Let us assume that the request contains a value for invoke identifier as 4. Let us also assume that request is a get operation using a network element as the base object and the scope level as 2 shown in Figure 4.4. Assume that the get request is issued such that values of all attributes for the selected objects are to be retrieved.

Figure 4.6 defines the structure for the multiple responses corresponding to each selected managed object. Each response specifies a value for invoke identifier which can be any value. For all responses except the last one, the invoke identifier is not used for correlation. Each of these responses includes the parameter linked identifier listed in the service definitions above. The value of this parameter equals the value of the invoke identifier in

the request. As long as the linked identifier parameter has a value that matches with the outstanding request, then this value is used to correlate, for example, get response 1 with the request. When multiple objects are selected, the receiving system needs to know when all replies have been received and the request has been completed by the managed system. There are two alternatives to close the request. Once all the responses are issued, the managing system can send a closing response with only the invoke identifier parameter and no linked identifier. The value of this invoke identifier equals the value in the request. This signals the completion of the request. Another approach allowed by the standard (though more difficult to implement) is to use in the last response the value for invoke identifier that equals the value in the request and the data associated with the last selected object. This implies that agent needs to track the responses and determine which is the last response so that appropriate value can be inserted in the invoke identifier. Instead with the first approach (shown in the figure) all responses are sent using linked identifier and when there are no more to send close the request with a response containing only the invoke identifier and no other data.

In the figure the third response uses a value for the invoke identifier which is the same as that in the request. This is not an issue for correlation because it is the linked identifier that is used for correlation. The number space for invoke identifier in the responses are assigned by the managed system and therefore they are independent of the space used by the managing system. Therefore there is no conflict if the same number is used. However, between the multiple responses to the same request, the managed system should use different values for invoke identifier and cannot reuse the same value. Strictly speaking, the invoke identifier is not required as long as the linked identifier is present. However, this will assist in determining if any responses are lost either because of a communication failure or an application not receiving the response. In order to perform this analysis, management of the invoke identifiers must be available.

Let us now consider how Figure 4.6 may also be used with multiple responses to an action request on a single object. Here again the linked identifier is used to do the correlation. The only difference is there is no requirement to include the managed object class and instance parameters in the multiple responses. These parameters are optional. By definition of the request, the responses are from the same object and therefore not required by the managing system to analyze the response. The last response can be a closing response with only the invoke identifier or contain the data as discussed for the multiple objects scenario.

As a side note, the use of linked identifier in CMIS is somewhat different from how it is used in other protocols like Transaction Capabilities (TCAP) in SS7. The purpose in TCAP is to invoke a child operation in response to a parent operation. The child operation may request additional information prior to completing the request. In CMIS, the child operation request is a response. The reason for this note is to point out for those familiar with TCAP the usage is somewhat different in this case. Readers not familiar with TCAP may ignore this as this does not impact how multiples responses are structured in CMIS.

4.2.5. Filtering Feature

Database management systems often support the mechanism to request that an operation be performed if a criteria is satisfied. This feature is provided in CMIS using the filter parameter identified in the tables for M-GET, M-SET, M-ACTION, and M-DELETE services. The criteria is specified in the filter parameter in terms of the test to be on one or more attributes. These tests are grouped as a logical expression using zero or more of the

following operators—and, or, not. The test on each attribute is defined in terms of one or more of the following:

- Is the value of the attribute equal to a given value?
- Is the value greater than or less than a given value (the general term used for these two cases is *ordering*)?
- Is attribute present or absent?
- Are the values a subset or superset of the given set of values?
- Is the intersection of the given set with the values of the attribute an empty/non-empty set?

If the result of the testing yields a true value, then the operation is performed on that object.

The selection using filter parameter is applicable both in the case of single and multiple objects. Even though it is a more common practice to combine filtering with scoping, it is possible to request an operation on a single object be performed subject to a criteria being met. Consider the case of the reset action defined earlier. Assume that in order to perform the operation, the administrative state of the circuit pack must have the value "locked." The request to perform the action can include in the filter parameter the assertion (administrative state = locked) to assure that the request is performed only if this condition evaluates to true. If this condition is not satisfied, the action is not performed. Evaluation of the condition to false is not an error. It means the object was unable to satisfy the requested criteria and therefore the action was not performed.

Let us now consider the more complex case where multiple objects are selected using the scope parameter. Figure 4.7 expands Figure 4.4 to include the attributes and their

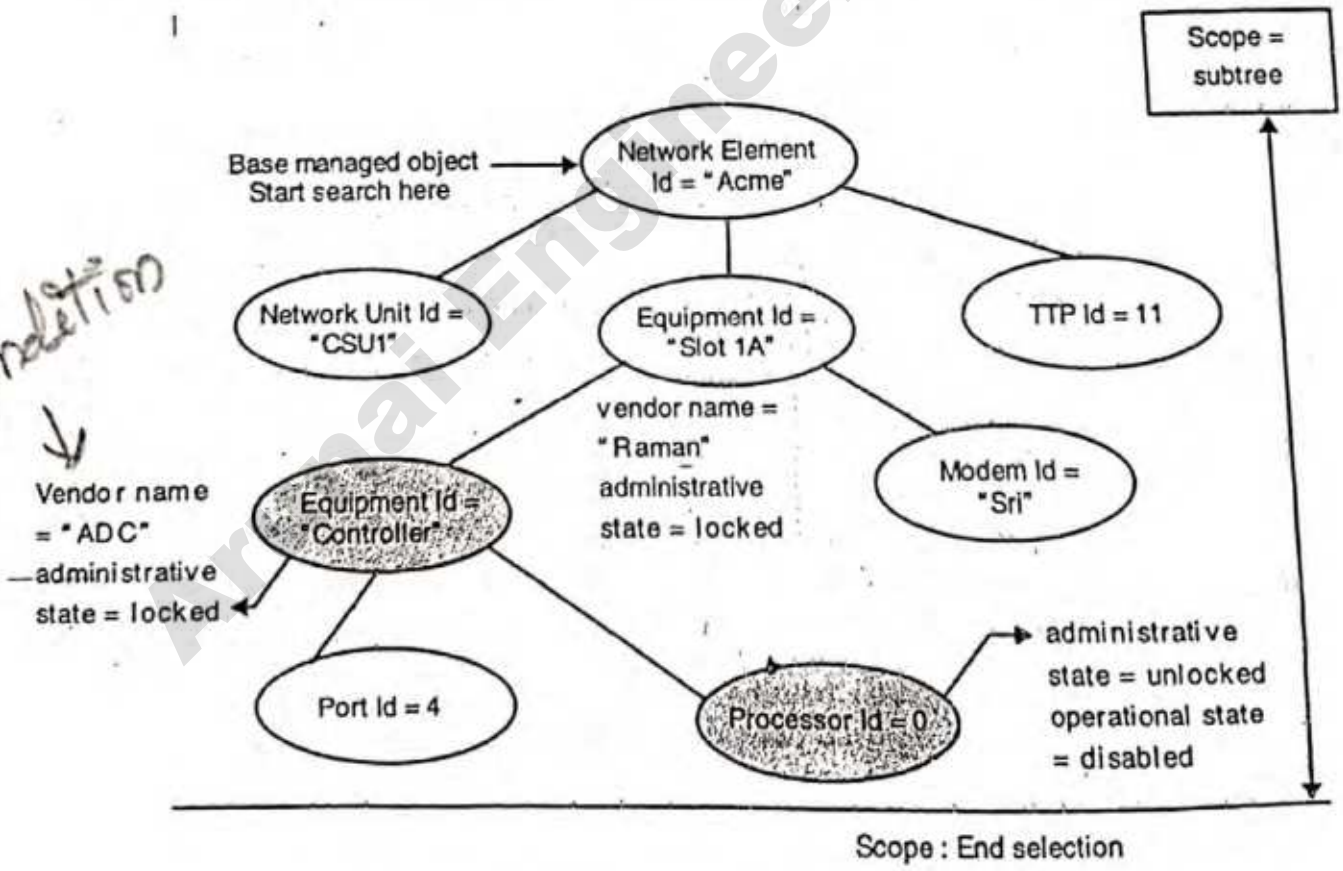


Figure 4.7 Example combining scope and filter features.

values. Let us assume that scope parameter indicates the subtree is to be included in the search. This implies the filter criteria is to be applied on all the objects in the subtree. Assume that the conditions on the attributes which should be tested is specified in the filter parameter as follows:

```
( (object class = equipment AND vendor name = "ADC") OR (object class = processor
AND operational state = disabled))
```

The result of applying the scope value of subtree to the tree of objects shown in Figure 4.7 equates to selecting eight objects as candidates for performing the operation. Without further constraints in filter parameter, this would result in eight individual responses. The application of the test yields the two shaded objects as the ones that meet the criteria. Even though there are two equipment objects that satisfy the condition object class = equipment only one of them meets the additional condition for vendor name. The test indicates either this condition or a second condition must evaluate to true. Two individual responses will be sent for the controller and processor (0) objects. As explained in the section on scoping, the number of responses are either two individual ones followed by a closing response or two responses where the second one also concludes the request by not using the linked identifier and using for the invoke identifier the value in the request.

Combining scoping and filtering features offers a powerful object selection mechanism with CMISE which is not available with other network management protocols. These features are very useful in network elements where there are several instances of objects such as termination points for the incoming and outgoing signals. Instead of having to send individual requests, a single request with scope and filter reduces the amount of traffic on the interface. This feature is used in some OS-NE applications to synchronize the data base of the management information in an NE with the database in the managed system (OS).

4.2.6. Synchronization

The synchronization parameter goes hand-in-hand with the scope parameter. Once multiple objects are selected (either using only the scope parameter or a combination of scope and filter parameters), this parameter can be used to request that the management operation be performed in a best efforts or atomic manner. The default method is the best effort which implies that the operation is performed on all objects without requiring to verify whether all objects are capable of performing this request successfully. Performing the requested operation (successful or otherwise) on one of the selected object does not impact the operation on another object. In contrast, the value of "atomic" for the parameter indicates that the operation is performed only if all the selected objects are capable of successfully performing the request.

Consider in the previous example that a request is issued to set the administrative state to lock as an atomic operation. Assume that the conditions are such that processor and controller objects are selected. In this case if it is not possible to set the administrative state to locked in both the objects then the operation will fail. This parameter does not address synchronization requirements among the attributes of a managed object. For example, if the circuit pack definition requires that a fixed relation exists between the type and the number

of ports available, and if the value of the number of ports is changed along with the type, then the requirement that the two changes be done together is not enforced by using the synchronization parameter. This is done in the definition of circuit pack object and will be discussed in the next chapter.

4.2.7. Functional Units

The services and features described in the previous sections are not always required by every application and every association between the managing and managed system. The concept of functional units was introduced in Chapter 3. Grouping services into units of functionality that are negotiable for use during an association is present in OSI protocols at the session, presentation, and application layers (not in all applications). The basic services are grouped together into a functional unit referred to as *kernel.* The name is used to indicate that the associated services are not subject to negotiation. Any implementation must be capable of supporting kernel in order to claim conformance to the service element. Table 4.16 defines the functional units for CMISE and the associated set of services/features.

Because the kernel functional unit is mandatory (not negotiated) in any association, an application using CMIS services may invoke any of the basic services without some[14] of the optional parameters listed in the tables. If an action for the managed object includes issuing multiple responses, this cannot be supported with only kernel.

Dependencies exist between multiple object selection and multiple reply functional units. Because the responses for multiple objects selected by using the scope parameter are returned individually, the support for multiple reply functional unit is required when multiple object selection is negotiated. However, it is possible for an association to use multiple reply functional unit without the multiple object selection to support action types that are defined with more than one response.

Even though no restriction is placed on the cancel get functional unit (may be used with only kernel to cancel an outstanding get request), the more practical use is when multiple responses are sent to a get request. The standard has identified this as a flow control mechanism when receiving large amounts of data with multiple replies.

The CMIS standard provides very little information on the extended service functional unit. This is not used in any TMN implementation or specifications based on the communication profile chosen for the presentation layer. It is explained here for completeness even though it is not relevant in the TMN context. At the presentation layer three functional units have been defined. The kernel functional unit includes the minimum services to set up and release presentation connection, negotiate the encoding rules, and provide for data transfer. Without going into the details of the presentation layer, two other functional units are available. These are known as *alter presentation* context and *restore presentation* context. These services include user data in addition to presentation layer specific control information. If extended service is negotiated then management information defined in CMIS can be included in the user data of the services supported by these two functional units. The actual mapping and other required procedures when these services are used have not been

[14]"Some" is used to indicate that there are certain parameters that are user options and not related to functional unit definitions. Examples are access control and current time. Parameters such as scope and filter are included in the functional unit definition.

TABLE 4.16 Functional Units in CMIS

Functional Unit	Services/Feature
Kernel	M-EVENT-REPORT, M-GET, M-SET, M-ACTION, M-CREATE, and M-DELETE services are available for use
Multiple object selection	Use of scope and synchronization parameters is permitted
Filter	Use of filter parameter is permitted
Multiple reply	Linked identifier parameter may be used. This allows several replies be sent for a single request.
Extended service	Presentation layer services other than P-DATA may be used to transfer CMIP protocol data units
Cancel get	M-CANCEL-GET service is available for use

defined. The standard leaves this responsibility to the definition of an application context which does not exist today in either TMN or ISO standards.

4.2.6. Association Services

The services described as M-XXX define the parameters for transferring management information. Prior to exchanging management formation, it is required to establish an association between the application entities in the two systems. Chapter 3 described the Association Control Service Element (ACSE) used for this purpose. ACSE protocol includes a parameter that is used by other application service elements to negotiate the features and other information for use during an association. The user information parameter is structured to include information required by the various application service elements forming the application entity. As pointed out in Chapter 3, the ASEs for Network Management Application Entity supporting interactive applications are ACSE, CMISE, and SMASE.

Three parameters are defined as part of CMIS for use during an association setup. These are functional units, access control, and user information. The functional units are one or more of those defined in the previous section. It is also possible that there exist agreements between the communicating entities that certain functional units will be used so that there is no need for negotiation. Another approach may be the use of network management profiles.

The access control is used to validate the managing or managed entity depending on which one initiated the association. In most of the implementations today, the association setup is requested by the managing system. One application where the access control parameter is used in implementations for authenticating the identity of the manager is the *Trouble administration function*[15] on the X-Interface. Successful validation based on the value of this parameter at association setup establishes default privileges applicable during the association. The same parameter as discussed above is also included when requesting operations. The operation specific parameter validates the privileges in the context of a specific operation.

Similar to the user information parameter in ACSE, the user information parameter in CMIS is used to include information required by the user of CMIS (system management application) for negotiating features during association setup. In practical implementations

[15]The family of applications between inter-exchange and local-exchange carriers developed within North America is referred to as *Electronic Bonding*. This approach of access control is used to authenticate the manager in applications within this category.

of CMISE and TMN, this parameter is not used. This is because as pointed out in Chapter 3, the negotiation of System Management application features are included as a separate field in ACSE user information instead of embedding it within CMIS user information.

In addition to the three parameters visible to the user of CMIS, a fourth parameter is also defined by the protocol for transfer during association setup. This parameters allows the negotiation of protocol versions. The reason that this is a parameter defined only in CMIP relates to keeping the service and protocol specifications separate. The user is not concerned with the version of the protocol but in the capabilities offered by the protocol.

The four parameters have either a default value or are defined as optional. If the functional unit field is absent, this implies that only kernel is available for use on that association. It is, however, possible to use the features corresponding to the negotiable functional units if agreements exist outside of the negotiation process. The access control and user information parameters are optional and can be omitted. The protocol version has a default value of version 1.

4.2.9. Protocol Specification

The common management information protocol (CMIP) is defined for exchanging the management information using the aforementioned services. The protocol is defined in terms of the request reply paradigm offered by the Remote Operations Service Element (ROSE). Services offered by ROSE are generic for any application using request/reply based interactions between the communicating systems. The generic definition of ROSE are specialized by CMIP for interactions specific to management applications.

As with any OSI application layer protocols, both ROSE and CMIP are specified using the notation Abstract Syntax Notation One (ASN.1). This is a high-level language for specifying protocol data unit definitions. The notation includes several data types such as integer, boolean, character string and allows construction of new types using the base types. The syntactic constructs offered by ASN.1 are very powerful. This facilitates the specification developer to concentrate on the semantics of the application and not be concerned with the octet level representation. Because rigorous rules are followed in defining the protocol data units using ASN.1, the specification is machine processable. Different encoding rules may be applied to generate automatically the octets exchanged on an interface. The reader is referred to books and standards listed in the reference section on ASN.1 to gain further understanding of the topic.

Because the "message" exchanged for network management using CMIP is built on the structure from ROSE, the next section describes ROSE protocol. CMIP structure is defined using this protocol and corresponding procedures.

4.2.9.1. ROSE Protocol Structure. The request/reply paradigm of ROSE is defined using four protocol data units. These are *invoke* an operation, *reply* with successful result of performing the operation, *respond* with error in performing the operation, and *reject* the request because of problems in the data contained in the request or response (successful or error). The structure of the four protocol data units are shown in Figures 4.8 (*a–d*).

The invoke Id parameter (same semantics as for Invoke Identifier field shown in CMIS services parameter tables) is used to correlate request with the response. The invoke operation in Figure 4.8(a) contains the mandatory parameters invoke Id used for correlation and operation value. Because ROSE PDUs are generic for any application using the

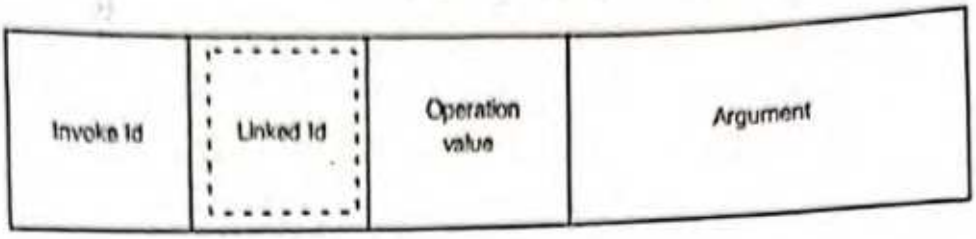


Figure 4.8a Invoke an operation.

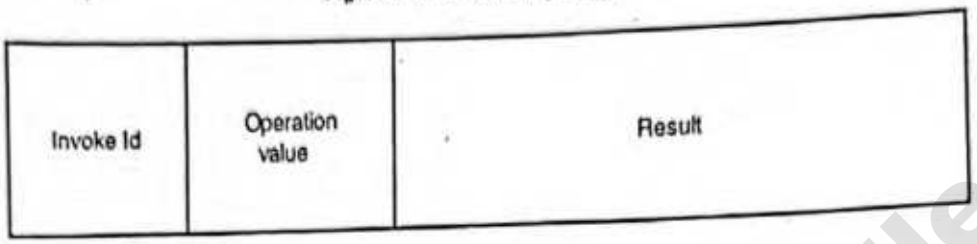


Figure 4.8b Successful response.

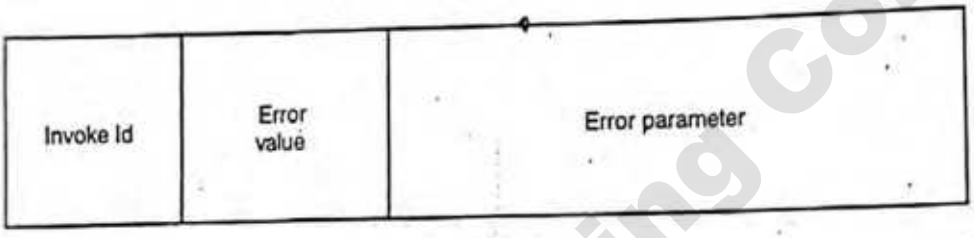


Figure 4.8c Error response.

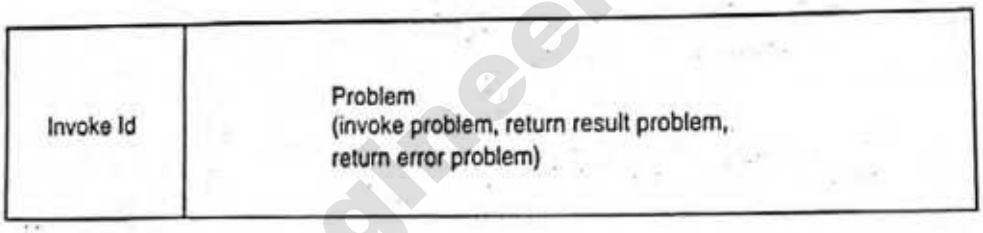


Figure 4.8d Reject a request or response.

request/reply paradigm, the operation values are not specified even though the syntax is specified. The operation value is either an integer or a globally registered value. Depending on the application, values are assigned to the operation. The argument field is dependent on the value of the operation. At ROSE level, a placeholder is present for including parameters appropriate to be included in the request. Even though there is only one parameter called argument in ROSE PDU, this may be translated to multiple parameters for any specific operation. The linked Id parameter is optional and is shown as such by enclosing with a dotted line. ROSE included this parameter in the invocation request to inform the receiver that this is a child operation. Assume that a request was sent from one application entity to another invoking an operation using the parameters invoke Id, operation value, and argument. If the receiving entity requires further information to perform the requested operation, a child operation may be initiated. In this case, the new invoke operation includes an invoke Id for this request. The linked Id contains the value of the invoke Id in the parent operation request. This parent–child relation between multiple operations is used in a specific manner in CMIP. The multiple responses to the same request is exchanged using the linked Id mechanism. This is confusing because the responses are sent using invoke child operations mechanism, even though valid according to the procedures for the protocol.

The successful result PDU structure requires that the invoke Id parameter be always present. This is used to correlate the request with the response. Depending on the operation, the response may be just an acknowledgment or include result information. In the latter case, the operation value must be present. This allows the receiving entity to determine if the response is valid for that operation. The valid parameters for the request are specified with the definition of operation.

An error response must include invoke Id (for correlating to request) and an error value. The error values are specified with the definition of operation. An error value is an integer with specific semantics. This value may be further augmented using additional information included in the error parameter. For example, suppose the affected object list in the circuit pack is being modified using set operation request. An error response may include in the error parameter the information "administrative state is unlocked" to augment the error value "processing error."

Any of the three (invoke, return successful, return error) protocol data units may be rejected by the receiver if invalid. The problem values are grouped into three categories as shown in Table 4.17.

Remote operation services may be used in a *synchronous* or *asynchronous* mode (referred to as operation class). If the operation is defined as the former category, invocation of another operation awaits the response to the previous operation. In the latter case, operations may be invoked irrespective of whether a response has been received. The mode used by CMISE is the asynchronous class.

Three association classes are defined for determining the permissions for invoking and responding to operation requests. These classes are coupled to how the association between the application entities are established. Class 1 restricts the invocation of the requests to only the initiator association. Class 2 corresponds to the case only the responder of the association can invoke an operation. Class 3 does not place any restriction—both the ini-

TABLE 4.17 Example Problem Values

Category	Problem Values	Description
Invoke problem	Duplicate invocation	Invoke Id used violates the rules for reuse. There is an outstanding request with the same value for which a response has not been received.
	Unrecognized operation	The operation value is not valid for the application using ROSE. The value supplied is not one of the values assigned by specification for that application.
	Resource limitation	There are no resources available to perform the requested operation.
Return result problem	Unrecognized invocation	The result is being returned with an invoke Id that does not correspond to an operation in progress. The invoke Id does not correspond to any outstanding operation request.
	Return response unexpected	No response is defined for the operation in question. Receiving a response is therefore invalid for the invoked operation.
Return error problem	Unrecognized error	The error value is not recognized in the context of the application using ROSE. The error value is not valid for the current application
	Unexpected error	The error values is not valid for the invoked operation (identified by the invoke Id)

tiator and responder may invoke the operation. Based on the services described previously for CMIS, it is obvious that either the initiator or responder to association request should be allowed to invoke an operation request. Assume a managed system needs to send an eventreport. The association should be set up before issuing the event report. This may be done either by the managing system or the managed system. Once the association is established by the managed system, the managing system may send a query to read attributes of a managed object. Thus the invocation of the operation is independent of how the association was established.

Given the framework for invoking an operation in a remote entity, let us consider the use of this structure in the context of the management protocol CMIP.

4.2.9.2. CMIP Protocol Structure. The various services described for CMISE are supported in CMIP as follows: assigning operation values and associated information in the argument field for invoke, defining the result information if valid for the operation, identifying the valid error values (see the examples in Table 4.15), and any associated parameters. In order to provide a semantic link between the various components for any specific operation, a notational technique (ASN.1 macro facility) defined by ROSE is used.

The generic structure for the four PDUs are augmented, as shown in Figure 4.9. All operations (includes management operations and notifications) for the invoke argument contain the reference to the managed object in terms of the class and instance.

Using the notational facility in ROSE, CMIP augments the remote operation definition with the following specifications. The operation values are assigned as shown in Table 4.18. The argument for invoking the operations, the result information, and the error appropriate for each operation are specified. In some cases where multiple responses are possible, this is also included in the definition. As shown in the generic structure above, all operation requests (irrespective of management operation or notification) include two parameters—class and instance of an object. Depending on whether scope parameter is present, this refers to the resource being managed or the root of the subtree for selecting multiple objects.

The operation-specific information is either completely defined or placeholders are left for further specification by the network management application using CMIP. To illustrate this let us consider the examples for get and event report.

The get operation is defined in terms of the following components: get argument, information in the argument field of the RO-invoke PDU, get result, information in the RO-RESULT PDU, list of errors values (further expanded in terms of the parameters associated with each value), and reference to the linked reply operation for multiple responses when required. The parameters of get argument are: base object class, base object instance, access control, synchronization, scope, filter, and list of attribute identifiers (names of the attributes). The semantics of these parameters are identical to that defined in the service description. The operation specific information is completely defined in this case in terms

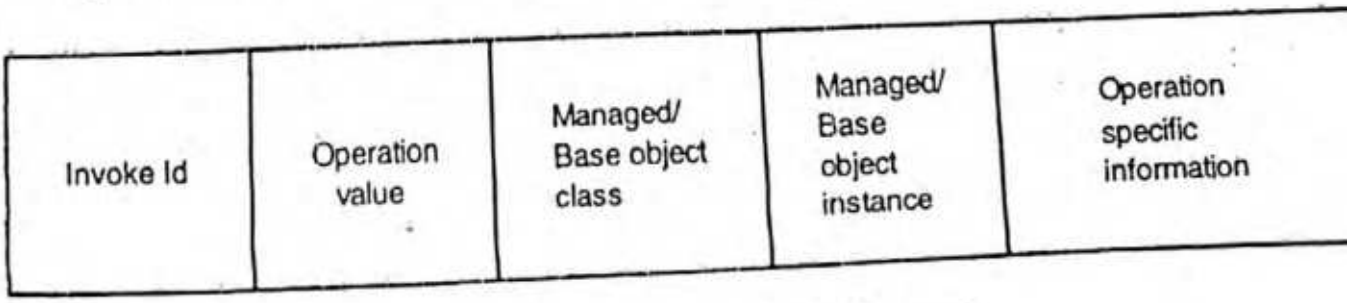


Figure 4.9 Generic structure of CMIP request.

TABLE 4.18 Operation Value Assignments in CMIP

Name of the Service	Operation value
Event report	0
Confirmed event report	1
Multiple responses*	2
Get	3
Set	4
Confirmed set	5
Action	6
Confirmed action	7
Create	8
Delete	9
Cancel get	10

of the syntax of every parameter. The appropriate values for the attribute identifiers depend on the managed object class in the request and is left to the information model. However, no further specification beyond CMIP is required for the syntax of the argument. For the result except the syntaxes for the values of the attributes (this is governed by the specific attribute types) the syntax of the information is well-defined.

In contrast, let us consider the case of event report. The components of the argument field are managed object class, managed object instance, event time, event type, and event information. The syntax of all the elements except event information is defined by CMIP. The event information parameter is just a placeholder and further specification of the syntax is left to the application. For example, the alarm reporting application specifies that when event type is communication alarm the parameters are: perceived severity, probable cause, whether the resource is backed up or not, current state of the resource, trending of the severity, diagnostic information, and specific problems. Another event like threshold crossing alert may include a different set of parameters and is defined by the performance monitoring application.

In both classes of definitions, the paradigm used is apparent from the generic structure and paves the way to the next chapter. All CMIP operations are specified in terms of the managed objects representing the resource being managed. The structure follows the object-oriented design principles in that the requests are made to the object at its boundary and how these requests are performed is determined is an integral part of the object definition. The protocol data units may be thought of as "messages" to the object. As such the challenge in the network management paradigm using CMIP is the development and understanding of the information models instead of the protocol itself. CMIP with ROSE is a simple request/reply protocol to carry the management information. The semantics of the information is determined by the resource, and the emphasis therefore is on the information models.

There are two protocol versions available in CMISE. Even though version 1 is the default this is almost never used in any known TMN application. The main difference in the functionality relates to modification operators add and remove in the set operation and

*"*Multiple responses* is not service in the sense of CMIS services mentioned earlier. This therefore strictly does not fit under the title of service. However, because linked responses are required when the scope parameter in the service is used, this is indirectly related to the characteristics of a service.

the addition of cancel get operation. There are other minor syntax differences that will affect the interoperability. Any information model defined in the standard to date have set valued attributes and require the add and remove operators. Protocol version 2 is specified as requirement for TMN interfaces in ITU Recommendation Q.812. Because version 1 is defined as default and is nonusable with any models available for TMN applications, all implementations are required to send this field during association setup. It is possible that some implementations, because of agreements outside of an open interface may choose not to send this parameter even though version 2 is used.

4.2.9.3. Conformance. The standards specifies the support of kernel as a minimum required for an implementation to claim conformance to the standard. As noted earlier kernel includes all the basic services without use of certain optional parameters. Kernel by definition is not negotiated and is always assumed to exist when CMISE is included in the application entity. However, it is possible for an implementation to specify in the conformance statements the services supported which may be a subset of those in the kernel. This is usually determined by the application. For example, an association may be dedicated to receiving events and another association may be set up to perform other interactions. This approach may be used as a load balance measure so that alarms are not waiting in the queue because responses are being sent for a previously received get request. The assignment of an association for events will be agreed between the application at the CMISE user (SMASE) level instead of subsetting the kernel functional unit.

Conformance specifications are defined in terms of *static* and *dynamic* requirements. The static requirements address the rules associated with CMIP. Examples are the availability of the kernel functional unit, use of ACSE for setting up the association, and the dependencies to be adhered to when selecting functional units. The dynamic requirements are associated with the procedures for exchanging the protocol data unit and handling of optional parameters when used in a PDU exchange.

The conformance to the protocol by an implementation is specified by completing the Protocol Implementation Conformance Statements (PICS) documented in Recommendation X.712 (ISO 9596-2).

4.2.9.4. Association Setup Rules. The application service element CMISE is defined such that it is possible to combine with other service elements to form an application entity. To provide for this flexibility, the rules for exchanging information during an association setup and release are specified without creating a service that maps to ACSE services. To clarify this statement, let us assume that CMIS defined services for initializing and releasing associations. If there is a direct mapping between the CMISE initialization and termination services to ACSE association setup and release services, then it will not be possible to include other application service elements as part of the application entity definition. Suppose that a management application like software download requires request/reply interactions to activate/deactivate the download or a patch to the software. Once the initial setup is made, the actual download may be done by using, for example, a file transfer service. One approach is to set up two associations—one for the request/reply interactions using CMIP and another for the file transfer. In this case the features are negotiated for each application service element individually. On the other hand, if the application entity is to include both application service elements (CMISE and FTAM), then direct mapping between each of the service elements to ACSE is not desirable. In one association

features required by both application service elements with widely different characteristics need to be negotiated. To facilitate this capability, unlike FTAM (discussed later), CMISE did not include initialization and termination services. A set of rules is provided as to how information is exchanged in the user information parameter of ACSE so that required features are negotiated.

The association rules use the model that information supplied by the user of CMIP along with information added by CMIP (such as protocol version) are included in the user information of ACSE. How this information is passed to ACSE is left unspecified (depends on the implementation). With this approach, CMISE has no direct control of all the information in the ACSE user information parameter. If another application service element requires negotiation of features, then this is accomplished without any coordination with CMIP. Each of the service elements may negotiate the relevant information independently, and how the user information is coordinated is defined as part of the application context. This is a subtle point and provides the reason why in CMIP a separate annex was created for association rules instead of initialization and termination services as defined in FTAM. The use of ASEs with different characteristics on the same association is more a theoretical exercise to date. When this technology becomes mature and implementations become prevalent, it will be easier to understand the rationale behind the association rules.

4.2.9.5. Naming Schemes. Discussions on the service definitions and protocol specifications indicate that all requests (except create) include the parameter for identifying a specific managed object. The instance is given a name. CMIP standard specifies three choices for the name of the object. These are globally unique name, unique name relative to a known context (local name), and a nonspecific form which is a string of octets. Even though three formats are available from CMIP perspective, in any implementation of TMN applications, only the first two forms are used. The examples shown in Figures 4.4 and 4.7 illustrate the second scheme. The context is the network element and once the association is established with the network element, all references to objects are unique in that context. The globally unique name is an extension of the local name and includes in this example the name of the network element (which itself will be specified in terms of, for example, the network in which it is contained).

The nonspecific form was included to allow the object to be uniquely identified with the combination of the class and instance parameters. Historically the development of CMIP was completed prior to standardizing the information modeling principles and associated naming rules. It was considered to be a simple approach for managing small network elements. However, this did not prove to be simple and makes it almost impossible to use the multiple object selection feature. The use of this approach has been abandoned at least in TMN (possibly in all uses of CMIP for network management).

The schemes used for local and global names are explained in detail in the next chapter on *information modeling*.

4.2.9.6. Network Management Profiles. Before discussing the profiles, let us consider the difference between conformant and interoperable implementations. If an implementation passes the conformance tests this does not necessarily imply interpretability with another conforming implementation. This is because conformance is tested to the features implemented according to the protocol implementation statements from the implementor. On the other hand, if different choices were selected by two implementations, while both

5

Information Modeling Principles for TMN

In contrast to the existing environment where messages are specified for managing various aspects of resources, TMN introduces a different paradigm. Interoperability in a TMN environment is not limited to getting the bits across an interface but also includes interpreting the syntax and semantics of the management information unambiguously. The protocol described in the previous chapter lends itself naturally to the need for information models. The managed resources are modeled using a set of object-oriented principles. This chapter discusses these principles.

5.1. INTRODUCTION

The TMN cube discussed in Chapter 1 introduced the information component as one of the axis. The information component may be defined either by specifying the messages or using information models. For the interactive class of application, the need for modeling management information is established by the application level system management protocol discussed in Chapter 4.

This chapter discusses the information component in terms of the principles used for developing information models. The principles along with the protocol[1] discussed in Chapter 3 provide the foundation for developing an interoperable multisupplier network management application. Even though there are several approaches present in the literature to

[1]There are efforts underway in ITU as well as groups such as Network (Tele) Management Forum and ATM Forum to use methodologies and tools that will enable the development of information models without presupposing the protocol to be used. This is referred to as "protocol neutral" or "protocol independent" models. While this may be a noble goal, creating information models that are to be used for message exchanges between systems independent of the protocol is very difficult. It may be possible to achieve this with protocols that have very similar capabilities. Tools such as OMT and UML are rapidly gaining acceptance for developing models without being concerned with syntactic structures.

develop information models, this chapter focuses on using object-oriented principles. This methodology was first introduced in software design and development. The advantages were software reuse, flexibility, and scalability. These principles have been adapted to object-oriented design, analysis, and building distributed applications such as network management and directory.

Section 5.2 discusses the need for information models to provide interoperable interfaces between the managed and managing systems. A definition of what a management information model represents and a simple example are provided in Section 5.3. The ingredients that make the object-oriented paradigm suitable for developing network management information models are discussed in section 5.4. The various components associated with the structure of management information are presented in Section 5.5. Sections 5.6 through 5.7 define the structures for defining management information. Because the approach provides for extensible specifications, Section 5.8 discusses how to use the concept of inheritance to extend existing specifications. In order to support interoperability between managing and managed systems when compatible definitions are implemented by different systems, the concept of allomorphism has been introduced. This is discussed in Section 5.9. The management information models define, in addition to the schema for the properties of the managed resource, rules for identification. Section 5.10 discusses these principles. Use of object-oriented principles are augmented in Section 5.11 with additional concepts to model relationships between the resources. Because the protocol and information models were developed with interoperability requirements between open systems, many aspects of a schema are given globally unique identification. Section 5.12 discusses registration of management information for this purpose. Three different trees are discussed in this chapter associated with developing a schema. Clarification of the three tree structures is provided in Section 5.13. The different concepts discussed as part of an information model are related to the components of the protocol in Section 5.14, and a summary of the chapter is included in Section 5.15.

5.2. RATIONALE FOR INFORMATION MODELING

Information modeling was initially introduced in database application. Schema was defined in terms of relationships between the components of a system. *Fundamental Concepts of Information Modeling* by M. Flavin discusses the task of information modeling as a top-down design procedure where the initial step is to start with a high level design. Details are added as the problem is decomposed, and this process continues until the data elements and the corresponding data structures are defined.

Designing system engineering specifications when developing a system simple or complex has the advantage of performing analysis prior to incurring costs associated with actual development. Many of the applications emerging with information superhighway and data warehouses are data intensive in terms of the volume and complexity. The system engineering specification result of adequately modeling the interactions between the various components of the application offers a discipline that enables a common understanding, between the user and the developer of the application.

Initial efforts on information modeling addressed the design of business systems. Often an information model design is intuitive and applies heuristic. As such, often information modeling is considered an art. Different books are available in literature on information

modeling that provide the rules and guidelines to impose a structure to a problem domain. These procedures aid in adding discipline and enhancing the accuracy of the models. However, they do not eliminate the subjective aspects associated with information modeling.

A major goal of the TMN architecture is interoperability. However, information modeling is not specific to network management. The requirement for successful communication between two application entities is to have a common understanding of the information exchanged irrespective of the application. Even though the details and methodology used may differ, there are many application standards or public domain documents with information models. Examples are Directory, message handling system, open distributed processing using the architecture developed by Object Management Group, Internet management, and database management.

Information modeling approaches vary widely. The entity relationship models were used to define the business entities and relationships between them subject to policy constraints. This approach popularly known as entity-relationship (E-R) modeling has been widely used ever since its introduction by P. P. Chen (extended further by E. F. Codd) in system analysis and database design for over 25 years. As such many of the performance issues that are associated with the recent object-oriented designs have been solved by several suppliers. Despite criticisms, the greatest strength of this approach is the simplicity. Even though within TMN an object-oriented approach is chosen, the relationships between objects are difficult to understand without learning the details of the model. The notation used to represent the model (discussed in the next chapter) facilitates machine processing rather than providing the user with an overview. Therefore to present a user-friendly representation of the entities and the relationships, often E-R-like diagrams are used.

5.3. WHAT IS A MANAGEMENT INFORMATION MODEL?

A management information model defines the schema for the information visible across an interface between the managing and managed systems. The phrase "information model" has been used sometimes to identify the model of a specific function for a specific technology or a group of functions as applied to one or more resources. In other words the boundary of an information model varies widely. In some cases for readability grouping called *fragments* is defined within a model.

Consider the following examples to illustrate the use of the term "information model." As part of X.700 series, several System Management functions have been defined. An information model is associated with several of the functions. For example, the logging functions define a model to manage a log and the records contained in them. In contrast an information model defined in G.774 addresses managing SDH network elements for configuration management and some fault management functions. The Generic Network Information Model in Recommendation M.3100 defines a model in terms of fragments of functionality—cross connection fragment, termination point fragment, equipment fragment, etc. These examples illustrate the fact that when the phrase "information model" is used, what it constitutes depends on the context. This is not to be treated as a major problem or flaw. The goal of interoperability is not compromised because an information model may represent one single function or a resource versus a collection of functions. What are important are well-defined concepts, rules that can be applied so that semantics and syntax of the management information are unambiguously understood by the communicating entities.

An information model in the context of this book is an abstraction of the resources and their properties for the purpose of management. A resource exists, for example, as part of telecommunications network in order to provide a service. A management information model addresses only the information that is relevant within the management context. The modeled resources may be physical (a line card, video module) or logical (log, cross connection map). Examples such as line card, video module, and cross connection maps are present to offer services and are to be managed (provisioning, reporting alarms, etc.). Resources such as log are used to aid in management. For example, logging alarms in a log facilitates the management system to retrieve history information. The log itself is not used for providing a telecommunication service but a management service. The two types of management abstractions are sometimes distinguished by using the terms *managed object* and *support managed object*. However, from the perspective of the management information model and network management in general they are both managed entities.

An information model may therefore be considered as a representation for the collection of resource(s) and function(s) to meet some requirement(s) for an application, the specific case here being network management. Let us take an example of a channel unit (CU) that supports a POTS interface. The properties of the channel unit that are monitored or controlled, representation of this information for exchange with a managing system, and a value for a specific CU are shown in Table 5.1. A channel unit is identified by the value of channel unit Id in order to distinguish one from another in the network element. The CU is developed by a supplier as indicated by vendor name. The channel unit can be used to offer services to four subscribers. The number of available ports in the channel unit indicate how many subscribers are assigned and how many are available for future assignments. A channel unit may support different services (POTS, ISDN, E1, DS1, etc.). The type of channel unit contains this information. The channel unit operating or not is determined from the value of working status. The replaceable property indicates the channel unit is to be changed as a unit instead of its components. The alarm status is used to denote if there are any outstanding alarms on the channel unit. The version of the channel unit and the serial number are also available for management purpose. The properties shown in the table are those present during the existence of the channel unit in the network.

If a channel unit has a failure, the working status combined with alarm status may be used to determine the corrective action. This will require the managed system to poll the

TABLE 5.1 Managed Properties for Channel Units

Property	Representation	Example Value
Channel unit Id	Integer	0
Vendor name	Printable string	"LGR"
Type	Printable string	"POTS and DATA"
Number of ports	Integer	4
Assigned ports	Integer	2
Working status	Boolean (Yes/No)	Yes
Replaceable	Boolean (Yes/No)	Yes
Alarm status	Critical(0), major(1), minor(2), clear(3)	3
Version	Printable string	"Version 1.5"
Serial number	Printable string	"LGR-PART-1058"

channel unit periodically to determine whether the subscribers are being provided the service. A proactive approach will be for the channel unit to signal the transition from working to not working and inform that a critical alarm has occurred. Emitting a notification of a critical alarm is also considered to be part of the information model for the channel unit. The channel unit may be tested by performing a loop back test where a specific stream of data is sent and if this stream is received with no errors then the channel unit is considered to be functioning correctly. This type of control is also considered to be a manageable aspect of the channel unit.

The first two columns may be considered to specify a template to represent the management abstraction of a channel unit. These properties, alarm report and loop back test, can be applied to every channel unit regardless of how it is implemented and the network element in which it is present. A specific instance of a channel unit according to the template assumes different values. Properties such as version and serial number are invariant in that they do not change as long as that specific channel unit is in existence. The alarm status, for example, is a case where the value may change during the existence of the channel unit. The information model definition includes the behaviour associated with these properties and whether these are allowed to be modified by the managing system.

The power of information modeling is to bring together the characteristics of the resource providing a specific function regardless of the actual implementation of the resource. A collection of instances represented in accordance with the templates for the corresponding resource types forms a repository. This repository in the case of management is called *Management Information Base* (*MIB*). Information models have been used in other applications even though details differ based on the application. A repository of instances in the case of directory application, for example, is known as *Directory Information Base* (*DIB*).

Development of complex distributed systems necessitates system engineering requirements that include well-specified interfaces. The power of information modeling makes it an essential component of this up-front engineering effort and can potentially reduce development costs.

Having discussed what an information model is for management application, different approaches may be used to define the models. The approach discussed here uses object-oriented modeling techniques. The object-oriented methodology is being used in peer-to-peer interface definitions as well as in distributed processing environment. Contrasted with this approach is the "message"-based paradigm where an information model may or may not be explicitly defined. Management of network elements by operation systems (management systems) using this approach is prevalent in several telecommunications network. The messages are specified using the *Man–Machine Language* (*MML*) standard Z.300 from ITU or a derivative of this language.

5.4. OBJECT-ORIENTED MODELING PARADIGM

Even though the message-based approach is simple and has been widely deployed, it lacks rigor in specification. The messages define only what is exchanged across an interface. Taking the case of channel unit described in the previous section, an alarm report message will indicate that the alarm has occurred for the channel unit. There is no easy way to define the conditions for generating the alarm. The type of operations that can be performed by a management system is inferred from the messages. For example, a write operation

not permitted on vendor name is inferred from the lack of a message. The object-oriented approach, concentrates on all the properties and allowed operations from the perspective of the resource instead of what is transferred across an interface.

The object-oriented methods have been accepted since mid–1970 for developing information models in many areas such as distributed processing, software development, telecommunications and data communications management, directory services, and message handling systems. The actual details, complexity, and representation techniques may differ even though many of the fundamental principles remain the same. Today in the industry three O-O methods are prevalent for object-oriented analysis and design. These are Booch 93, Object Modeling Technique (OMT) from Rumbaugh, and Object Oriented Software Engineering (OOSE) from Jacobson. There are tools available for these techniques, and one method is more suitable than the other depending on the application. The OOSE method defines use cases to describe business level requirements and analysis. These use cases have been adopted in defining the business level requirements within Network (Tele) Management Forum. The OMT approach is suitable for data intensive applications. Because the management information models are data intensive, a mapping is provided by NMF (renamed to be TMF) between the representation technique used in network management standards and OMT. Booch '93 approach is found to be more suited for the design phase and has been applied in engineering intensive applications.

Even though this chapter is focused on the principles used in developing information models in TMN, the need for industry accepted tools based methodology during the requirements and analysis phases has been recognized recently within TMN standards and NMF. In addition, efforts in groups developing distributed processing application standards and communications infrastructure are also converging on a methodology that combines the strengths of the three techniques. This is called *Unified Modeling Language* (*UML*) and is being adopted by *Object Management Group* (*OMG*). Tool support is also expected to be available in a 1997–1998 time frame for this method. The adoption of UML within TMN is being discussed, and there is some level of support given the wide acceptance of this method in several other groups.

Because the aim of this chapter is to describe information modeling principles as currently adopted in TMN specifications, further discussion of these techniques is not presented here. However, where appropriate, an example of using OMT to describe interactions between objects is provided. The fundamental concepts used in developing object-oriented models are encapsulation, modularity, extensibility, and reuse. These are discussed below.

5.4.1. Encapsulation

The O-O methodology, unlike the structured methods provides an abstraction that combines data and functions that use the data and operate on them into an object. Let us consider the example of channel unit presented above. The management functions such as retrieving remote inventory, reporting alarms use and possibly modify the data (serial number and alarm status, for example). The channel unit represented as an object encapsulates all the relevant properties visible to the management interface. Irrelevant details such as the signaling pattern not relevant to management are suppressed, thus providing a management abstraction.

A second aspect of encapsulation is associated with how the object is responsible for maintaining the integrity of the encapsulated data when it receives requests (the message received by an object) to perform an operation. The object controls how the operation is performed by enforcing applicable consistency constraints. Continuing the channel unit ex-

ample, let us assume that there is a failure and the working status changes to not working and the alarm status indicates there is a critical alarm. A request to change the working status to yes will be denied because the consistency constraint requires the alarm status must be clear or minor. Suppose a channel unit is capable of supporting both POTS and data services. If there are two ports available and the data service requests a rate of 128 Kbps then the request to assign the two ports will be accepted. However if the request is for 256 Kbps this request will be denied. (Note to support this example a closer coupling between the assigned port and the service type is required than shown in the example).

5.4.2. Modularity

The O-O methodology naturally lends itself to a modular specification. In defining objects, one encounters the issue "what constitutes an object." This is often subjective and depends on how modular the specification should be thus providing for flexibility and future extensions. This can be demonstrated with the example of traffic management measurements. Assume that an access network based on HFC architecture is used to provide POTS services to customers. In an HFC access network two types of concentration points exist. The call processing protocol such as V5.2 provides the first level of concentration where the number of lines between the access node and the local exchange is less than the number of subscribers connected to the access node. The second level of concentration is in the use of the RF spectrum. A performance measure is the number of blocked calls due to unavailability of the channel because of congestion. The blocking may be split into two parameters for originating and terminating calls. In order to understand the level of congestion another measure required is the number of call attempts (again can be separated as originating and terminating if necessary). Calls may also be blocked because all the lines connected to switch are in use.

For simplicity let us only consider blocking as a result of unavailable RF channels. The measures identified above are applicable in this case. Let us assume that an object called equipment is defined for modeling the network interface device where the concentration takes place. The traffic measurements may be modeled as property of the equipment by encapsulating within it the traffic parameters. This approach offers a simple model where one object has all the information for the functions identified at a given time. However, this is not a modular specification and does not apply the advantage of the O-O methodology effectively. The disadvantage with the simple model is it makes it difficult to use the equipment terminating at the network interface for another application, for example, to determine the usage on the subscriber line. Another disadvantage is if the traffic measurements are to be discontinued for a period of time. In this case, disabling only that function when it is integrated with all other capabilities of the object is difficult and cumbersome. A more modular specification is to collect the traffic measurement parameters together as a separate object and define a mechanism to relate to the corresponding equipment objects. This level of modularity translates to flexibility in managing the device. The managing system may turn on or off the measurements, modify the parameters to be measured by deleting an existing one and creating a new one without affecting the object representing the network interface unit.

Another example is provisioning a customer's subscription data for various services. Modeling a subscriber line with parameters such as call forwarding, call waiting, etc., does not provide a modular specification. Instead modeling the parameters of a service as a separate object and linking to the subscriber offers the necessary flexibility for enhancing the parameters without impacting the subscriber line object definition.

Modular specification may also be translated to flexibility in software development. However, if taken to the extreme, there is additional cost because the number of objects and relationships to be maintained increases. Integrity constraints must be verified across multiple objects, and this may increase processing time.

5.4.3. Extensibility and Reusability

The requirement for augmenting a given definition based on new technologies and services has been well recognized in every industry. In the message paradigm, to manage a resource developed for a new technology, usually a new message is created. The O-O methodology takes a different approach. Using a technique known as *inheritance*, discussed later, a given specification is extended to add new capabilities not included in the current technology.

Consider the case of performance monitoring. Parameters applicable in PDH technology are code violation (a count of parity errors); errored second (a one second period with one or more parity error occurrence); severely errored seconds (a one second period with greater than *n* parity error occurrences); and unavailable second (count of 1 second intervals for which the path is not available). The network elements are providing for new parameters as they include more measurement capabilities. Let us suppose that new parameter called AIS second (count of 1 second interval containing one or more alarm indication signal defect) is to be measured. The object definition can be easily extended to include this additional parameter.

Extending a specification implies building on existing specifications. The original definition of the object is reused and extended to include new parameters (properties in general). Reusability at the specification level can also be extended to software development of the objects.

5.4.4. Relationships

In addition to the three capabilities mentioned above, expressing relationships between the objects is an important aspect. In a purely object centric approach, relationship is not explicitly considered. As we will see later, the basic principles have been augmented with the strength of E-R approach, namely modeling the relationship. The cardinality of the relation between two object classes may vary. An example of the multiple types of associations that may exist between different objects is illustrated in Figure 5.1 using OMT technique. The type of the object is referred to as a *class*. For example, a channel unit described earlier is a class and let us assume that a subscriber served from a channel unit is modeled by a line termination object. The association between these two object classes may be one to many because a channel unit can support, for example, four subscribers.

Use of object-oriented methodology offers some of the key ingredients required for information modeling in a continuously evolving network with new technologies and services. However, the same advantages do result in increased cost depending on the model. A balanced approach in determining the objects without causing undue expectations in terms of available memory and processor speed[2] is required.

[2]This is a major concern when developing products for time-critical applications. The management functions in a network element should not impact the call processing functions with a very stringent time budget.

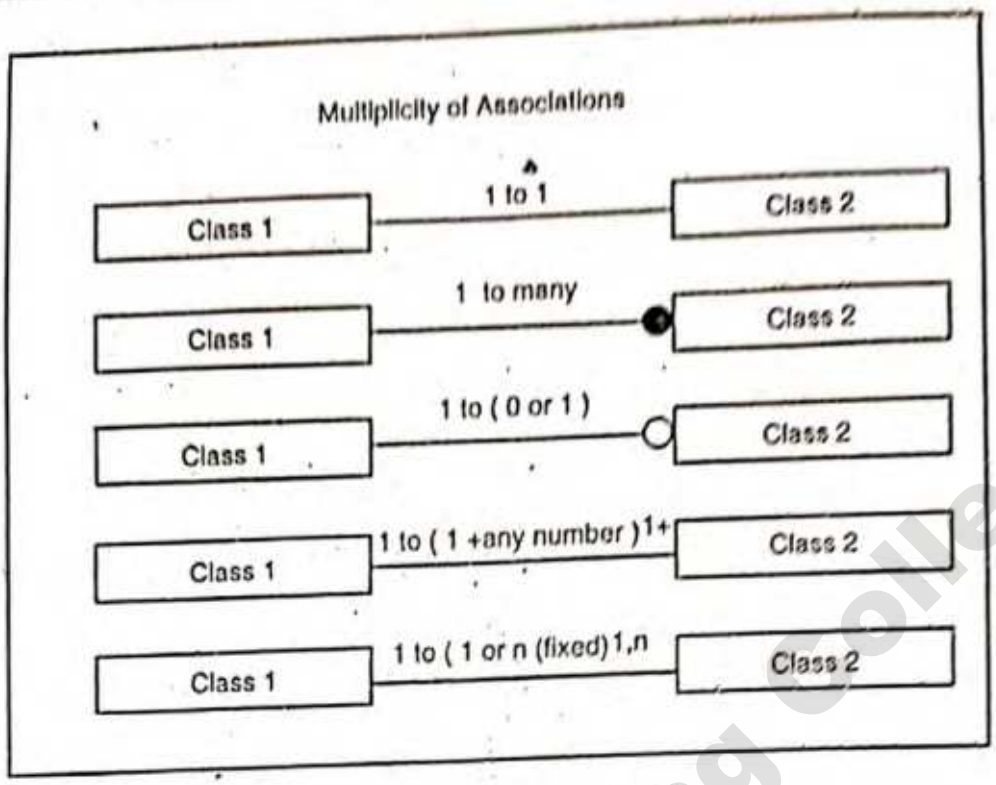


Figure 5.1 Relationships between objects.

Taking advantage of the capabilities offered by the O-O methodology, the next sections describe in detail the concepts used in developing information models in TMN. Examples are provided to explain these concepts.

5.5. STRUCTURE OF MANAGEMENT INFORMATION

The term *Structure of Management Information (SMI)* is used in both OSI management-based applications (TMN environment) as well as in Internet network management. Even though there are differences in modeling principles, the fundamental goal remains the same in both cases—the concepts and techniques used to represent how management information is structured for developing an interoperable interface.

Several variations of object-oriented design principles are available in the industry. As pointed out earlier tool support exist in some cases. The information models defined for use with CMISE, though object-oriented, adhere to the principles outlined in SMI series of recommendations instead of other approaches available in literature. As part of OSI Systems Management, a series of documents (Recommendation X.720 series | ISO/IEC 10165 Parts 1–7) have been developed to define the structure of management information. Table 5.2 lists the documents available in this series and a brief description.

The minor extensions and corrections published as amendments and corrigenda are not listed in the above table. The modeling principles and the notational technique form the basis for the information models that are available within the set of TMN recommendations.

The following sections describe the principles and concepts in detail with the help of examples. The notational technique for representing the information models is discussed in the next chapter. Even though the principles are well-structured, modeling is still an art.

ISO/IEC : International organization for standardization /International Electrotechnical commision.

TABLE 5.2 Structure of Management Information Standards

Recommendation/Standard	Title	Description
X.720 | ISO/IEC 10165–1	Management information model	Defines the object-oriented principles and features used to specify a management information model.
X.721 | ISO/IEC 10165–2	Definition of management information	Specifies the Information models containing management information for generic and some specific functions (Recommendations X.730–736).
X.722 | ISO/IEC 10165–4	Guidelines for the definition of managed objects	Defines the notations for expressing the management information using a semiformal approach.
X.723 | ISO/IEC 10165–5	Generic management information	Specifies at a generic level the management information pertaining to communication entities such as connection mode protocol machine.
X.724 | ISO/IEC 10165–6	Requirements and guidelines for ICS Proforma associated with management information.	Defines a tabular notation for expressing conformance of an implementation to the management information contained in the models.
X.725 | ISO/IEC 10165–7	General relationship model	Extends the O-O principles to include the model of the relationship.

How to group the various properties into one object versus multiple ones is dependent on a number of factors, some of them even competing at times.

The following sections will illustrate that there is more than one way to model the functions of a resource for management. Considerations such as optimizing data transfer for external communication, ease of retrieval and manipulation of data, granularity level for the information, and possible implications for implementations (memory size, processor power, speed of processing) will drive decision between multiple choices.

5.6. MANAGED OBJECT CLASS DEFINITION

The central concept as mentioned earlier is the management view of a resource and is modeled as a managed object. A resource may be physical (e.g., equipment holder, circuit pack, video tap) or logical (e.g., cross connection, customer line, call forwarding features). Taking the example of the channel unit mentioned earlier, different instances of channel units may exist from multiple suppliers. The schema of a channel unit applicable to multiple instances is defined as a *managed object class.*

The next subsections describe the distinction between the schema definition and an instantiation of the schema.

5.6.1. Class versus Instance

A managed object class is defined in terms of the characteristics that are present in multiple instances of that class. The properties are defined using a substructure called *packages* discussed later. Leaving aside for simplicity this substructuring mechanism, an object class is determined by the operations it supports, notifications emitted, attributes, and behaviour. Figure 5.2 depicts the possible components of a managed object class.[3]

[3]The terms *managed object class* and *object class* are used interchangeably.

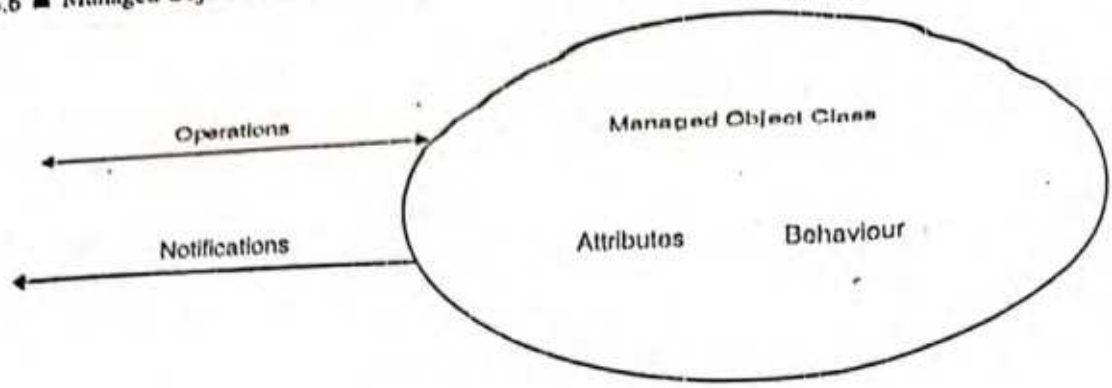


Figure 5.2 Managed object class.

This simple definition does not address variations that may exist in different instances of the same class because of options offered by one implementation of the resource versus another. Figure 5.2 shows that the object class is characterized by behaviour of the management aspects of the resource, attributes that assume different values during the existence of the object. An example is a state attribute that indicates if the resource is working or not. Notifications such as an equipment alarm to indicate the failure of the resource or a state change are used to describe events emitted by the resource. Requested operations may include a request to perform a test and returning a response after the test is executed by the resource. It is not required that every managed object class definition must contain all the properties.

Taking the example of channel unit, it can be represented by a managed object class called *circuit pack*. This object class defined in Recommendation M.3100 represents different types of cards in a system. Different instances of a circuit pack may represent not only different channel units, but also other types of cards—processor, line interface, power unit, cards that performs cross connection, etc.

All managed objects are instances of an object class by applying values to the properties defined by the template. Figure 5.3 shows the distinction between the class and instances for circuit pack. Either the phrase "managed object instance" or "managed object" may be used to identify an instance of a managed object class.

Figure 5.3 shows the template for circuit pack with a set of characteristics, referred to as *attributes*. The behaviour definition is not shown in the figure. The template also includes a notification called *equipment alarm* and an action operation called *reset*. Two instances of this class are shown with values assigned to the properties. One managed object is representing a channel unit and a second is a E1 card. The values assigned to the attributes may or may not be the same across different instances. The attribute *equipment Id* is used to name an instance of the circuit pack class. In the example the notation (empty) is used to indicate that there are no elements in the list (it is empty).

As shown in the figure, the managed object class is a type definition or a schema to follow and managed objects are instances of a class. Different instances (without assuming any options) contain the properties identified in the template; however, they take different values depending on the resource represented. The values may vary with time. For example, a minor alarm status may get cleared in the E1U shown above. The value of the alarm status will change to reflect that there is no alarm in the corresponding resource.

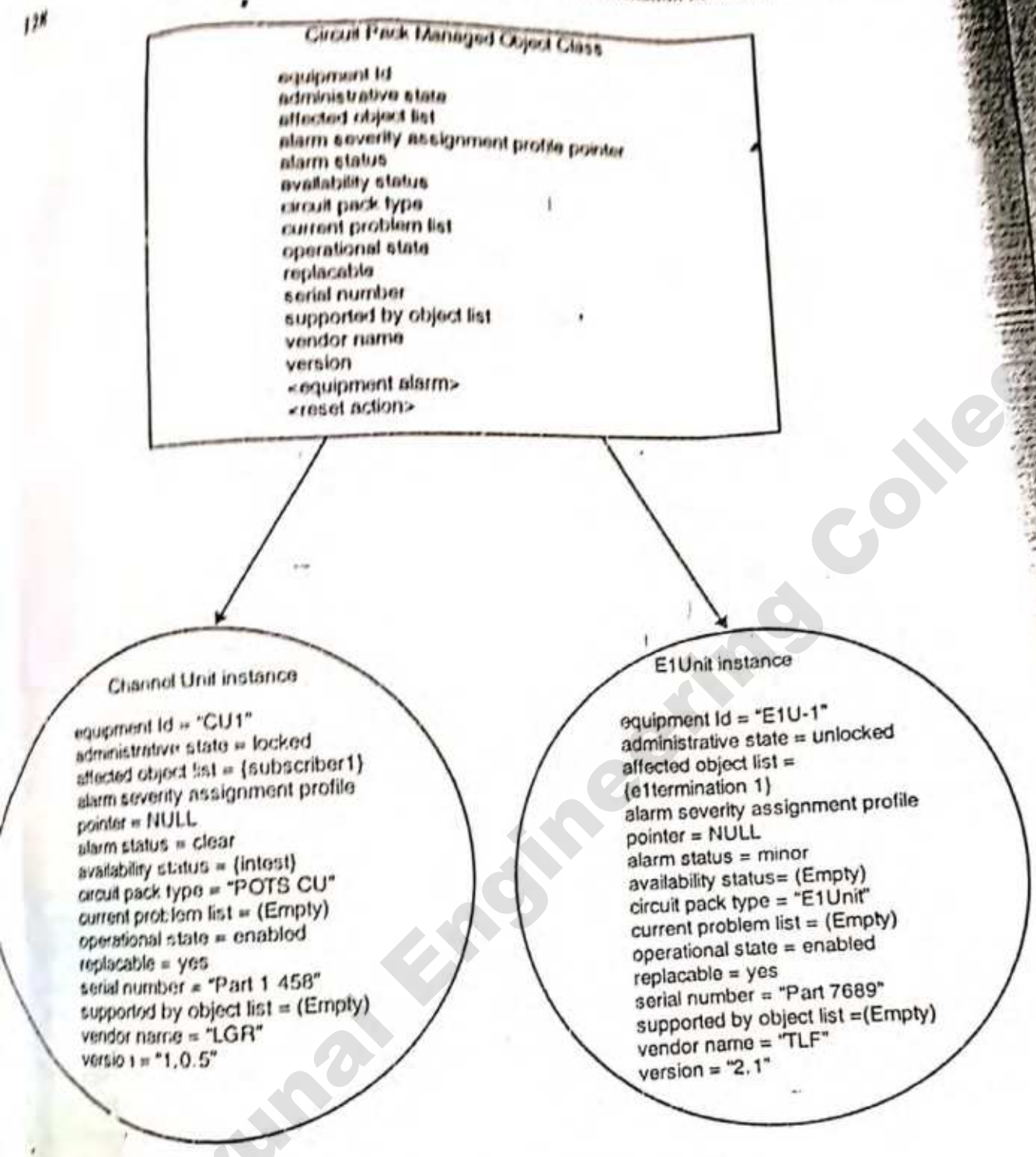


Figure 5.3 Managed object class versus instance.

5.6.2. Management Information Base

The managed object class definition is part of the information model. Managed objects are created in a system to represent the resources managed according to the managed object class definition. For example, in TMN a number of circuit pack objects may be created to represent the various cards, channel units in the system. The repository of these managed objects in a system like an NE is called *Management Information Base* (*MIB*). Even

though the instances are objects, the database used to store the MIB may vary across different implementations. For example, a relational or an object-oriented database may be used. Representation of an MIB in literature is shown using a tree structure as indicated in Figure 5.4. Note that the same structure was also shown in Chapter 4 where the management protocol was presented.

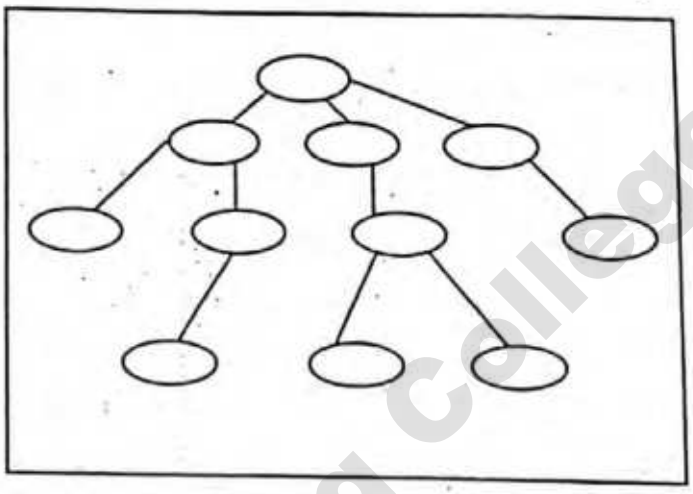


Figure 5.4 Management Information Base representation.

Even though the MIB is shown as a tree, this is not strictly correct. An MIB includes all the management information which is a collection of the managed objects and the information contained in them. The tree structure is conventionally used because it represents the managed objects arranged according to how they are referenced or named. This is discussed in detail later on naming a managed object.

The term MIB has been used in another context, managing data communications equipment using the Internet management protocol, *Simple Network Management Protocol* (*SNMP*). Modeling principles defined in this approach do not distinguish between an object class and instances of a class. Even though the management information is defined in terms of "OBJECT TYPE," they represent data items instead of a collection of properties. Both the schema and the repository of instances with values assigned to the data items are both referred to as MIB. This is possible because of the simpler modeling principles used to define the schema. The Internet RFCs containing the structure of management information (definitions of object types) are called, for example, ATM MIB, SONET MIB.

5.7. PACKAGES

A managed object class definition includes all the properties visible at the object boundary for managing a system with a simple peer-to-peer interface. A more general statement using object-oriented terminology is to state that these properties are visible at the object boundary and are accessible by another object irrespective of whether the interacting object is in the same system or in a different system. The rigorous definition of an object class in terms of the properties is extended to allow for variation among different instances. This is obtained by a construct called "package." A package is just a substructure and is a collection of characteristics (behaviour, attributes, operations, and notifications) mentioned

earlier for the managed object class. A managed object class in turn consists of one or more packages. When a package is included in an instance, all properties of the package are made available. Further selection to include or remove the characteristics within a package is not permitted.

The package concept is only an aid in specification to group all the properties that are required to be included together in an instance of a class. Another reason is to allow for optionality which is discussed below. The package boundary does not exist once a managed object is created in accordance with the schema. The properties of a package become embedded into the object and the package in which they come from is irrelevant.

Two types of packages may be included in a managed object class definitions. The additional constraints associated with a package definition as a result of the two types and the components used to define a package are discussed in detail in the following subsections.

5.7.1. Conditional and Mandatory Packages

A "mandatory package," as expected, includes properties that are always present in every instance of that class. A "conditional package," on the other hand, defines a group of properties that are included in an instance when a condition is satisfied. The condition is checked at object creation time and the properties of the package are either present or absent. Once the object is created, additional properties included in a conditional package cannot be added. This restriction is not meaningful for those in the mandatory package because these are always required and cannot be excluded.

At the time of creating the object, the condition is evaluated and if the result is a "true" value then the package must be included. The condition is phrased as an "IF" condition only. As such, while the package must be included if the condition evaluates to "true," it may or may not be present in the opposite case. The optional presence where it is a user or implementation option can be considered to be a special case of a true condition. It is not an evaluation done during the creation time, but a decision made by the implementor to support that feature, for instances, of a class.

A managed object class, as shown in Figure 5.5, is composed of zero or more mandatory and zero or more conditional packages. Each package, irrespective of mandatory or conditional, consists of behaviour, attributes, operations, and notifications. Each of these characteristics may or may not be present in a package.

Let us consider the example of the circuit pack object class used to represent different types of cards—processor, controller, channel unit, etc. Some of the circuit pack objects are resettable, and others are not. Assume also that some circuit pack objects retain the alarm information. Taking the characteristics identified earlier, a circuit pack may be defined as follows: a mandatory package that includes all the properties except reset action, current problem list, and alarm status. Because not all circuit packs can be reset by an external request from a management system, it is appropriate to create a conditional package with only the reset action. The condition for the presence may be "if the resource represented can support reset capability." The two attributes alarm status and current problem list together provide the information associated with an active alarm for the resource. A conditional package may then be specified to include the two attributes. If this package is present, both alarm status and current problem list will be included in the managed object. The condition may be "if the resource supports it."

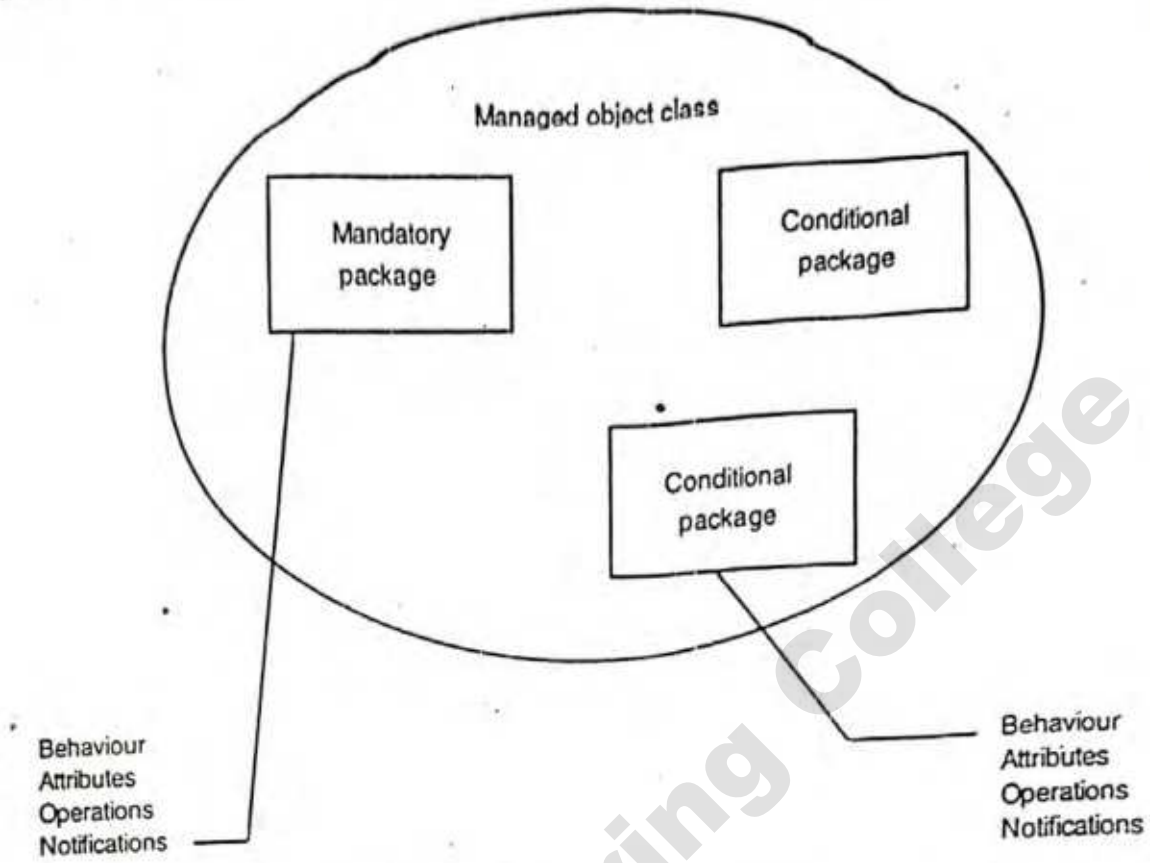


Figure 5.5 Management object class composition.

The example discussed two conditions with slightly different flavors. The reset package is present based on the resource represented. The second case is where the vendor may choose to retain the alarm information. This is a vendor option as this information is usually derived and may or may not be retained for later retrieval. In TMN models from standards, the condition "if supported" is used to indicate user or vendor choice.

Even though the conditional package may or may not be present in instances of the same managed object class, they are included only at creation time. Once the object is created with a conditional package, the properties associated with the package persist for the life of the object. It is not allowed to add the properties of a conditional package after the creation. If either deletion or creation of the packages[4] are required, then the object must be deleted and recreated.

If any of the characteristics are included in more than one package only one copy is present in the managed object, for example, if an attribute such as availability status is included in two packages (mandatory or conditional), and including both the packages in an instance does not result in two copies of that attribute. The attribute is present only once, even though the specification may give the impression that there are two copies of the same attribute. This reinforces the earlier statement that package boundaries are present only at the specification level. The impact on the allowed values for an attribute when different sets may be defined in the two packages is discussed in a later section. It is also the responsibility of the information model designer to assure that there are no contradictions when the same characteristic is included in multiple packages. For example, if an attribute cannot be

[4]The term *package* is used as a shorthand. What is meant is the associated properties as the package boundary is not identifiable once included in an object.

modified in one package, allowing modification will cause a contradiction at the managed object class level.

The components of a package definition are discussed in the following sections.

5.7.2. Behaviour

Behaviour definitions may be included at different levels—package, attribute, action, notification, and attribute group. They describe the semantics and integrity constraints associated with that level. When included in a package, behaviour definitions can describe interaction among the multiple components forming the package. Because a managed object class is described only in terms of packages, relationships between objects of different classes as well as between characteristics in different packages are included within a package. It is customary to include this information within a mandatory package. The behaviour definition at the object class level is a glue that brings together all the properties.

The use of behaviour to bring together various characteristics can be seen by taking the circuit pack object class definition. In this case, two attributes alarm status and operational state are defined, the former being in a conditional package and the latter in a mandatory package. As a result of a failure in the circuit pack, the alarm status indicates an outstanding critical alarm. The operational state should be automatically updated to be "disabled" as long as the failure (thus alarm) persists in the resource represented by the circuit pack. This description is appropriately included in a behaviour definition.

Integrity constraints may be specified using pre-, post-, and invariant conditions. Consider an example where the ratio of two attribute values is constrained to be one of a specified set of discrete values. Any change request is permitted, as long as the two attribute values obey the condition that the ratio between them is in the allowed set. This integrity constraint can therefore be defined such that when requests are received to modify the values of these two attributes, the post condition is the new ratio that must be in the allowed set.

As pointed out earlier, behaviour definitions may also describe how characteristics within a package may be influenced by the presence or absence of a conditional package. The administrative state attribute of the circuit pack object class is controllable by the managed system. Setting the value to "locked" implies the normal operation of the resource is inhibited. Let us suppose that the "reset action" may be permitted only if the resource is locked. In the example of a channel unit, by locking the circuit pack, the resource will not accept any new call requests. The behaviour definition of the mandatory package may include that when reset package is present, then the administrative state must be set to locked before performing the action. Two possible scenarios may exist—one where the managed system is required to set the administrative state prior to issuing the reset action, second where the administrative set is set to locked internal to the agent system prior to performing the reset action. In either case, the behaviour as a result of presence of a conditional package should be clarified.

The behaviour definitions discussed so far illustrate the use in describing interactions between characteristics either within a package or across packages within a managed object class definition. Another context where behaviour definitions are used is to describe the effects on a relationship among objects of the same or different classes. Let us assume that an object class "equipment holder" is defined to model the slot where a channel unit can be plugged in. The attribute holder status indicates if a circuit pack is inserted (occupied) or empty. When a channel unit (circuit pack) is inserted, the holder status must be updated to

the value "occupied." In this case, the value of an attribute is dependent on the presence or absence of another object. Behaviour definitions are used to describe such interactions.

While the behaviour definitions within a package describe effects on the characteristics considering the object as an entity, when used with other components (e.g., attribute), they describe only that component.

5.7.3. Attribute

The characteristics of the resources that are present for the lifetime of the object are modeled as attributes. They could be considered as static aspects from this perspective; however, the values themselves may be dynamic and vary to reflect changes to the resource. An attribute is defined by the combination of an identifier and value assigned to an occurrence of it in a managed object. The identifier represents the type of the attribute and has the same value irrespective of the object class in which it is included. In the example of the channel unit described earlier, the first column with names such as "vendor name" is an attribute identifier (type).

The value assumed by an attribute is determined by the syntax definition. The second column of the channel unit table specifies the representation for vendor name as a printable string. A value for this attribute such as "LGR" is assigned when the attribute is included in a managed object. The combination (attribute identifier, attribute value) which in this case ("vendor name," "LGR") is referred to as an attribute value assertion. The example has shown "vendor name" in a human-friendly representation to make it easy to understand. The actual method of representing the attribute identifier is by using a globally unique value. How to assign these unique values in defining management information model is discussed later.

An attribute specification with the syntax represents how the information is defined and how it is communicated on an interface. Other properties may also be included as part of an attribute specification. The syntax (as with CMIP discussed in the previous chapter) is represented using Abstract Syntax Notation One. Use of this notation facilitates building complex structures and lists if necessary to represent how the information is communicated between the managing and managed systems. While arbitrarily complex structures are allowed, it may be appropriate to consider modeling this complex structure as a managed object class with individual elements as attributes. Depending on the type of operations to be performed, this style of modeling may be more appropriate. However, there is the added expense for maintaining the object such as name, its type, etc.

When discussing the filtering feature of CMISE, it was pointed out that requests may be structured to check for certain criteria. The conditions to filter the selected managed objects are specified in accordance with the matching rule specified for the attribute. The matching rules are dependent on the syntax for the attribute. For an attribute like the assigned ports defined for channel unit, the matching rule can include checks for whether a given value is greater than, less than, or equal to the value of the attribute. This is known as checking for ordering. For some syntax representations, for example, Boolean, the only applicable matching rule is equality. In some cases the syntax can be a complex structure with multiple elements. While checking for ordering is not appropriate, it is possible that for some element of the structure this is a meaningful thing to do. In this case, behaviour statements are used to describe how the matching rule is applied.

An attribute specification including syntax, matching rules, and behaviour may be done without being concerned about the package (managed object class indirectly) where

it will be included. For example, a set of definitions exists for a generic state model which may be applicable to many different resources. A state called "operational state" is available to represent whether a resource is working or not. Though this definition is generic, further specification in the context of the managed object representing the resource is sometimes required. The operational state of "not working" when included in a circuit pack may imply either there is a failure or the initialization required to bring it to "working" state has not been completed. The semantics of the same attribute, when included in a logical object like a software unit may imply that the software has a fatal error.

The subsection on attribute-oriented operations describes how modification of the values of an attribute are performed. In accordance with object-oriented principles, methods are performed as a result of receiving messages.

Attributes fall into two categories: *single* and *set valued.* Even though the difference is easily explained based on the syntax used for representing the two categories, let us first consider the semantic differences between the two cases. A single value attribute is managed as a single data item with respect to modification. Though not intuitive, an attribute is considered to be single valued either if one value is associated with it or an ordered collection of values. The set valued attribute is a collection of values that is not ordered. The major distinction is seen when modifying the values of the attributes. In the single valued case, the value may only be replaced. Therefore, even if an ordered collection such as a sequence of increasing integers consists of more than one value, a generic modify operation will replace one ordered collection with another. As a side note, in order to insert or remove within an ordered list a special mechanism has been defined in ITU Recommendation Q.824, Customer Administration for ISDN. Viewing from the syntax representation, a set value attribute is defined using "set of" construct. The set is considered as a mathematical set in that multiple copies of the same value are not present (see later the impact of modification operations).

The values assumed by an attribute are determined by the syntax. For example, if an attribute syntax for number of ports is an integer, any value for the integer is valid. However, in the context of the managed object class definition, further restrictions may be imposed on the specific values or range of values. When the number of ports attribute is defined for a circuit pack, any integer value is not meaningful. A range from 1 to 10 may be specified. Another example is the availability status which qualifies why the resource is in a not working state. A specific set of integers is assigned for cause values such as not installed, in test, off line, failed even though syntax representation is an integer. Two types of restrictions may be specified on the values, and these are not mutually exclusive. The set of permitted values specifies all possible values the attribute may assume for that managed object (package). The required values specify the minimum requirement for support when the attribute is present in an implementation of the managed object. Let us consider the example of the availability status. The list of values provided above may be considered as the set of permitted values. In addition, the specification for circuit pack may specify the values that must be supported for all circuit pack objects are not installed and failed. Irrespective of whether a circuit pack implementation chooses to support all the permitted values or not, the two required values that must always be supported are not installed and failed. The "required value" does not imply these values are always the values assumed by the availability status. The circuit pack must be capable of assuming these values when appropriate, whereas others in the permitted list may never be assumed by an instance. If a defined list is provided for the permitted values, no other value outside this list can be assumed for that attribute in the context of the managed object. Given the definition of permitted and re

quired values, it can be easily concluded that the set of required values must be either a subset or the same set as the permitted values.

Within a package definition, an attribute may be assigned an initial value, default value, or both. Care must be exercised when the initial value assignment is made because this is the value that the attribute assumes when the object including this attribute is created. This is distinguished from the default value which specifies the value assumed by the attribute if none is provided at creation time. It is not possible to override the initial value with a different value. In other words, the create operation will not succeed if the value provided is different from the initial value. The default value, on the other hand, can be superseded by providing a different value at creation time and the create request will not be rejected. However, in either case the value need not be included in the create request. An example where an initial value and default value are applicable is with performance monitoring counters. Both of them can be set to zero to indicate that when the performance data are created it is always initialized to zero. The default value provides for the additional capability to reset[5] the value of the counter during the existence of the object. The default value may be a specific value or a derived value. The specification defines how the value is derived—for example, computed from the values of two other attributes.

5.7.4. Attribute Group

An attribute group is a shorthand notation to reference a collection of attributes. A package may include zero or more attribute groups. There are two ways an attribute group may be used. When retrieving the values of the attributes, instead of individually identifying each attribute, a reference to the group is made in the request. The reference is expanded (like a macro expansion) into the components and values corresponding to the individual attributes are retrieved. If the component attributes of the group are defined with default values, a set to default operation on the attribute group may be requested. As with read, the components of the group are expanded dynamically and the current values of each attribute is replaced with its default value. The reference to the group is specified by assigning an identifier that is globally unique similar to the attribute itself. In contrast to an attribute, the group identifier does not specify a syntax and as such there is no value associated with the group. The reason is self-evident because the group reference is only a handle to a collection.

Two types of attribute groups are possible: *fixed* and *extensible*. The fixed group is formed by an explicitly defined set of attributes. The members of the group are not modifiable once the group is assigned a unique value. In order to include new attributes or remove existing attributes, a new group definition will be required. An attribute group with fixed attributes may be included in a package only if the corresponding attributes are also part of the package definition. An example where a fixed group may be used is in performance monitoring. Assume that a PM package is defined for an object that represents the termination of the signal path. The path termination, depending on the technology (SDH, PDH, ATM) includes different performance parameters. Examples are coding violation, severely errored seconds, unavailable seconds. These performance parameters are modeled as counter attributes with integer syntax. In addition, as the counters are resettable both at creation and during collection period, a default value of zero is associated with these counters. A fixed attribute group may then be defined for the set of PM parameters and included

[5]The chapter on CMISE discussed how to use the set operation to reset the value to a default value.

in the package with the parameters themselves as attributes. This permits retrieving and setting to zero all the parameters by using the reference to the group. The group definition optimizes the data traffic in the request.

The extensible group, as is to be expected, refers to a collection of zero or more attributes where new members may be added after the group is defined. As with the fixed group, if the definition included some members they cannot be removed. The extensible attribute groups are useful so that the definition can be reused when a new object class is defined with additional attributes that have the similar characteristics to the ones in the initial collection. The new attributes added must be present either in a conditional package where the attribute group is included or in a mandatory package.

A simple but possibly an extreme example of an extensible group is a group definition with a description of the behaviour and zero attributes. Such a group is defined so that attributes with appropriate characteristics that belong to the class (or in a conditional package) may be grouped together dynamically. The components of the group may vary from one instance to another depending on the attributes associated with the object. Let us consider an attribute group called *state* defined in ITU Recommendation X.731. This group is defined as extensible with no component attributes. The behaviour of the group specifies that the components of the group represent the state information associated with an object containing this attribute group. Assume that an object class definition for equipment contains the state/status[6] attributes: operational state, usage state, procedural status, and alarm status. A class definition for a resource representing a log contains administrative state, and availability status. If the state attribute group is included in both equipment and log definitions, the result of a request to read the group called "state" will vary with the object. An instance of the equipment object returns three attributes and two different ones are returned from log. The elements may vary not only between instances of different classes but also between instances of the same class. This is because a conditional package may include state/status attributes. Suppose the availability status is included in a conditional package of log. Whether an instance has this conditional package or not will determine the elements of the group.

5.7.5. Operations

An object-oriented design methodology defines interactions with the object in terms of the operations issued at the object boundary. These operations directed at the object interface are further classified into object-oriented and attribute-oriented operations. Both types of operations are directed at the object. Because some of the operations are generic and can be applied to all objects, they have been specified so as to enable reuse. The two types are discussed here.

5.7.5.1. Attribute-Oriented Operations. The discussions on attribute definition earlier pointed out how to define the characteristics of an attribute irrespective of the package(s) that contain it. These properties describe, for example, the syntax for transferring it on a communication interface and the type of matching criteria to be applied.

[6]The status attributes are defined as providing further qualification of the basic states such as is the resource working or failed. The failed state may be further augmented by a procedural states indicating initialization is required prior to the resource changing to the working state

The following operations are possible on an attribute which may be either single valued or set valued: read, replace with any appropriate value, replace with a default value, read only but may be set at creation time. In the case of set valued attributes, operations to add and remove members to the set are also included. It is also allowed to specify an attribute such that once defined for an object class it may never be allowed to be modified in any specialization.

Even though the attribute definition in a package allows various operations, it does not imply the request to perform the operation will always succeed. It is the responsibility of the object to assure that the integrity constraints defined using the pre-, post- and invariant conditions in behaviour are not violated as a result of performing the requested modification. Replacing a value of an attribute may imply more than just changing a value. An attribute called "reset" may be defined for a circuit pack with a syntax Boolean (true/false). Replacing the value to true will invoke initialization of the circuit pack and will be service affecting. Thus the request may be rejected if the circuit pack is currently in use.

Attribute-oriented operations are specified to reflect whether, in the context of a specific managed object (actually the package), the value of an attribute is read only versus modifications are permitted. It is possible that the same attribute may be allowed to be modified within one managed object class, and this operation is not permitted in another object class. Even though the above statement is true in general, there are exceptions as in the case of attributes defining the state model in ITU Recommendation X.731. The characteristics of an operational state reflects the operability (resource is working) or otherwise of the underlying resource. This attribute, by definition, has to be read only because a management system changing the value from not working to working will not correct the problem in the resource. The resource will have to be repaired or replaced in order to internally change the value of the operational state attribute.

Consider now the following example to illustrate how the same attribute may be defined with different operations. The performance monitoring (PM) parameters such as severely errored seconds are modeled as attributes. Two classes of objects are defined to contain the PM parameters. One class, called *current data* is used to contain the values corresponding to the current collection interval and another class, history data contains values collected in the previous interval (historical information). At the end of the collection period, say 15 minutes, the attribute values in current data are moved into an instance of history data. The attributes in current data will be defined as readable as well as resettable to allow zeroing them at any time during the collection interval. However, when the same PM parameters are included as attributes of history data, the only allowed operation is read. Any modification of the historical information will be disallowed. Another example relates to reports sent when attribute values are modified by a management system. The reported attribute for the resource will also become an attribute of a record object if the report is logged. Again as an attribute of the record object, the value cannot be modified whereas it was modifiable as part of the object that reported the change.

An attribute that is read only is further distinguished relative to whether the value may be set at creation time. Attributes such as the ones used to name an instance of a class are only readable once the object is created. The managing system, when requesting the creation of the object, may specify a value for the naming attribute. The naming attribute will then be specified as readable and settable at creation time.

When discussing the different types of restrictions on the values, it was noted that an attribute may have a default value specification. The default value is used, for example, when a value for that attribute is not provided in the create request. An operation associated

with the default value is "replace with default." The attribute discussed for performance parameters is an example of where the replace with default operation may be applied.

When an attribute is set valued, two other operations in addition to those mentioned earlier are possible. These are adding and removing values to the set. The set of values is treated as a mathematical set. Adding an existing value while not an error will not include a second copy. Similarly, removing a nonexisting value is not considered an error.

When an attribute-oriented operation is requested, errors may be defined to explain failure to complete the request. The errors may be generic such as the invalid value provided in the request or specific. The latter case is used to explain the reason for the failure. One example is violation of an integrity constraint if the requested change to the attribute is to be made.

Depending on the attribute and the resource being modeled, the package specification includes all the possible operations. The operations specification does not address access privileges or security constraints associated with the requesting entity. Therefore even if the attribute is specified to be replaceable, if the requesting entity does not have the access privileges to perform the modification, the request will be rejected. A model defining how to include security services is available in ITU Recommendation X.741.

5.7.5.2. Object-Oriented Operations. Requests to create or delete an object and perform an action are considered object-oriented operations. The request to create an object is not directed at the object itself as this does not exist until the object is created. In some O-O methodology there exists a concept of factory object used in creating a new instance. The three object-oriented operations are considered in detail below.

5.7.5.2.1. ACTION. Action definitions are used to model operations that a resource is capable of performing when requested by the management system. The definition is specified using an action type, information sent with the request to perform the action, response data if any, and behaviour describing constraints and conditions to perform the action (including any process descriptions such as initial setup required, etc.). The syntax of the information associated with both request and response is also required for external communication. Multiple responses may be required in some cases when performing the action. Errors may also be defined if the action is not executed successfully.

An action definition includes often a description of a process for performing the operation. As stated earlier, it should not be assumed that a process definition or having effects that are more than modifying a value implies an action definition is required. Defining an attribute to have a side effect where a process is initiated is an acceptable way of modeling a process. Similarly performing an action may result in modifying the value of an attribute. There are scenarios where action is a better modeling tool to use than an attribute-oriented operation. When requesting some operations, information may be required to be sent in the request. If the parameters in the request are not part of the characteristics of the resource (as such not required to be present as an attribute for the lifetime of the object), it will not be possible to use attribute-oriented operation. Describing multiple-object interactions, coordination of activities across objects of the same or different classes and state changes of the object are easier to describe using an action than with modification to an attribute. When multiple responses are to be included as a result of performing an operation, action definition will be required. Multiple responses may be used, for example, when performing a test to indicate the progress of the test and conclude the final response with a pass/fail indication.

The previous scenarios can be illustrated using the following examples. The cross connection between termination points is defined using an action request to a fabric in the network element. The connect action to the fabric is specified to support different flavors of identifying the two end points. The termination points participating in the cross connection may be specified directly or indirectly. In the latter case, a pool of terminations may be provided and any available termination may be selected. The action may also request a point-to-multipoint cross connection. Another example is the test model where action is used. In this case, information used for initialization of the test are supplied in the request. Other examples are bundling a read and write operation as an atomic operation or creating multiple objects with one request. The latter is the result of the restriction that the create operation instantiates only one object with one request. The modeling of customer administration information for services of ISDN will result in a performance bottleneck if individual create is used for each feature (modeled as an object) subscribed by the user. An action is defined in ITU Recommendation Q.824 where all the relevant objects are instantiated with one request.

Guidelines are defined in ITU Recommendation X.722 for using action versus set (modify an attribute). These guidelines are not prescriptive and the modeling construct used depends on the specification author or compromise between members of any standards group developing information models.

5.7.5.2.2. CREATE. The object level operation create is used to instantiate an object following the schema definition for that specific class. A create operation, unlike action and delete cannot be directed at an existing object. Even though many of the object-oriented modeling techniques include creation of the object as part of the schema for the class, the approach used here is different. The object class schema does not include create, the creation operation is defined as part of the rules for naming discussed later. The result of this separation of the create operation from the class definition is to offer flexibility in how an object is created depending on the environment, physical architecture and policies.

Assume that a network unit placed close to customer premises is modeled using equipment and equipment holder objects. Some network units serve multiple residences (subscribers), while others serve a single home. In the case of the former, depending on how many slots are populated with channel units, explicit create from the managing system (while configuring the subscribers) will be required for the equipment holder objects. The single home unit, on the other hand, auto creates the equipment holder when the network unit is configured because one and only one holder is present in this case. If the create operation is defined as part of the class, then there will be no flexibility in customizing the different instances.

Associated with the create specification is the identification of the attribute used for naming an instance of the class. This attribute itself is specified as part of the class definition even though the explicit designation of the attribute as a naming attribute is specified when create operation is defined. Behaviour and specific errors may also be included as with action.

5.7.5.2.3. DELETE. The delete operation is used to remove an instance of a class from the managed system. Similar to create, this operation is also not defined as part of the class definition. Deletion of an object may result in one of the following cases: The object and all its contained objects are deleted or it is deleted only if there are no objects included in it.

Consider the network unit object mentioned above. The network unit contains equipment holders and circuit packs. If the network unit is deleted, then either the containing

equipment holder and circuit pack objects are deleted or as long as the equipment holders and circuit packs are present, the deletion request is rejected. Behaviour and error specifications may be included with the delete operation definition.

5.7.6. Notifications

The attribute-oriented operations and create and delete operations, model characteristics of a resource similar to database schema definition. In contrast to the operations using the request as the trigger, notifications are emitted by the resource due to either internal or external events.

Internal events may be generated from the resource for several reasons. Taking the example of the circuit pack representing the processor unit, failure of the resource will generate an event called *equipment alarm*.

Definition of the notification will include the conditions for generating the notification and information included when reporting the event. The result of the failure changes the operational state value to "disabled." Assume that the processor card has a backup so that once the failure occurs, an automatic protection switch to the backup card takes place. The backup status of the protecting card changes from standby to active. In addition to the equipment alarm, additional internal events are generated—a state change notification from the failed circuit pack and a second state change notification from the protecting unit. The definition of the circuit pack object class will include both the alarm notification and the state change notification. Note that even though different state/status values may change, a generic notification is defined for changes to any state variable of a resource.

Notifications as a result of external events can be best explained by considering a multiple manager environment. Assume that two operations systems are used in managing a network element. The two operation systems perform different functions. One is dedicated for maintenance purposes—receives alarms, performs diagnostics, etc. Another system, let us assume, performs provisioning functions. The provisioning OS issues create requests to the NE for a new card plugged into a slot to expand the capacity supported, and analog line terminations for new users. Assuming that these objects were created successfully, the NE responds to the provisioning OS with the names of the newly created objects along with the values for the attributes of these objects. At the same time, the NE may send creation notifications to the maintenance OS so that when faults or state changes are to be reported, the maintenance OS will recognize these new objects. Here the object creation notifications are the result of an external event which is a create request from the provisioning OS.

The package definition that contains a notification may be further augmented with behaviour statements explaining the conditions for generating it. The notification specification, similar to action, is specified by assigning a value for the type of the notification (for example, equipment alarm, object creation) and information that is sent when the notification is reported to a remote system. The information is determined by the type of the event. An *event type* of equipment alarm includes information on the severity of the alarm, probable cause, and possible diagnostics to correct the problem. A state change notification type contains the names and values (new and as an option old value) of the state and status attributes where changes have occurred. The syntax for the parameters of the event information is required in order to communicate them to an external system. To allow for extensibility and reuse of the notification across object classes, it is conventional to provide for a parameter whose syntax is customized when the definition is reused in a managed object class different from where the original definition was created.

• The occurrence of an event in a managed object does not imply that this is always reported to an external system. A mechanism discussed in Chapter 7 may be used to configure the criteria so that an event may be forwarded if the condition evaluates to true. The criteria may specify "report only event types equipment alarm and communication alarm when the severity is critical." The notification definition itself is part of the schema for the managed object class irrespective of whether an event that occurs in a specific managed object is reported to an external system or not.

This subsection described the various characteristics of a resource that may be modeled from the perspective of management. A managed object class represents the resource being managed and the properties are specified using a grouping concept called *packages*. Given the basic components that constitute a managed object class definition, the next step to consider is how the OO concepts "extendibility" and "reuse" may be applied.

5.8. INHERITANCE

The object-oriented methodology, as noted earlier, offers a technique for easy extension of a specification. By definition, extending a specification reuses the original definition. When a managed object class is defined with a set of packages (mandatory or conditional) properties appropriate for management are included. As seen in the previous section, these are modeled using attributes, notifications, and actions.

Once a definition of a managed object class exists and is possibly implemented, extensions (a specific type of modification) may be required for several reasons. Some of the reasons are the definition may be provided at a generic level irrespective of a specific technology, features in a resource that may be included by a supplier for value-added product, new service parameters, and deficiency in the original specification. The process of extending a managed object class definition with new properties is called *specialization*.

Specialization of a definition extends a managed objet class definition by adding to the properties obtained as a result of inheritance. The extended object class inherits the properties or rather includes the properties of the original class and adds new properties. The original class is called the *superclass* and the specialized class is called *subclass*. This is very similar to subtyping a type in data types. The original type represents larger collection of entities. The specialization by adding properties restricts the set of valid entities. A simple example is as follows: While a general definition of a vehicle includes those that can travel on ground, water, and air, the specialization to land vehicles restricts the applicable instances by the additional requirement that it must be used on the road.

Inheritance may be used differently even among the various object-oriented methodologies. In the context of TMN modeling strict inheritance is applied. This implies that the specialized class is allowed to only add properties. The addition may be done in different ways. The subclass may include zero or more new attributes, notifications, and actions. Because the properties are always grouped into packages, the new class may include mandatory and conditional packages. The extended class may make a conditional package mandatory. However, the reverse is not permitted. Similarly, an existing conditional package cannot be removed in the extended class.[7] When discussing attribute-oriented operations, it was pointed

[7]This implies that care must be taken when defining generic classes that are expected to be specialized for specific applications. Having conditional packages that are not useful when specializations are developed leads to nonoptimal solution.

Network Management Tools, Systems, and Engineering

OBJECTIVES

- *System utilities for management*
  - *Basic networking tools*
  - *SNMP tools*
  - *Protocol analyzer*
- *Measurement of network statistics*
  - *Traffic load*
  - *Protocol*
  - *Data*
  - *Error*
  - *Traffic monitoring using MRTG*
- *MIB engineering*
  - *Limitations of SMI and SNMP*
  - *Counters and rates*
  - *Object-oriented design*
  - *SMI tables*
  - *SMI actions*
  - *Guiding principles for MIB design*
- *Design considerations of NMS*
  - *System and user requirements*
  - *Local and remote management*
  - *Functional requirements*
  - *Architectural aspects*
  - *Key design considerations*
  - *Functional modules and managers*
  - *Root cause analysis*
  - *Distributed network management*
  - *Server and client platforms*
- *Network management systems*
  - *Display of functional views of network management*
  - *Examples of commercial and open source NMSs*
  - *System and applications management*
  - *Enterprise management system*
  - *Telecommunications management system*

Thus far we have covered SNMP standards for IP network management. This includes protocols and Management Information Bases (MIBs). In this chapter we will discuss assorted tools and techniques that can be used for the management of networks using SNMP (and other management protocols).

In Section 9.1 we start with commonly available utilities that can be used for management. This is followed by a discussion of tools for gathering network statistics in Section 9.2. Next, in Section 9.3 we examine the design of MIBs (MIB Engineering), which is important for any vendor of networking equipment. In Section 9.4 we turn to the design of a typical network management system (NMS) server for a large telecom network. In Section 9.5 we outline several commercial and free NMS products for different types of networks.

9.1 SYSTEM UTILITIES FOR MANAGEMENT

A significant amount of network management can be done using operating system (OS) utilities and some freely downloadable SNMP tools. These can be put together quickly using simple scripting languages such as Perl [Cozens; Lidie and Walsh, 2002]. Some of these tools are described below.

9.1.1 Basic Tools

Numerous basic tools are either a part of the OS or are available as add-on applications that aid in obtaining network parameters or in the diagnosis of network problems. We will describe some of the more popular ones here under the three categories of status monitoring, traffic monitoring, and route monitoring.

Network Status Tools. Table 9.1 lists some of the network status-monitoring tools that are available in the Linux/UNIX and Microsoft Windows (XP and Vista) environments. We use Linux and UNIX interchangeably in this section as the same set of basic utilities and tools are available on both. This also applies to Solaris, HP-UX, AIX, MacOS X, and Free BSD.

The command *ifconfig* on a UNIX system is used to assign an address to a network interface or to configure network interface parameters. Usually, one needs to be a super-user (su) in UNIX to set interface parameters. However, it can be used without options by any user to obtain the current configuration of the local system or of a remote system. In this and other commands, you may invoke *man command-name* to obtain the manual page of a command in a UNIX system. An example of *ifconfig* is shown in Figure 9.1.

Table 9.1 Status-Monitoring Tools

NAME	OPERATING SYSTEM	DESCRIPTION
ifconfig	Linux	Obtains and configures networking interface parameters and status
ping	Linux/Windows	Checks the status of node/host
nslookup	Linux/Windows	Looks up DNS for name–IP address translation
dig	Linux	Queries DNS server (supersedes nslookup)
host	Linux	Displays information on Internet hosts/domains

```
netman: ifconfig –a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
        inet 127.0.0.1 netmask ff000000
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST>
        mtu 1500 inet 192.207.8.31 netmask ffffff00 broadcast 192.207.8.255
```

Figure 9.1 Example of Interface Configuration (*ifconfig*)

The option –a is for displaying all interfaces. As we can see, there are two interfaces, the loopback interface, lo0, and the Ethernet interface, hme0. Option –a provides information on whether the interface is up or down, what maximum transmission unit (mtu) it has, the Ethernet interface address, etc.

One of the most basic tools is *ping* (packet Internet groper). A frequent use of it is when an execution of a command on a remote station fails. As one of the first diagnostics, we want to ensure that the connection exists, for which the *ping* utility is executed to the remote host. If it fails, then there is a problem with the link. If it passes, then the trouble is with either the application or something else.

You have already used the *ping* tool in the Chapter 1 exercises. It is based on the ICMP echo_request message and is available in both UNIX and Microsoft Windows OSs. "Pinging" a remote IP address verifies that the destination node and the intermediate nodes have live connectivity and provides the round-trip delay time. By pinging multiple times, we can obtain the average time delay, as well as percentage loss of packets, which is a measure of throughput. This feature can be used to check the performance of the connection. There are numerous implementations of ping. Read the manual page for details on the specific implementation on each host.

The UNIX commands, *nslookup*, *host*, and *dig*, are useful for obtaining names and addresses on hosts and domain name servers. A domain name server provides the service of locating IP addresses. The *nslookup* tool is an interactive program for making queries to the domain name server for translating the host name to the IP address and vice versa. The command, by default, sends the query to the local domain name server, but any other server can also be specified. The command *dig* is a more powerful replacement for *nslookup*.

For example, the command *dig nimbus.tenet.res.in* on the host *beluga* yields the result shown in Figure 9.2.

An interpretation of the above result follows. The host, beluga, obtained the IP address of nimbus.tenet.res.in (203.199.255.4) from the domain's name server 203.199.255.3. This information is given in

```
[beluga]~> dig +nocomments nimbus.tenet.res.in
nimbus.tenet.res.in.  604800    IN    A     203.199.255.4
tenet.res.in.         604800    IN    NS    volcano.tenet.res.in.
tenet.res.in.         604800    IN    NS    lantana.tenet.res.in.
volcano.tenet.res.in. 604800    IN    A     203.199.255.3
;; Query time: 2 msec
;; SERVER: 203.199.255.3#53(203.199.255.3)
;; WHEN: Fri Mar  6 14:12:43 2009
;; MSG SIZE  rcvd: 149
[beluga]~>
```

Figure 9.2 Example of Network Address Translation (*dig*)

the first line of the output (the A-record). The *authority* for this information, i.e., the name server(s) for the domain, is given in subsequent NS-records. The name server that responded to this query is given in the comments at the end.

Instead of the host name, the IP address could also be used as the option parameter in the dig command. For example, the command dig +short –x 203.199.255.5 will return the information that this IP address is assigned to *lantana.tenet.res.in. dig* can give very verbose output. The options *+nocomments* and *+short* are used to suppress some of this.

dig or *nslookup* can help when one wants to find all the hosts on a local area network (LAN). This can be done using a broadcast ping on the LAN. We need to know the IP address to execute this command, which we may not know. However, if we know a host name on the LAN, we could obtain the IP address using *nslookup*.

The interactive command *host* can also be used to get the host name using the domain name server. However, with the appropriate security privilege, it can be used to get all the host names maintained by a domain name server. The *dig* and *host* utilities also provide additional data on the hosts, besides host names. Refer to the manual page for details.

Network Traffic-Monitoring Tools. Table 9.2 lists several traffic-monitoring tools. One of the tools is ping, which we have mentioned as a status-monitoring tool in Table 9.1. As we stated earlier, by repeatedly executing ping with a large repeat count (e.g., ping –c 100) and measuring how many of those were successfully received back, we can calculate the percentage of packet loss. Packet loss is a measure of throughput. The example in Figure 9.3 displays zero packet loss when five packets were transmitted and received. It also shows the round-trip packet transmission time.

Another useful tool, heavily based on ping, is called *bing* (point-to-point bandwidth ping). The *bing* utility determines the raw throughput of a link by calculating the difference in round-trip times for different packet sizes from each end of the link. For example, if we want to measure the throughput of a hop or point-to-point link between L1 and L2, we derive it from the results of the measurements of ICMP echo requests to L1 and L2. The difference between the two results yields the throughput of the link L1–L2. This method has the advantage of yielding accurate results, even though the path to the link L1–L2 from the measuring station could have a lower bandwidth than the link L1–L2 for which the measurement is

Table 9.2 Traffic-Monitoring Tools

NAME	OPERATING SYSTEM	DESCRIPTION
ping	UNIX/Windows	Used for measuring round-trip packet loss
bing	UNIX	Measures point-to-point bandwidth of a link
tcpdump	UNIX	Dumps traffic on a network
getethers	UNIX	Acquires all host addresses of an Ethernet LAN segment
iptrace	UNIX	Measures performance of gateways
ethereal, wireshark	Linux/Windows	Graphical tool to capture, to inspect, and to save Ethernet packets

```
netman: ping -s mit.edu
PING mit.edu: 56 data bytes
64 bytes from MIT.MIT.EDU (18.72.0.100): icmp_seq=0. time=42. ms
64 bytes from MIT.MIT.EDU (18.72.0.100): icmp_seq=1. time=41. ms
64 bytes from MIT.MIT.EDU (18.72.0.100): icmp_seq=2. time=41. ms
64 bytes from MIT.MIT.EDU (18.72.0.100): icmp_seq=3. time=40. ms
64 bytes from MIT.MIT.EDU (18.72.0.100): icmp_seq=4. time=40. ms

—mit.edu PING Statistics—
5 packets transmitted, 5 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 40/40/42
```

Figure 9.3 Example of Traffic Monitoring (*ping*)

made. However, there is a practical limit to it (about 30 times). In practice, this means that if you have a 64-kbps connection to the Internet, the maximum throughput of the link you can measure is 2 Mbps.

Other commands examine the packets that traverse the network to provide different outputs. The commands, *tcpdump* and *ethereal* (also known as *wireshark*), put a network interface in a promiscuous mode, in which raw data are gathered from the network without any filtering, and log the data. All of them could generate an output text file, associating each line with a packet containing information on the protocol type, length, source, and destination. Because of the security risk associated with looking at data in a promiscuous mode in these cases, the user ID is limited to super-user. An example of the output of *ethereal* is shown in Figure 9.4. Each line shows information about one packet. Several packets are Address Resolution Protocol (ARP) requests. In these cases, the source MAC address is shown. For ease of reading, the vendor ID of the MAC address is shown in text form, followed by the 24-bit serial number. The three lines in gray scale in the middle of the window show packets belonging to a single HTTP session over a transmission control protocol (TCP) connection. They capture the three-way handshake used by TCP to establish the connection. The first of these packets is a connection request from 10.6.21.59 to the HTTP port on server 10.94.3.215. The server responds with an SYN+ACK, and finally, the initiator ACKs to complete the connection establishment handshake.

For an example of *tcpdump* command output, please see the SNMP get-request and get-response PDUs shown in Figure 5.17 that were obtained using the *tcpdump* tool.

The command *getethers* discovers all host/Ethernet address pairs on the LAN segment (a.b.c.1 to a.b.c.254). It generates an ICMP echo_request message similar to ping using an IP socket. The replies are compared with the ARP table to determine the Ethernet address of each responding system.

The *iptrace* tool uses the NETMON program in the UNIX kernel and produces three types of outputs: IP traffic, host traffic matrix output, and abbreviated sampling of a pre-defined number of packets.

Network Routing Tools. Table 9.3 lists three sets of route-monitoring tools. The *netstat* tool displays the contents of various network-related data structures in various formats, depending on the options selected. The network-related data structures that can be displayed using *netstat* include the routing table, interface statistics, network connections, masquerade connections, and multicast memberships. The example of the routing tables obtained using *netstat* is given in Figure 9.5. It shows the ports associated with various destinations. *netstat* is a useful diagnostic tool for troubleshooting. For example, the routing table information shown in Figure 9.5 informs the network operator which nodes have been

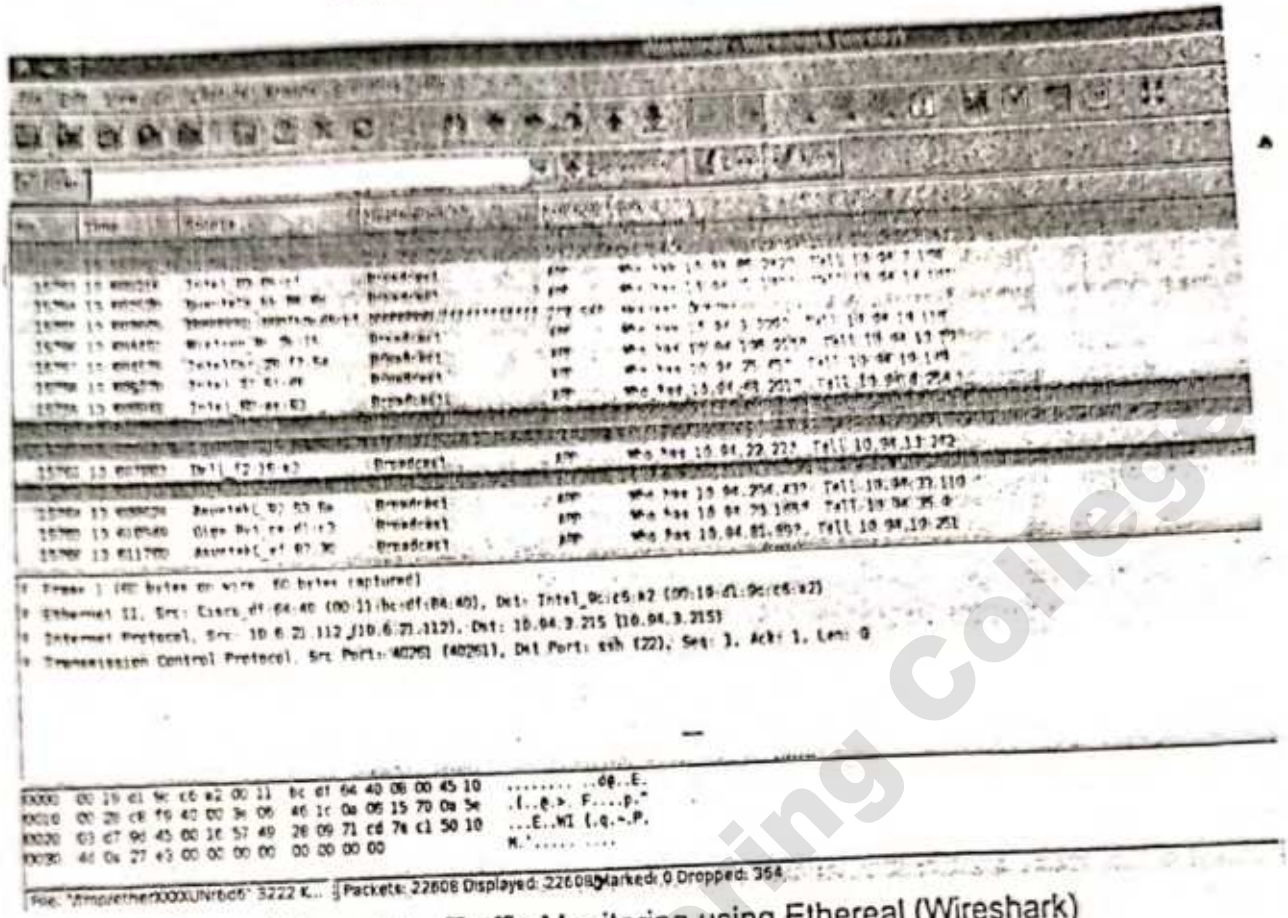


Figure 9.4 Traffic Monitoring using Ethereal (Wireshark)

Table 9.3 Route-Monitoring Tools

NAME	OPERATING SYSTEM	DESCRIPTION
netstat	UNIX	Displays the contents of various network-related data structures
arp rarp	UNIX, Windows	Displays and modifies the Internet-to-Ethernet address translation tables
traceroute tracert	UNIX/Windows	Traces the route to a destination with routing delays

active since the last purge of the table, which is typically in the order of a minute. The most frequently used options of netstat are: −r that obtains the contents of the routing tables; −i that prints the table of all networking interfaces; and −a that prints information about all active Internet connections and UNIX-domain sockets.

The *arp* tool displays and modifies the Internet-to-Ethernet address translation tables (ARP cache) used by the ARP. Some UNIX systems provide an additional tool for manipulating the contents of Ethernet-to-Internet address translation tables (RARP cache). The name of the tool is *rarp* and its use is similar to *arp*.

The third set of routing tools shown in Table 9.3 is *traceroute* (UNIX) or *tracert* (MS Windows), which is the basic tool used most extensively to diagnose routing problems. The tool discovers the route taken by packets from the source-to-destination through each hop. It is very useful in localizing th-

```
netstat -r
Routing tables

Internet:
Destination      Gateway           Flags    Refs    Use      Netif  Expire
Default          gw.litech.net     UGC      44      541550   de0
172.16.15.1      gw.litech.net     UGH      0       0        de0
ah.litech.net    0:80:48:ee:74:b4  UHLW     9       2653683  de0    202
uucp.litech.net  uucp.litech.net   UH       0       0        lo0
slp-17.litech.net  blg             UH       0       5551     ppp3
dip-244.litech.net gw.litech.net   UGH      0       2472     de0
univers-litech-gw gw.litech.net    UGH      0       47       de0
194.44.232       gw.isr.lviv.ua    Ugc      0       171831   ppp9
OSPF-ALL.MCAST.NET localhost       UH       1       86491    lo0
OSPF-DSIG.MCAST.NE localhost       UH       1       25127    lo0
```

Figure 9.5 Routing Table using netstat–r

source of route failure. It is also useful in performance/packet delay evaluation as the result gives the delay time to each node along the route. *traceroute* is based on the ICMP time_exceeded error report mechanism. When an IP packet is received by a node with a time-to-live (TTL) value of 0, an ICMP packet is sent to the source. The source sends the first packet with a TTL of zero to its destination. The first node looks at the packet and sends an ICMP packet since TTL is greater than 0. Then, the source sends the second packet with a TTL larger than the TTL needed to get to the first node, and thus *traceroute* acquires the second node. This process continues until all the nodes between the source and the destination have been determined.

Figure 9.6 gives two sample traces taken close to each other in time between the same source *mani.bellsouth.net* and destination *mani.btc.gatech.edu*. We notice significant differences in the route taken, the delay times, and the number of hops. We would expect these differences since each packet in an IP network is routed independently. Each line shows the router that the packet traversed in sequential order. The three time counts on each line indicate the round-trip delay for each router on three attempts made from the source. We notice that there is a jump in the round- trip delay from 2–5 milliseconds to greater than 10 milliseconds when the packet crosses over from the local BellSouth network. In some lines, for example in lines 9 and 13 in Sample 1, one round-trip delay reads high, which could be attributed to the router being busy. In Sample 2, the second half of the route appears congested, indicating consistent large round-trip delays. We also notice that some of the routers respond with their IP address, while others do not. The lines that are marked with asterisks are responses from those routers, which have been administratively prevented from revealing their identity in their responses.

There are also Web-based *traceroute* and *ping* utilities available in some systems. The use of these tools significantly decreases the time necessary to detect and isolate a routing problem because the final decision is based on independent data obtained from various hosts on the net.

```
Tracing route to mani.btc.gatech.edu [199.77.147.96]
over a maximum of 30 hops

  1   2 ms   3 ms   3 ms  bims008001.bims.bellsouth.net [205.152.8.1]
  2   4 ms   2 ms   3 ms  172.16.11.2
  3   5 ms   4 ms   3 ms  172.16.4.2
  4   5 ms   3 ms   3 ms  bims011033.bims.bellsouth.net [205.152.11.33]
  5   4 ms   4 ms   4 ms  205.152.13.98
  6    *      *      *    Request timed out
  7   5 ms   9 ms  12 ms  205.152.2.249
  8  33 ms  31 ms  31 ms  Hssi0-0-0.GW2.ATL1.ALTER.NET [157.130.65.229]
  9  68 ms  10 ms  11 ms  105.ATM3-0-0.XR1.ATL1.ALTER.NET [146.188.232.66]
 10  11 ms  14 ms  12 ms  195.ATM12-0-0.BR1.ATL1.ALTER.NET
[146.188.232.49]
 11  16 ms  14 ms  14 ms  atlanta1-br1.bbnplanet.net [4.0.2.141]
 12  19 ms  15 ms  17 ms  atlanta2-br2.bbnplanet.net [4.0.2.158]
 13  21 ms  56 ms 328 ms  atlanta2-cr99.bbnplanet.net [4.0.2.91]
 14  17 ms  18 ms  17 ms  192.221.26.3
 15  32 ms  20 ms  18 ms  130.207.251.3
 16  20 ms  17 ms  17 ms  mani.btc.gatech.edu [199.77.147.96]
Trace complete
```

Sample 1

```
Tracing route to mani.btc.gatech.edu [199.77.147.96]
over a maximum of 30 hops:

  1   3 ms   3 ms   4 ms  bims008001.bims.bellsouth.net [205.152.8.1]
  2   3 ms   3 ms   2 ms  172.16.11.2
  3   5 ms   4 ms   4 ms  172.16.4.2
  4   5 ms   3 ms   4 ms  bims011033.bims.bellsouth.net [205.152.11.33]
  5   7 ms   4 ms   4 ms  205.152.13.98
  6    *      *      *    Request timed out
  7   9 ms   8 ms   9 ms  205.152.2.249
  8 228 ms 214 ms 191 ms  206.80.168.9
  9 230 ms 246 ms 234 ms  maeeast.bbnplanet.net [192.41.177.2]
 10 243 ms 222 ms 212 ms  vienna1-nbr2.bbnplanet.net [4.0.1.93]
 11 230 ms 213 ms 202 ms  vienna1-nbr3.bbnplanet.net [4.0.5.46]
 12 247 ms 227 ms 236 ms  vienna1-br2.bbnplanet.net [4.0.3.149]
 13 228 ms 235 ms 238 ms  atlanta1-br1.bbnplanet.net [4.0.2.58]
 14    *    257 ms 238 ms  atlanta2-br2.bbnplanet.net [4.0.2.158]
 15 225 ms 234 ms 233 ms  atlanta2-cr99.bbnplanet.net [4.0.2.91]
 16 240 ms 229 ms 251 ms  192.221.26.3
 17 235 ms 245 ms 225 ms  130.207.251.3
 18    *    268 ms 243 ms  mani.btc.gatech.edu [199.77.147.96]
Trace complete
```

Sample 2

Figure 9.6 Examples of Route Tracing (*traceroute*)

9.12 SNMP Tools

There are several tools available to obtain the MIB tree structure, as well as its values from a network element. Each of these tools has several implementations. We will not go into specific implementations here, but will describe their functionality. You may obtain details on the use and options from the manual page describing the tool.

SNMP MIB tools are of three types: (1) SNMP MIB browser uses a graphical interface; (2) a set of SNMP command-line tools, which is primarily UNIX- and Linux/FreeBSD-based tools; and (3) Linux/FreeBSD-based tool, *snmpsniff*, which is useful to read SNMP PDUs.

SNMP MIB Browser. An SNMP MIB browser is a user-friendly tool that can be accessed from public software libraries or commercially purchased. All extract MIB-II of SNMPv1 and some can extract SNMPv2 MIB. Some can also acquire private MIB objects. You specify the host name or the IP address first. Then information on a specific MIB object, or a group, or the entire MIB can be requested, depending on the implementation. The response comes back with the object identifier(s) and value(s) of the object(s).

For example, using the graphical MIB browser *tkmib* (http://www.net-snmp.org) and requesting the variable *system.sysDescr* from host 10.65.0.32 yields the response shown in Figure 9.7. The MIB tree is shown in graphical form in the top window. By clicking on a leaf node in the tree and invoking a *get*,

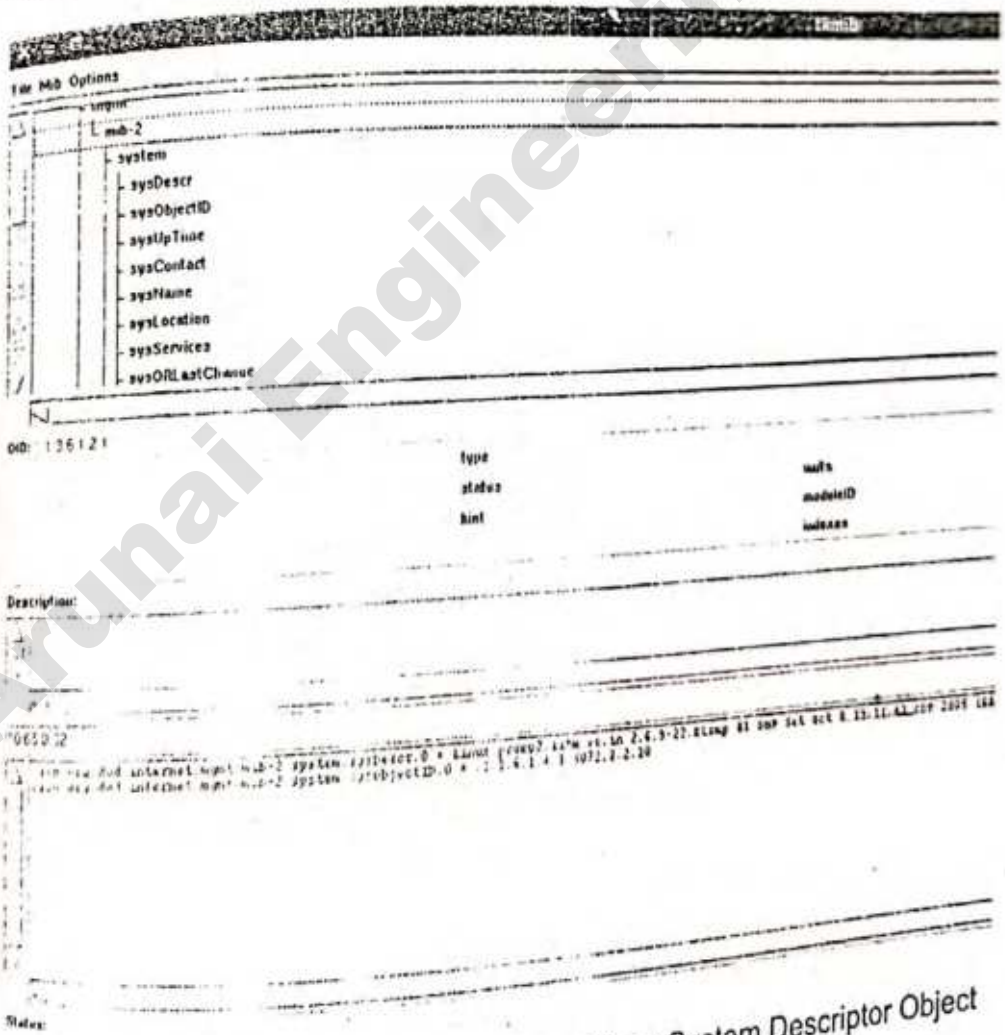


Figure 9.7 MIB Browser Example (*tkmib*) for a System Descriptor Object

```
199.77.147.182:
  sysDescr.0 : SunOS noc5 5.6 Generic_105181-03 sun4u
  sysObjectID.0 : 1.3.6.1.4.1.11.2.3.10.1.2
  sysUpTime.0 : 8d 22:21:53.74
  sysContact.0 :
  sysName.0 : noc5
  sysLocation.0 :
  sysServices.0 : 72
  sysORLastChange.0 : 0d 0:00:00.00
```

Figure 9.8 MIB Browser Example (Text Based) for a System-Group

the response from the agent is shown in the bottom panel. We see that the node is a Linux-based proxy server running kernel 2.6.9-22 with symmetrical multiprocessor (SMP) support. Figure 9.8 shows the results obtained by using a text-based browser to retrieve the system group from host 99.77.147.182.

SNMP Command-Line Tools. There are several SNMP command-line tools available in the UNIX, Linux, or FreeBSD and Windows OS environments. Command-line tools generate SNMP messages, which are get, get-next, getbulk, set, response, and trap. Public domain software packages can be downloaded that are capable of operating either as an SNMPv1, SNMPv2, or SNMPv3 manager. A popular suite is available from *http://www.net-snmp.org*. The following commands are described in SNMPv1 format. An option of –v2 or –v3 is used to generate SNMPv2c or SNMPv3, respectively.

SNMP Get Command.

snmpget [options] host community objectID [*objectID*]

This command communicates with a network object using the SNMP *get-request* message. The *host* may be either a host name or an IP address. If the SNMP agent resides on the host with the matching *community* name, it responds with a *get-response* message returning the value of the *objectID*. If multiple *objectIDs* are requested, a *varBind* clause is used to process the message containing multiple object names (see Figure 4.48). If the get-request message is invalid, the get-response message contains the appropriate error indication.

For example,

snmpget 199.77.147.182 public system.sysdescr.0

retrieves the system variable system.sysDescr

system.sysdescr.0 = "SunOS noc5 5.6 Generic_105181-03 sun4u."

The 0 at the end of the *objectID* indicates that the request is for a single scalar variable.

SNMP Get-Next Command.

snmpgetnext [options] host community objectID [objectID]

The command is similar to *snmpget* except that it uses the SNMP *get-next-request* message. The managed object responds with the expected get-response message on the *objectID* that is lexicographically next to the one specified in the request. This command is especially useful to get the values of variables in an aggregate object, i.e., in a table.

For example,
Snmpgetnext 199.77.147.182 public Interfaces.ifTable.ifEntry.ifIndex.1 retrieves
Interfaces.ifTable.ifEntry.ifIndex.2 = "2."

SNMP Walk Command.

snmpwalk [options] host community [objectID]

The *snmpwalk* command uses *get-next-request* messages to get the MIB tree for the group defined by the *objectID* specified in the request. It literally walks through the MIB. Without the *objectID*, the command displays the entire MIB tree supported by the agent.

SNMP Set Command. The *snmpset* command sends the SNMP set-request message and receives the *get-response* command.

SNMP Trap Command. The *snmptrap* command generates a trap message. Some implementation handles only SNMPv1 traps and others handle SNMPv1, SNMPv2, and SNMPv3 and can be specified in the argument. Note that this acts as an SNMP agent

SNMP Sniff Tool. The SNMP Sniff tool, *snmpsniff*, is similar to the *tcpdump* tool and is implemented in Linux/FreeBSD environment. It captures SNMP packets going across the segment and stores them for later analysis.

9.13 Protocol Analyzer

The protocol analyzer is a powerful and versatile network management tool. We will consider it as a test tool in this section, and later on look at its use as a system management tool. It is a tool that analyzes data packets on any transmission line. Although it could be used for the analysis of any line, its primary use is in the LAN environment, which is what we will focus on here. Measurements using the protocol analyzer can be made either locally or remotely. The basic configuration used for a protocol analyzer is shown in Figure 9.9. It consists of a data capture device that is attached to a LAN. This could be a specialized tool, or either a personal computer or workstation with a network interface card. The captured data are transmitted to the protocol analyzer via a dial-up modem connection, a local or campus network, or a wide area network. The protocol analyzer analyzes the data and presents it to the user on a user-friendly interface.

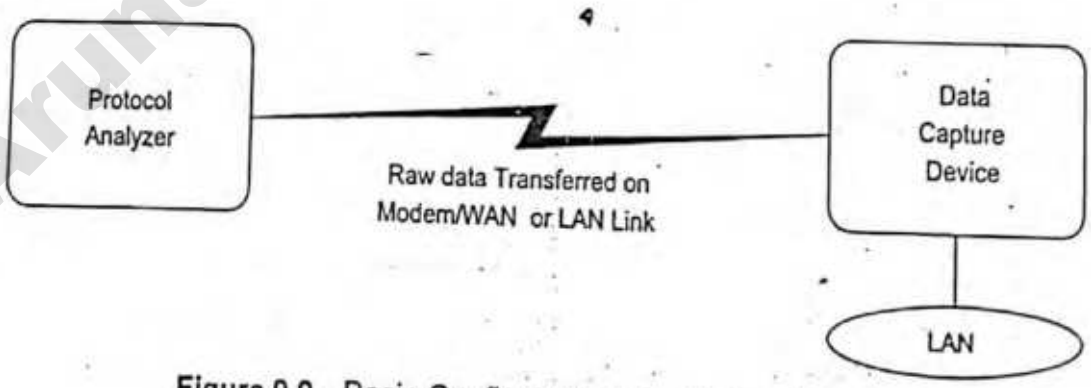


Figure 9.9 Basic Configuration of a Protocol Analyzer

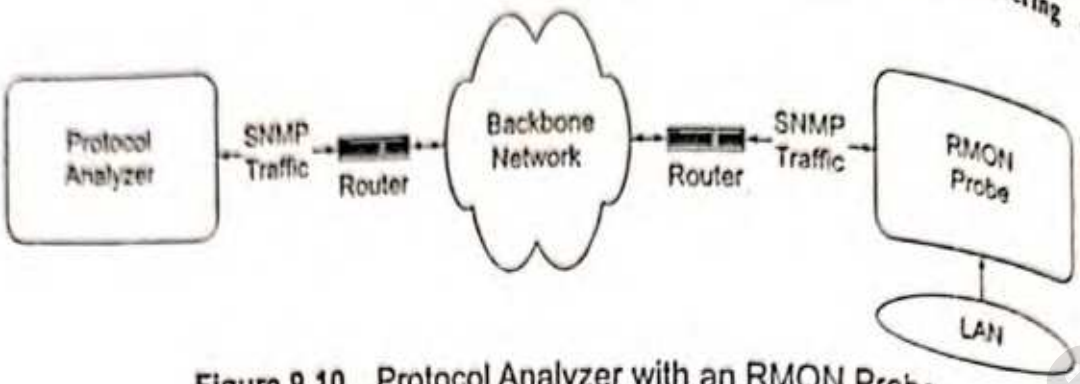


Figure 9.10 Protocol Analyzer with an RMON Probe

The protocol analyzers that are available in the commercial market are capable of presenting a multitude of results derived from the data. Contents of data packets can be viewed and analyzed at all layers of the OSI reference model. The distribution of various protocols at each layer can be ascertained. At the data link layer, besides the statistical counts, the collision rate can be measured for Ethernet LAN. At the transport layer, port information for different applications and sessions can be obtained. The distribution of application-level protocols provides valuable information on the nature of traffic in the network, which can be used for performance tuning of the network.

Numeous commercial and open-source protocol analyzers and sniffers are now available. Sniffer can be used as a stand-alone portable protocol analyzer, as well as on the network HP. NetMetrix protocol analyzer is a software package loaded on to a workstation. It uses LanProbe as the collector device, which can be configured also as an RMON probe. The communication between RMON and the protocol analyzer is based on the SNMP protocol, as shown in Figure 9.10.

A protocol analyzer functioning as a remote-monitoring analyzer collects data using an RMON probe. The raw data that are gathered are pre-analyzed by the RMON and transmitted as SNMP traffic instead of raw data in the basic configuration mentioned earlier. The statistics could be gathered over a time period for analysis or displayed on a real-time basis. In the promiscuous mode, the actual data collected by the probe could be looked at in detail, or statistics at various protocol layers could be displayed. The results are used to perform diagnostics on network problems, such as traffic congestion. They could also be used with the help of the tool to do network management functions such as traffic reroute planning, capacity planning, load monitoring, etc.

Using an RMON probe for each segment of the network and one protocol analyzer for the entire network, as shown in Figure 9.11, the complete network can be monitored. The RMON probe for each type of LAN is physically different. Even for the same type of LAN, we need a separate probe for each segment, which could be expensive to implement.

9.2 NETWORK STATISTICS MEASUREMENT SYSTEMS

One key aspect of network management is traffic management. We will consider performance management as one of the application functions when we deal with them in Chapter 11. However, we will first consider how the basic tools that we discussed in Section 9.1 are used to gather network statistics in the network at various nodes and segments. We will then cover an SNMP tool, Multi Router Traffic Grapher (MRTG), which can be used to monitor traffic.

One of the best ways to gather network statistics is to capture packets traversing network segments or across node interfaces in a promiscuous mode. We have learned that protocol analyzers do just that

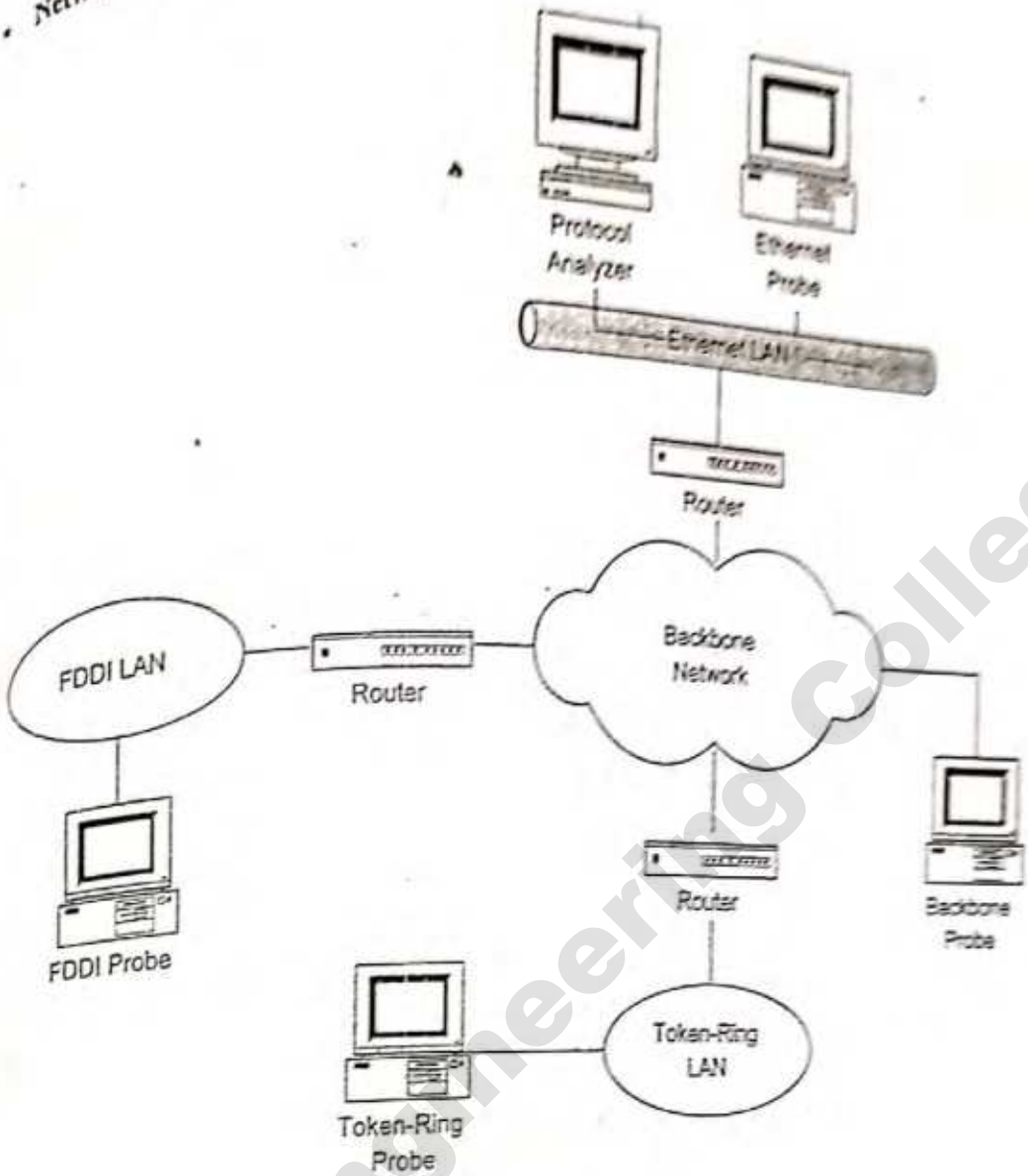


Figure 9.11 Monitoring of Total Network with Individual RMON Probes

Thus, they are good tools to gather network statistics. Another way to gather network statistics is to develop a simple application using a function similar to *tcpdump*, using a high-performance network interface card and processor, and analyze the data for the required statistics. After all, that is the basis on which protocol analyzers are built.

The RMON MIB that we studied in Sections 8.3 and 8.4, along with the SNMP communication protocol, provides a convenient mechanism to build network-monitoring systems. The configurations shown in Figure 9.10 and Figure 9.11 can be used as the network-monitoring system to gather various RMON objects. The RMON1 MIB groups and tables shown in Tables 8.2 and 8.3 are used to gather statistics at the data link layer in Ethernet and token-ring LANs. The RMON2 MIB groups and tables presented in Table 8.4 define parameters for higher-layer statistics.

9.2.1 Traffic Load Monitoring

Traffic load monitoring can be done based on the source, the destination, and the source–destination pair. We may want to balance the traffic load among the various LAN segments, in which case we need

to measure the total traffic in each network segment or domain. Data for traffic monitoring can be sampled at the data link layer using the RMON1 MIB history group. Traffic relevant to a host, either as source or as a destination, is available in the host group. Hosts can be ranked on the traffic load that they carry using the *HostTopN* group. In the absence of an RMON probe, there is no convenient way to measure traffic in a segment directly except to compute if externally knowing the hosts in the segment.

Load statistics in an IP network can also be obtained by measuring IP packets at the network layer level. The entities in the network layer host and the network layer matrix groups in RMON2 MIB can be used for this measurement. Figure 9.12, Figure 9.13, and Figure 9.14 show the load statistics measured in a Fiber Distributed Data Interface (FDDI) LAN segment using the NetMetrix protocol analyzer as the Load Monitor and FDDI probe.

In Figure 9.12 there are 1,609 sources that generated data packets. The top ten have been identified, with the highest entry being news-ext.gatech.edu. The entry LOW-CONTRIB is a combination of sources other than those specifically identified. Traffic is measured as the number of octets. Figure 9.13 presents similar statistical data on traffic that is destined to the hosts in the network segment. Figure 9.14 presents the top ten conversation pairs of hosts. Each line identifies the host-pairs, traffic from NetHost1 to NetHost2. Oct1to2 and Oct2to1 denote the traffic in octets from NetHost1 to NetHost2 and NetHost2 to NetHost1, respectively. For example, news-ext.gatech.edu transmitted 3K octets/second of outgoing traffic to and received 60K octets of incoming traffic from howland.erols.net.

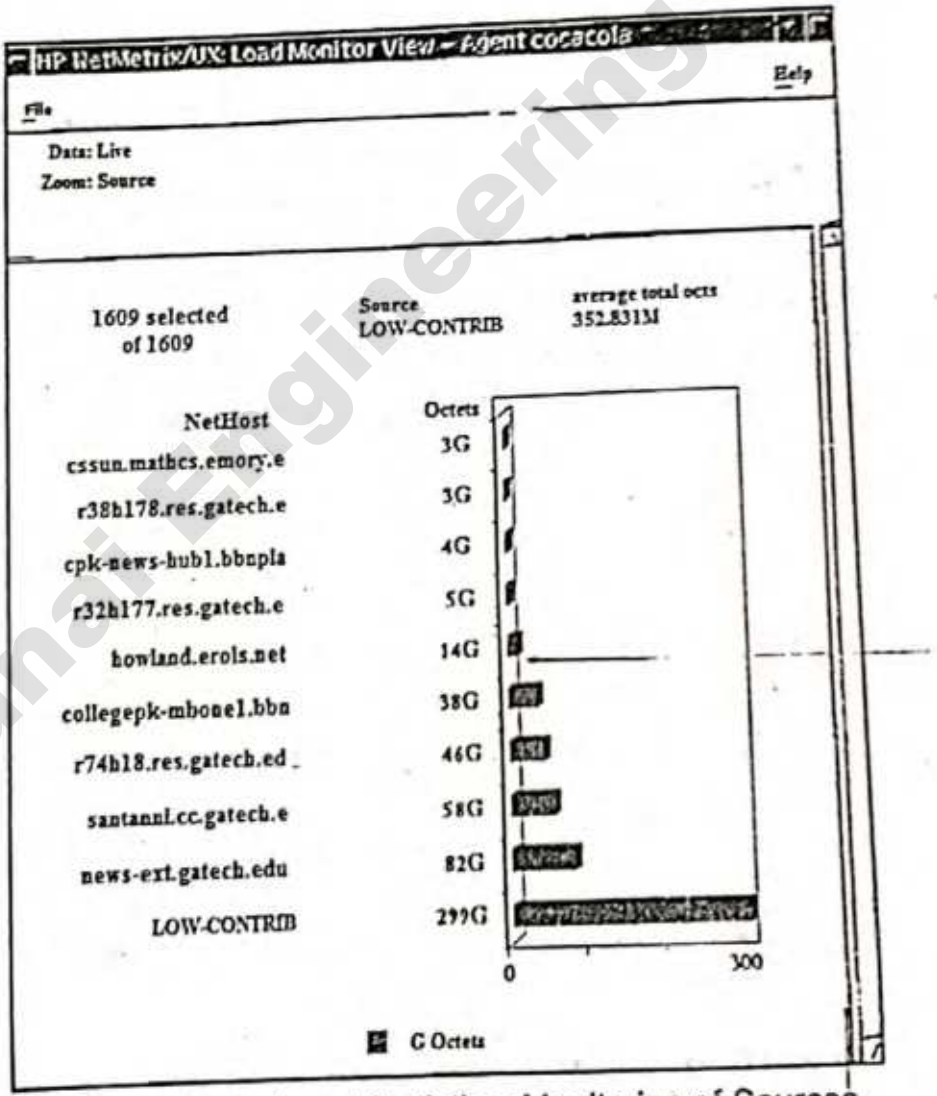


Figure 9.12 Load Statistics: Monitoring of Sources

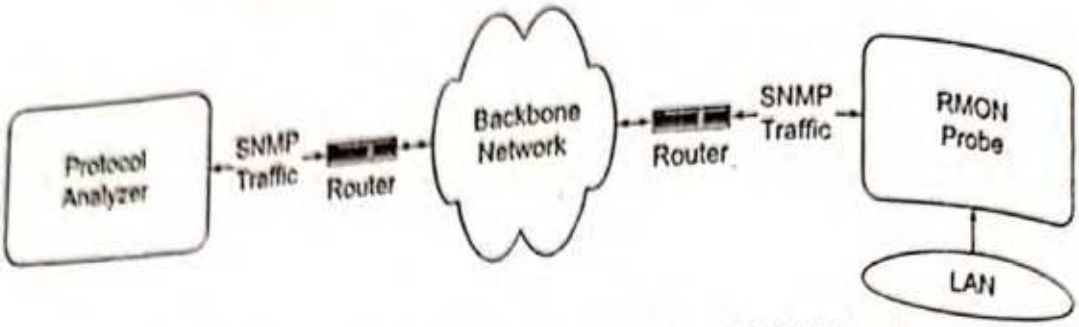


Figure 9.10 Protocol Analyzer with an RMON Probe

The protocol analyzers that are available in the commercial market are capable of presenting a multitude of results derived from the data. Contents of data packets can be viewed and analyzed at all layers of the OSI reference model. The distribution of various protocols at each layer can be ascertained. At the data link layer, besides the statistical counts, the collision rate can be measured for Ethernet LAN. At the transport layer, port information for different applications and sessions can be obtained. The distribution of application-level protocols provides valuable information on the nature of traffic in the network, which can be used for performance tuning of the network.

Numeous commercial and open-source protocol analyzers and sniffers are now available. Sniffer can be used as a stand-alone portable protocol analyzer, as well as on the network HP. NetMetrix protocol analyzer is a software package loaded on to a workstation. It uses LanProbe as the collector device, which can be configured also as an RMON probe. The communication between RMON and the protocol analyzer is based on the SNMP protocol, as shown in Figure 9.10.

A protocol analyzer functioning as a remote-monitoring analyzer collects data using an RMON probe. The raw data that are gathered are pre-analyzed by the RMON and transmitted as SNMP traffic instead of raw data in the basic configuration mentioned earlier. The statistics could be gathered over a time period for analysis or displayed on a real-time basis. In the promiscuous mode, the actual data collected by the probe could be looked at in detail, or statistics at various protocol layers could be displayed. The results are used to perform diagnostics on network problems, such as traffic congestion. They could also be used with the help of the tool to do network management functions such as traffic reroute planning, capacity planning, load monitoring, etc.

Using an RMON probe for each segment of the network and one protocol analyzer for the entire network, as shown in Figure 9.11, the complete network can be monitored. The RMON probe for each type of LAN is physically different. Even for the same type of LAN, we need a separate probe for each segment, which could be expensive to implement.

9.2 NETWORK STATISTICS MEASUREMENT SYSTEMS

One key aspect of network management is traffic management. We will consider performance management as one of the application functions when we deal with them in Chapter 11. However, we will first consider how the basic tools that we discussed in Section 9.1 are used to gather network statistics in the network at various nodes and segments. We will then cover an SNMP tool, Multi Router Traffic Grapher (MRTG), which can be used to monitor traffic.

One of the best ways to gather network statistics is to capture packets traversing network segments or across node interfaces in a promiscuous mode. We have learned that protocol analyzers do just that.

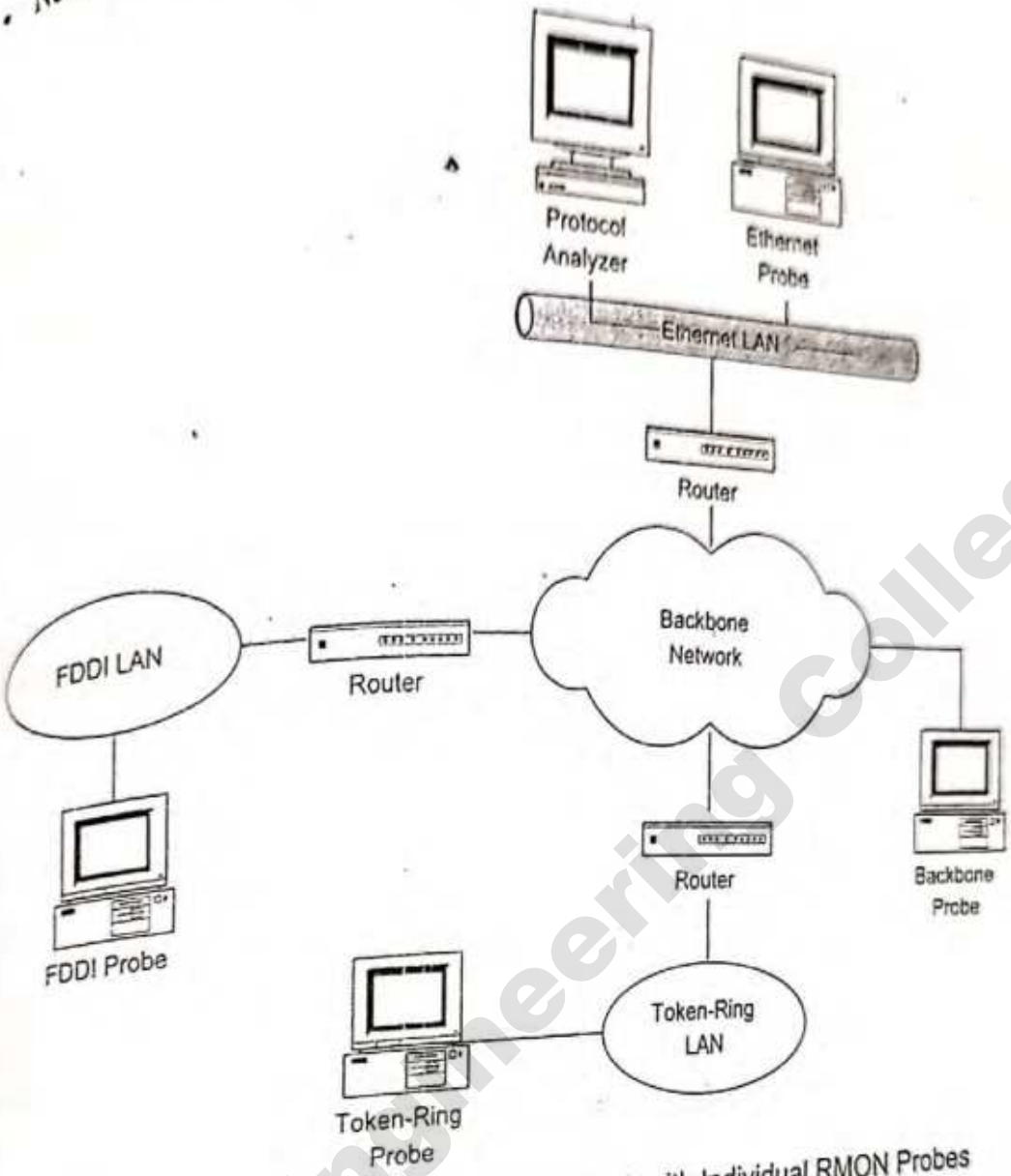


Figure 9.11 Monitoring of Total Network with Individual RMON Probes

Thus, they are good tools to gather network statistics. Another way to gather network statistics is to develop a simple application using a function similar to *tcpdump*, using a high-performance network interface card and processor, and analyze the data for the required statistics. After all, that is the basis on which protocol analyzers are built.

The RMON MIB that we studied in Sections 8.3 and 8.4, along with the SNMP communication protocol, provides a convenient mechanism to build network-monitoring systems. The configurations shown in Figure 9.10 and Figure 9.11 can be used as the network-monitoring systems. The configurations shown in objects. The RMON1 MIB groups and tables shown in Tables 8.2 and 8.3 are used to gather various RMON the data link layer in Ethernet and token-ring LANs. The RMON2 MIB groups and tables presented in Table 8.4 define parameters for higher-layer statistics.

9.2.1 Traffic Load Monitoring

Traffic load monitoring can be done based on the source, the destination, and the source–destination pair. We may want to balance the traffic load among the various LAN segments, in which case we need

to measure the total traffic in each network segment or domain. Data for traffic monitoring can be sampled at the data link layer using the RMON1 MIB history group. Traffic relevant to a host, either as source or as a destination, is available in the host group. Hosts can be ranked on the traffic load that they carry using the *HostTopN* group. In the absence of an RMON probe, there is no convenient way to measure traffic in a segment directly except to compute it externally knowing the hosts in the segment.

Load statistics in an IP network can also be obtained by measuring IP packets at the network layer level. The entities in the network layer host and the network layer matrix groups in RMON2 MIB can be used for this measurement. Figure 9.12, Figure 9.13, and Figure 9.14 show the load statistics measured in a Fiber Distributed Data Interface (FDDI) LAN segment using the NetMetrix protocol analyzer as the Load Monitor and FDDI probe.

In Figure 9.12 there are 1,609 sources that generated data packets. The top ten have been identified, with the highest entry being news-ext.gatech.edu. The entry LOW-CONTRIB is a combination of sources other than those specifically identified. Traffic is measured as the number of octets. Figure 9.13 presents similar statistical data on traffic that is destined to the hosts in the network segment. Figure 9.14 presents the top ten conversation pairs of hosts. Each line identifies the host-pairs, traffic from NetHost1 to NetHost2. Oct1to2 and Oct2to1 denote the traffic in octets from NetHost1 to NetHost2 and NetHost2 to NetHost1, respectively. For example, news-ext.gatech.edu transmitted 3K octets/second of outgoing traffic to and received 60K octets of incoming traffic from howland.erols.net.

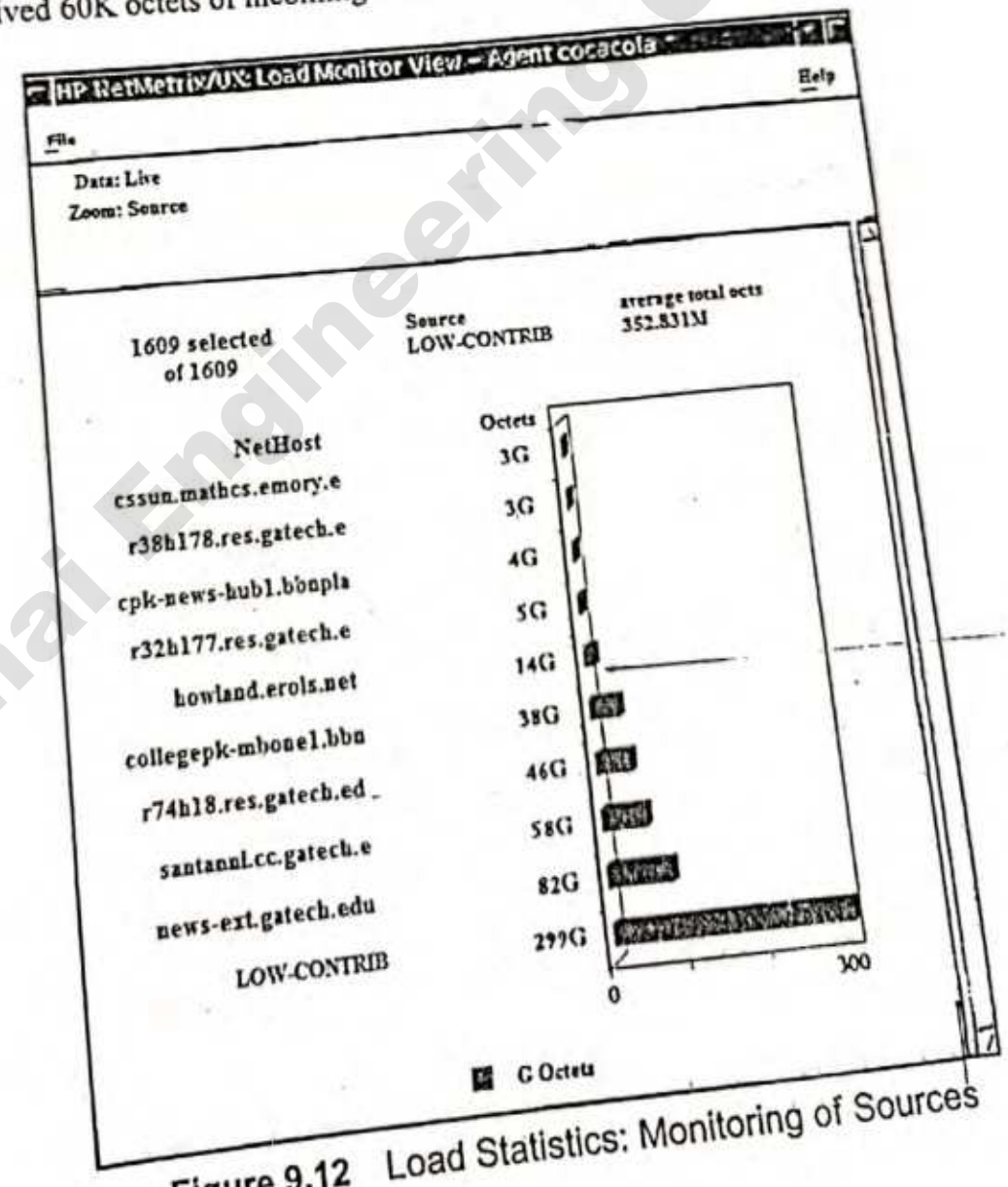


Figure 9.12 Load Statistics: Monitoring of Sources

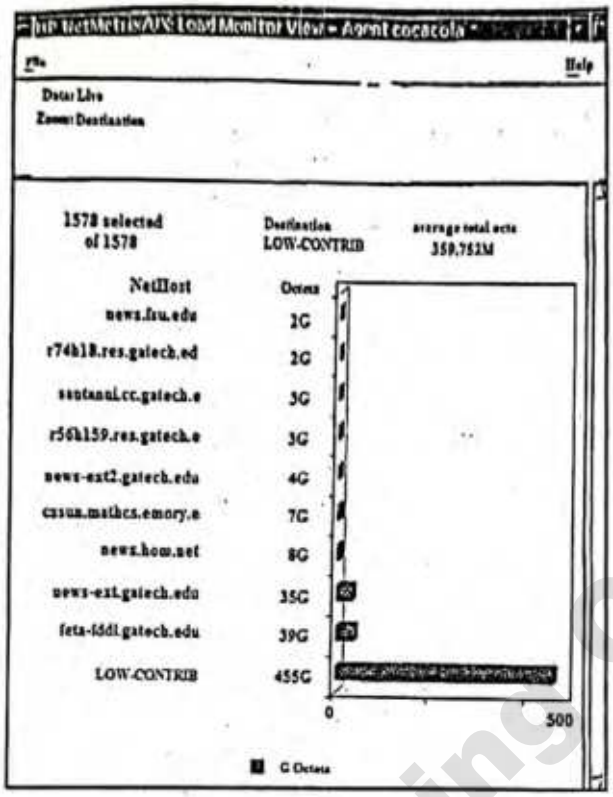


Figure 9.13 Load Statistics: Monitoring of Destinations

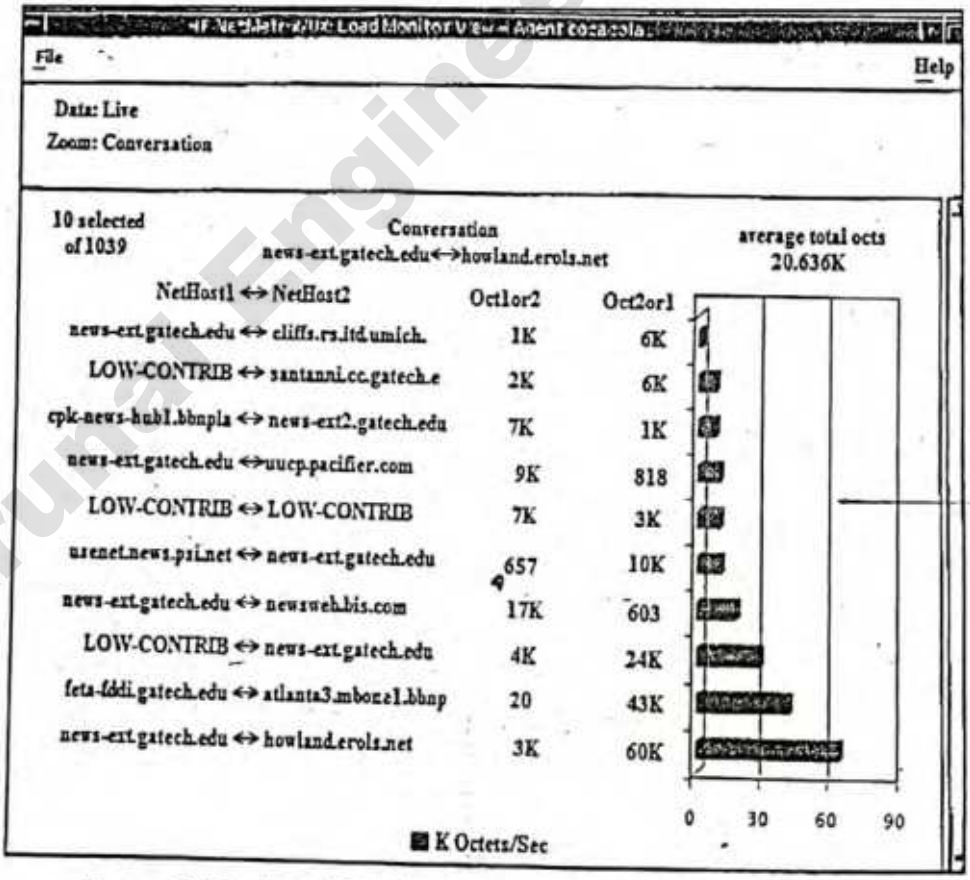


Figure 9.14 Load Statistics: Monitoring of Conversation Pairs

9.2.2 Protocol Statistics

Packets can be captured by data capture devices based on the filter set for the desired criteria. From the captured data, we can derive protocol statistics of various protocols at each layer of the OSI Reference Model. This is very useful at the application layer level. We can obtain the traffic load for different applications such as file transfer (FTP), Web data (HTTP), and news groups (NNTP). This information can be used for bandwidth management of real-time and non-real-time traffic.

Figure 9.15 shows the distribution of protocols at the data link (top left corner), network (top right corner), transport (bottom left corner), and application (bottom right corner) layers, obtained using NetMetrix LanProbe and a protocol analyzer. Data link and network layers show 100% LLC and IP protocols. The majority of the transport layer protocol packets belong to TCP and the next in order is UDP. The other category is undefined. At the application layer(s), the distribution contains HTTP (Web protocol), NNTP (news protocol), FTP-data, UDP-other, TCP-other, and undefined other. The Georgia Tech Internet backbone network in which the measurements were made carries a complex variety of protocol traffic including multimedia traffic and next generation Internet traffic.

9.2.3 Data and Error Statistics

Data and error statistics can be gathered directly from managed objects using the specifications defined in various MIB groups. The RMON statistics groups for Ethernet [RFC 1757] and token ring [RFC 1513] contain various types of packets and errors in the data link layer. Similar information is available on higher-level layers from specifications detailed in RMON2 [RFC 2021]. Information on statistics can also be gathered for the individual medium from the respective MIBs under the transmission group. For example, statistics on Ethernet can be derived from the Ethernet-like statistics group in Ethernet-like

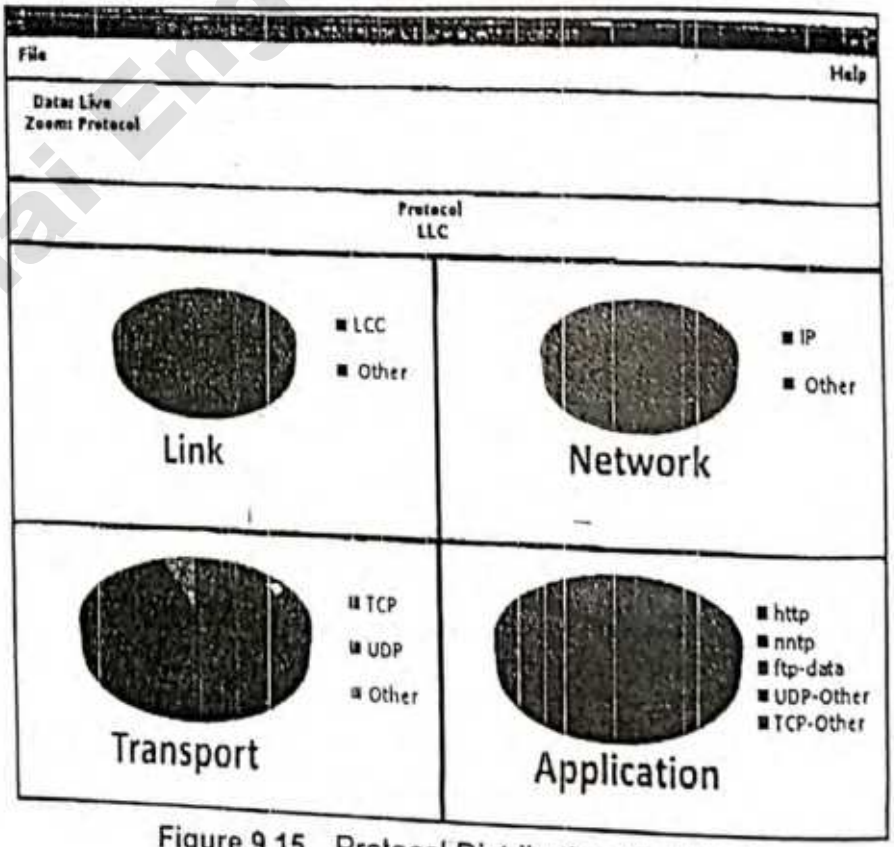


Figure 9.15 Protocol Distribution (NetMetrix)

interface types MIB [RFC 1284], token-ring statistics from IEEE 802.5 MIB [RFC 1748], and FDDI data from FDDI MIB [RFC 1285].

9.2.4 Using MRTG to Collect Traffic Statistics

The MRTG is a tool that monitors traffic load on network links [Oetiker and Rand, http://oss.oetiker.ch/mrtg]. It generates a live visual representation of traffic data by reading the SNMP traffic counters on routers and creates graphs that are embedded into Web pages. These can be monitored using any Web browser. Visual presentations of traffic data are presented as daily view, the last 7 days view, the last 4 weeks view, and the last 12 months view. The generic software can be implemented in either UNIX or Windows NT platform. An example of the views can be seen on http://switch.ch/network/operation/statistics/geant2.html.

9.3 MIB ENGINEERING

In the SNMP model of management, information about each network element is contained in its MIB (Chapter 4). The manager's view of the NE is defined by and is limited to its MIB. The ease of developing management applications depends on how closely the MIB matches the needs of the manager. In most cases, the MIB is hardcoded into the NE by the vendor. Thus, care must be taken in designing the MIB. This is the focus of *MIB engineering*.

There are some commonly used constructs in many MIBs. The use of these *idioms* makes it easier for the manager to understand and use the MIB. The SNMP protocol and structure of management information (SMI), by virtue of their stress on simplicity, have some limitations. Fortunately, there are work-arounds to enable the manager to accomplish complex tasks despite these limitations.

First we cover some basic principles, limitations, and idioms of SMI. We then take a number of frequently occurring requirements and show how to design MIBs for them.

9.3.1 General Principles and Limitations of SMI

SMI provides a very simple view of the network. Every element is completely defined by a set of variables (euphemistically referred to as *objects*), which may take on a limited number of data types. These include scalar or primitive types (Boolean, Integer, IP address, String, Counter, etc.) and three structured or constructed types (arrays, records, and sets).

An *array* is an ordered list of elements all being of the same type. Each element is identified by its index. A *record* is an ordered collection of elements of different types, with each element referred to by name. A *set* is an unordered collection of elements. The elements could either all be of the same type or of different types. E.g., the interfaces table, *ifTable*, is an array with one element for each interface in the node. Each element in the table contains a record with several fields describing the interface (see Figure 4.28).

As an example of a set, suppose the manager wishes to define a trap filter in an agent. The filter consists of a set of conditions that are AND'ed together. Only if all conditions are satisfied, the trap is forwarded. We could define:

```
TrapFilter ::= SET OF Conditions
```

The order of evaluation of the conditions does not matter.

There are restrictions on the construction of types:

- An array can contain a record and vice versa. An array can contain a record that itself contains other records. However, an array cannot contain a record that contains an array
- A record can be defined to hold related variables pertaining to one part of an NE (in one subtree of the MIB). Another record with identical fields but different names must be defined

The SNMP protocol allows a manager to get or set the value of a variable (see Figure 3.9). However, it does not permit a manager to perform an action such as resetting an interface or deleting a file. In object-oriented terms, an SNMP object has member variables and accessor methods, but does not have any general methods to perform actions.

The SNMP protocol supports request response transactions where each transaction can automatically either get or set a limited number of variables. The limit is imposed by the size of an SNMP PDU. It does not allow combining gets and sets in one transaction, nor does it allow a sequence of operations to form a session. SNMP transactions are very limited compared to database transactions. The latter allow a large sequence of select and update queries to be combined in a single transaction, which either succeeds completely or is not executed at all. In addition, access to the relevant parts of the database by other users can be prevented during the transaction.

9.3.2 Counters vs. Rates

Rates are central to network management. Performance is measured largely in terms of rates. For example, the throughput of a link is given in bits/second or packets/second, the throughput of a server in transactions/second, and user behavior in requests/hour.

Rates are also important in some aspects of fault management, specifically in determining when the load on the system is reaching its capacity and hence is liable to fail. For example, if a 1 Mb/s link is subjected to traffic at the rate of 0.98 Mb/s, congestion is very likely with the consequent increase in delays, packet loss, timeouts, and retransmissions. If the congestion persists, it could result in the failure of end applications or the routers at either end of the link. Proactive fault management attempts to spot congestion in its nascent stage and then take congestion avoidance measures to prevent failures. The onset of congestion is indicated by comparing the live throughput with predefined thresholds. The threshold depends on the capability of the router to tolerate temporary overload by means of buffering of packets. Suppose for a router with a 10-packet buffer we set the congestion threshold at 0.8 Mb/s; if the router has a 20-packet buffer, we might increase the threshold to 0.9 Mb/s. The threshold settings depend on the mix of packet sizes and other practical factors. In practice, the threshold is tuned by the operator based on experience.

The manager may also require the counter value. For example, if broadband subscribers are billed based on data transferred, the NMS manager would retrieve the counter of bytes transferred and pass it on to the Billing System.

A counter is a direct performance measure. On a network link, the network interface maintains several counters such as packets and bytes transmitted, packets and bytes received, errors, etc. Whenever a packet is processed, one or more counters are incremented appropriately. These counters are thus available to the NMS agent at no extra cost, are always up to date, and accurate. Hence, the MIB should include the counter. Be mindful that the counter reading wraps around and should be interpreted correctly.

client may display them in green (normal) instead of red due to an incompatibility. This is much more serious as the operator will ignore the fault.

The situation is aggravated when an operator needs to login to several NMS servers in different regional networks. The servers, from the same vendor, may be of different versions. The operator's PC will need to have several versions of the client application installed and the operator would have to run the corresponding client application depending on which NMS server s/he is working with!

Browser or Web Client. The browser or Web client promises to combine the advantages of the terminal and rich clients without their disadvantages. It is rapidly becoming *de facto* for NMS clients (and for many other applications also). The browser client provides all its functionality through the GUI of a Web browser such as Firefox, Internet Explorer, or Safari. (Firefox is open source and runs on almost any OS platform, Internet Explorer runs only on Microsoft Windows, and Safari is supplied with MacOS X. The NMS server includes a Web server to which the user connects via the browser. The NMS server throws up Hyper Text Markup Language (HTML) pages to the browser. For a better user experience, the HTML pages may include some client-side processing through Javascript. Network maps and icons are usually provided using Ajax.

Since there is no software installed on the client PC, there are no issues of version incompatibility between the client and the server. Any client-side NMS processing is done through Javascript code that is downloaded in the HTML page. It is not stored on the disk of the client PC (though it may be cached temporarily to enhance performance).

Today, browsers are ubiquitous. Almost any PC has a good, recent browser installed. Many mobile phones, including some low-end models, can run a browser, though the small screen limits the amount of information that can be presented. Almost everyone who uses a PC and the Internet is familiar with the use of a browser. Hence, the training effort for new users is greatly reduced.

Owing to differences in the way in which different browsers render an HTML page, the problem of portability to different browsers is still an issue, albeit a much smaller one than that of portability of rich client applications. For instance, a data entry form in which the labels and boxes are aesthetically placed in one browser may have some of these GUI components overlapping in another browser. So, the server needs to determine which browser the user is using and feed HTML pages that are tuned to the peculiarities of that browser. Likewise, Javascript code that works on one browser may not work on another browser.

Fortunately, the incompatibilities between browsers are decreasing and are well-documented. Also, there are a variety of open source and proprietary code libraries available that permit the developer to write code that works equally well across a variety of browsers.

9.4.10 Summary: NMS Design

Starting from the requirements for management of a telecom network or a large enterprise network, we derived the architecture for an NMS server. After some general design decisions, we discussed the design of the main modules in an NMS server, the discovery module, PM, and the FM.

The FM is the most complex of the modules in an NMS. This is because it has to deal in real time with a wide range of events that occur at unpredictable times. Rapid fault detection and rectification is the key to the operation of a network. The FM is especially relied upon when there are serious faults in the network. At such a time, the FM would experience a burst in processing requirements. The design of the FM needs to be especially careful to ensure that the NMS itself does not fail due to overload during such periods of severe network problems.

We traced the path of an event through the FM. We explained different methods used to indicate the fault to the operator and the various alarm states. We explored assorted techniques for the correlation of

seemingly unrelated events and alarms. These range from the simple matching of fields in event records, to sophisticated AI and graph algorithms.

Finally, we discussed approaches to distributed management. This enables scaling to very large networks. It also mirrors the structure of typical large organizations.

9.5 NETWORK MANAGEMENT SYSTEMS

So far we discussed the use of simple system utilities and tools for management. This was followed by a detailed examination of the design of a high-end NMS server. In the rest of this chapter we will describe several commercial and open-source NMSs. We start with the management of networks, and then cover management of systems and applications. This is followed by enterprise management and telecommunications network management. Finally, we describe approaches for distributed management.

9.5.1 Network Management

A network consists of routers, switches, and hubs connected by network links. Servers, workstations, and PCs are connected to LANs in the network. Various access technologies may be used. In network management, we are primarily interested in the health and performance of the routers, switches, and links. We may also monitor the health of servers.

The first task involved in network management is the **configuration** of the above network elements, their agents, and the NMS itself. This includes discovery of network elements and the topology of the network.

Daily users of NMSs are people in the NOC who do not have the same engineering background as those who designed and implemented the systems. Therefore, ease of use is an important factor in the selection of NMSs. For example, an operator does not constantly sit in front of a monitor and watch for failures and alarms. Thus, when an alarm goes off, it should attract the attention of the operator visibly, audibly, or both. It should present a global picture of the network and give the operator the ability to "drill down" to the lowest level of component failure by successive point-and-click operations of the icon indicating an alarm.

Figure 9.36, Figure 9.37, and Figure 9.38 show hierarchical views of a network testbed at the NMSLab, IIT Madras, which were captured with a CygNet NextGen NMS. (The IP addresses of the nodes have been changed for security reasons.) Figure 9.36 shows the global view with network segments and numerous domains behind routers and gateways. Figure 9.37 is obtained by clicking the mouse (also colloquially referred to as drilling down) on the network segment icon 192.168.9.0 in Figure 9.36 and shows the nodes that are part of that private LAN. The LAN has a switch (SW-11) connected to two routers RTR-1, and RTR-2), and a few other IP hosts. The port details on the switch SW-11 are obtained by drilling down on the SW-11 icon in Figure 9.37. The result shown in Figure 9.38 contains 12 ports, some of which are free.

Next, the **fault management** capability of the NMS must support monitoring of the health of the NEs and links. In an IP network, this is usually done using ICMP ping and SNMP get messages. The NMS reports faults to the operator in a variety of ways depending on the nature, severity, and importance of the fault. It may assist in the rectification of the fault.

The NMS must support **performance management** especially of the expensive WAN links. Planning and management reports keep upper-level management apprised of the status of the network and system operations. Reports in this category include network availability, systems availability, problem reports, service response to problem reports, and customer satisfaction.

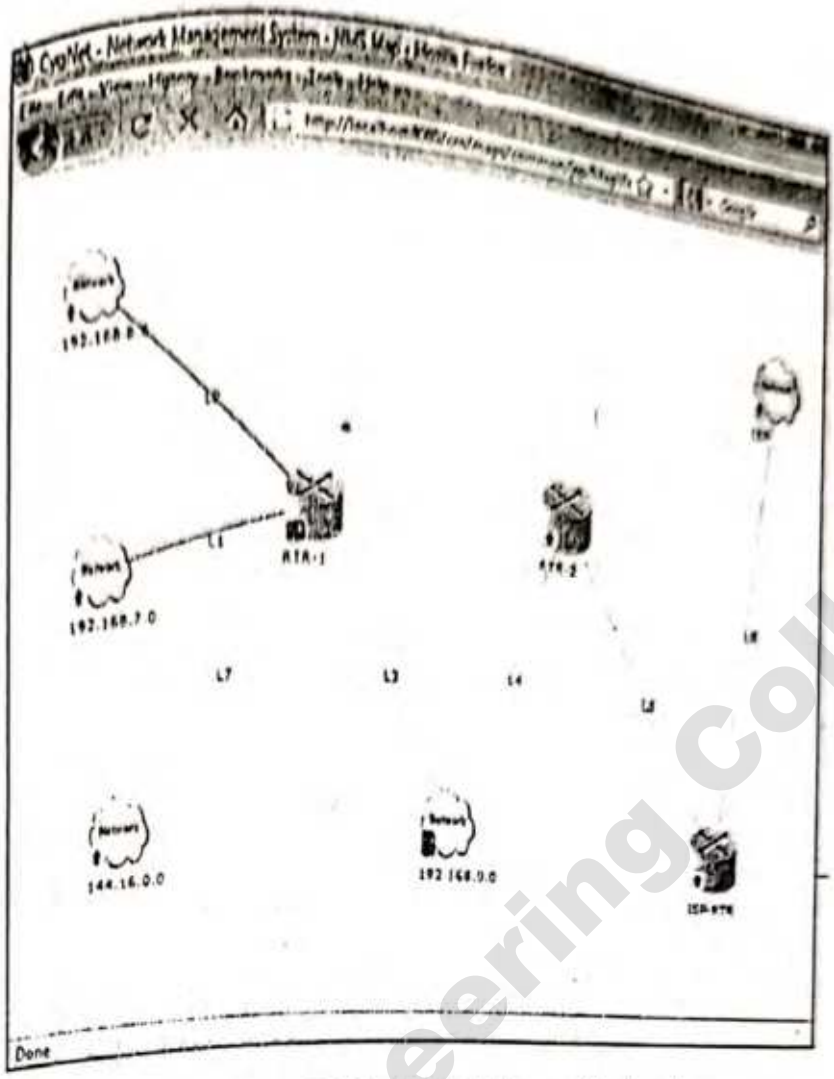


Figure 9.36 Global View of Network

Performance management provides traffic trend reports to enable the network administrator to identify bottlenecks. The administrator can take action to alleviate the bottleneck, such as re-routing traffic via an alternate route, or changing priorities for different classes of traffic. Performance reports help the administrator see long-term traffic trends in order to plan capacity expansion in a timely and cost-effective manner. Trends in traffic should address traffic patterns and volume of traffic in internal networks, as well as external traffic.

Accounting management is probably the least developed function of network management applications. Accounting management could include individual host use, administrative segments, and external traffic. Accounting of individual hosts is useful to identify some of the hidden costs. For example, the library function in universities and large corporations consumes significant resources and may need to be accounted for functionally. This can be done by using the RMON statistics on hosts.

The cost of operations for the Information Management Services department is based on the service that it provides to the rest of the organization. For planning and budget purposes, this may need to be broken into administrative group costs. The network needs to be configured so that all traffic generated by a department can be gathered from monitoring segments dedicated to that department.

External traffic for an institution is handled by service providers. The tariff is negotiated with the service provider based on the volume of traffic and traffic patterns, such as peak and average traffic. An internal validation of the service provider's billing is a good practice.

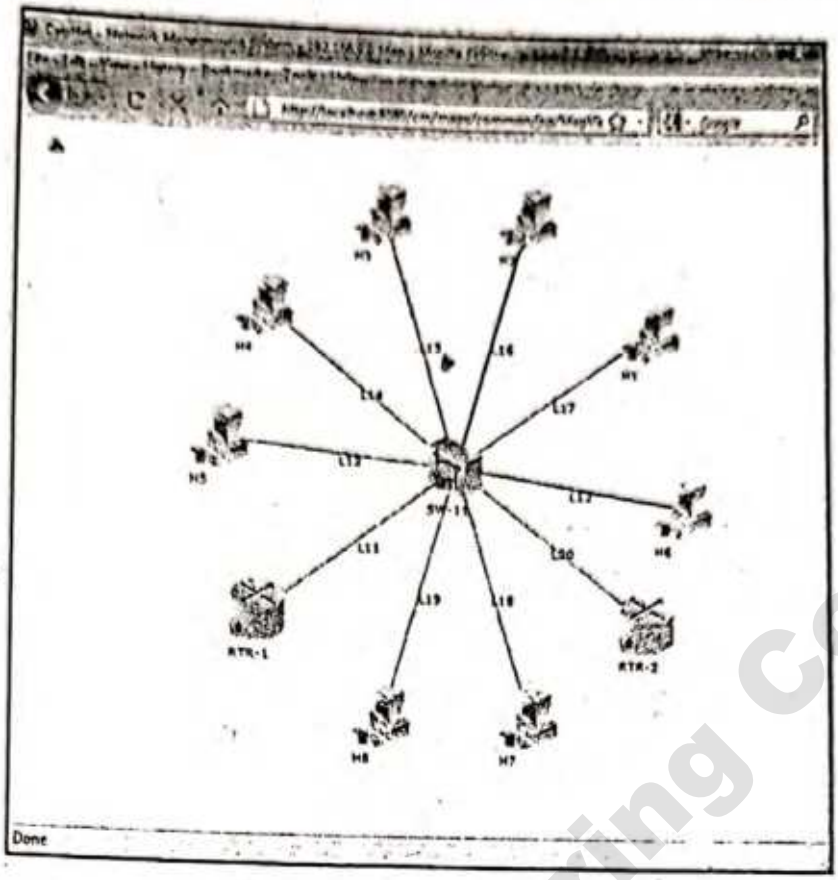


Figure 9.37 Domain View

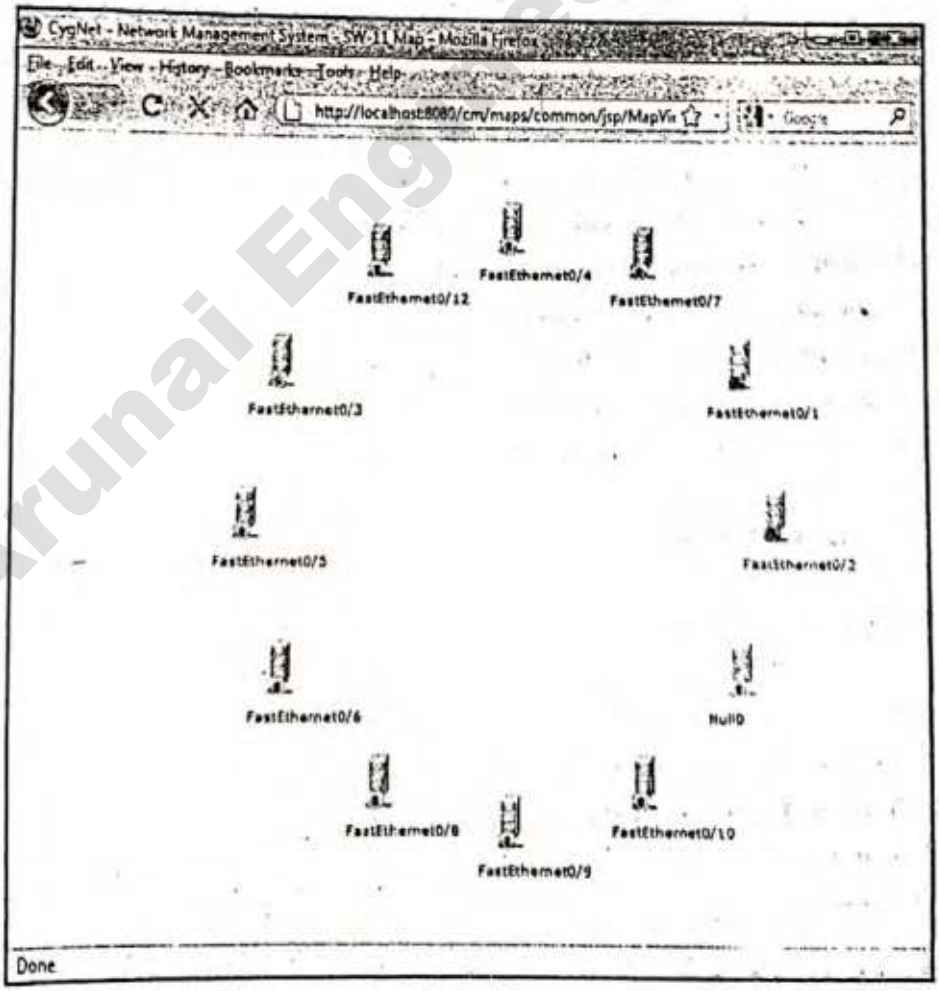


Figure 9.38 Interfaces View

Security management is both a technical and an administrative issue in information management. It involves securing access to the network and the information flowing in the network, access to data stored in the network, and manipulating the data that are stored and flowing across the network. The scope of network access not only covers enterprise intranet network, but also the Internet that it is connected to.

Security management also covers security of the NMS itself. The NMS database contains a wealth of often confidential information about the organization. This must be made available to authorized personnel, but kept away from all others. Likewise, a user of the NMS could reconfigure NEs throughout the network. Hence, login to the NMS must be carefully controlled.

Thus, network management involves the complete **FCAPS** spectrum of application functions defined in Chapter 3. Configuration, fault, and performance management are found in almost every network management deployment. In some cases, the NMS is also used for security and accounting management.

OpenNMS. This is an open-source NMS (http://www.opennms.org), which claims to be the first project aiming to build a complete enterprise-class open-source NMS. OpenNMS is written largely in Java and has a browser-based UI. It is primarily used for managing SNMP devices. It is used to manage small networks of under 25 NEs to large networks with over 80,000 NEs.

The major functional areas covered by OpenNMS are autodiscovery, status polling of NEs, performance polling, and fault management. Reports are provided using the JFreeChart package. Much of the operation of OpenNMS can be customized by the user by means of filters. A filter consists of rules, each of which is essentially a simplified form of an SQL statement. It configures how the component of the NMS behaves. For example, the notification rules control whether or not a received event triggers a notification to the user. Filters can be added to control autodiscovery, to specify the list of IP interfaces that are included in data collection, polling, etc.

Unlike many commercial NMS products, OpenNMS does not have a graphical map for the display of the NEs and their status. The designers of OpenNMS believe that seasoned network administrators prefer to see event lists. OpenNMS provides event lists grouped in various convenient ways. On the main WebUI page there is a "real-time console" (RTC) that reflects the status of categories of devices. These categories reflect groups of devices like database servers, Web servers, etc. However, anything in the database can be used to create a custom category list, and grouping devices by location, building, vendor, IP range, etc., is very common. The categories list follows a basic tenet of OpenNMS; once configured, it should be simple to use and as automated as possible. As new devices are added, the categories automatically update. There is no need for manual customization, such as would be required with a useful map. (There is a group of developers working on a map, so this may become available in due course.)

By default, the FM receives SNMP traps. Other event detectors can be configured through XML configuration files. When an event is accepted, it results in a notification to the user, which can be on-screen, via email, SMS, etc.

In keeping with the philosophy of being utilitarian, OpenNMS has a variety of reports. Most are simple tables with some graphs. They lack the frills that may be attractive to a novice, but contain a wealth of information in a format that is easy for a network administrator to comprehend.

With OpenNMS, the user has the ability to do comprehensive monitoring of a network at the price of just the server hardware. The NMS is customizable by the network administrator without any programming. As with any open-source product, the enterprise has the comfort that it could always get unusual customizations done as the source code is freely available. Currently, OpenNMS supports F, P, and part of C in FCAPS. With

the on-going efforts of the developer community, it is likely that other functional areas will be covered in future. Commercial support is also available from the OpenNMS Group for enterprises that need it.

9.5.2 System and Application Management

Network management addresses only the managing of a network (i.e., managing the transport of information). System management deals with managing system resources, which complements network management. For example, *ping* is used to test whether a host is alive. However, we want to know more about the use of system resources on the host, such as the amount of CPU use on the host or whether a specific application is running on the host.

Historically, enterprises had two separate and distinct organizations. The telecommunications department took care of communications, giving rise to network management. The management of information systems (MIS) department took care of the computers, a task that involved system management. However, in the current distributed environment of client/server architecture, the computing depends heavily on the communication network and the distinction has disappeared. System and network management now form a single umbrella, headed by a chief information officer. System and network management are beginning to be considered together as a solution for information management issues and problems. System management tools, which used to be custom developed, are currently available as commercial systems and are being integrated with network management.

System management tools monitor the performance of computer systems. Some parameters that can be monitored are (1) CPU use, which measures the number of processes that are running and the resource consumption of each; (2) status of critical background processes (called *daemons* in UNIX terminology); and (3) application servers such as the Simple Mail Transfer Protocol (SMTP), File Transport Protocol (FTP), and Domain Name Server (DNS). System management may also include backup of server databases, desktop workstations, and operations support systems that support operations such as help desk and trouble ticket tracking.

Several UNIX-based tools can be used to monitor systems. Such tools are constantly evolving and are updated on Web sites that are used to track them. http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html is a site active from 1996 that provides a comprehensive list of commercial and open-source network management tools.

High-End System Management. The Computer Associates (CA) Unicenter TNG and Tivoli Enterprise Manager TME 10 are two integrated systems solutions available commercially [ZDNet]. Both solutions offer features that can be classified as high end and thus require the vendor's ongoing active participation. They meet the requirements of large enterprises, particularly as they are offered as integrated solutions, which we will discuss in Section 9.5.3.

Low-End System Management. System management of hosts can be accomplished by installing simple and free public domain software and configuring it to local system management.

Nagios. This is a fairly comprehensive open-source NMS (http://www.nagios.org). The project has been active for over 10 years so it is stable. The design is scalable and extensible.

Nagios provides the basic network management features. These include status and performance polling, fault handling, and configuration management. The FM can indicate faults via a graphical map-

color-coded event lists, and email, pager, and cellphone. It allows the administrator to schedule the downtime of hosts, servers, and the network. The reporting feature supports capacity planning.

Unlike other NMSs, Nagios does not have built-in mechanisms for polling. Instead, it relies on external plugins. These can be executables or scripts and hence give substantial flexibility. Nagios by default has support for managing routers, switches, and other IP network components; Windows, Linux, UNIX, and Netware servers; network printers; publicly available services such as HTTP, FTP, SSH, etc. With its extensible design, Nagios has more than 200 community-developed plugins available.

It has a rich UI, which includes a map (unlike OpenNMS described in Section 9.5.1). Sample screenshots are available at http://www.nagios.org/about/screenshots.php. The UI is customizable to each user. This facilitates specialization and also helps maintain security of sensitive information.

As with OpenNMS, Nagios is a good choice for a small organization that cannot afford a big-budget commercial NMS. It is also sufficiently comprehensive and scalable that even large organizations use it. The Nagios Web site claims that it can handle networks of over 100,000 NEs. Given the high overhead of the external polling mechanism, the server hardware would be very expensive if all the 100,000 NEs are polled.

Big Brother. A well-known low-end system management product is Big Brother [MacGuire S]. Although it has very serious limitations in terms of both platforms and functionality, it may be adequate for small and medium-sized company networks.

Big Brother is an example of software that can be implemented with relative ease to manage system resources and is Web based. A central server on a management workstation runs on a UNIX platform and clients on managed objects. Big Brother is written in C and UNIX shell scripts and hence can be run on multiple platforms. The central management station presents the status of all systems and applications being monitored in a matrix of colored cells, each color designating a particular status. It supports pager and email functions that report the occurrence of alarms. Software can be downloaded and modified to meet local requirements for the operations, services, and applications to be monitored.

Big Brother performs the dual function of polling clients and listening to periodic status reports from clients that are UNIX based. Polling checks the network connectivity to any system. Client software periodically wakes up, monitors the system, transmits information to the central server, and then goes back to sleep. Textual details can be obtained on the exact nature of a problem. The systems are grouped for ease of administration.

9.5.3 Enterprise Management

We will next describe the two commercially available integrated solutions to system and network management. The two solutions are offered by two vendors, Computer Associates and Tivoli, the latter has been acquired by IBM. Both partners with several NMS vendors provide integrated solutions.

Computer Associates Unicenter TNG. The CA Unicenter TNG framework [CA] provides infrastructure to support integrated distributed enterprise management. It is based on a client/server architecture having an agent in each host and a centralized workstation. CA provides Unicenter TNG agents that can run on a large number of platforms; and the list continues to increase as the customer base diversifies. Besides TCP/IP and SNMP, the TNG framework supports numerous other network protocols, accommodating a varied enterprise environment.

CA describes Unicenter TNG as a framework comprising three components: Real World Interface, object repository, and distributed services. The Real World Interface presents a visual depiction of managed objects from different user perspectives. Both two-dimensional and three-dimensional presentations are available. The object repository is a management information database that includes multiplicity of data, such as managed objects, metadata class definitions, business process views, policies, topology, and status information. The distributed services link elements at the communications level.

Because of the TNG agents running in the hosts, during autodiscovery the system discovers not just host identifications, but also details of the processor, disk, and other components of the system. The discovered components are presented in a Real World Interface that presents a unified GUI at the higher levels. An object repository stores all the autodiscovered information and any other management information needed to support the Real World Interface. System and network management views are extended to business process views, whereby objects related to an administrative group or functional group can be presented. This approach makes operations easier for human personnel because they do not have to know the technical details behind the operations they are performing.

Event management in the TNG framework includes a rule-based paradigm that correlates events across platforms and presents the resultant alarms at the central console. These events include standard SNMP traps. Standard drilling through layers to detect the lowest level component failure is built in as desktop support.

An additional feature of the TNG framework is calendar management, which provides a shared calendar so that all personnel can view each other's activities. The calendar handles both one-time and periodic activities. Operations such as triggering an alarm pager or email can be programmed according to weekly and weekend shift schedules.

Some of the other notable features of the TNG framework are backup and disaster recovery, customized and canned report generation, and virus detection—all of which can be programmed as part of calendar management activities. In addition to these standard features, numerous optional modules, such as advanced help desk management, Web server management, and software delivery are available.

Tivoli Enterprise Manager. Tivoli's management framework, originally named the Tivoli TME 10 framework, provides system and network management and is in the same class as the Unicenter TNG framework. Tivoli has changed the TME 10 from a two-tier, client/server architecture to a three-tier architecture and has renamed it Tivoli Enterprise Manager. Tivoli claims that the new architecture has increased the capability of handling from 300 to 10,000 managed nodes. The extended three-tier architecture also has a gateway as a middle layer that is designed to handle as many as 2,000 agents. The agent module has been redesigned to consume fewer resources.

Tivoli merged with IBM, allowing it to integrate the features of its systems with IBM's NetView NMS. NetView performs the network management function, and the complementary features of the Tivoli management applications perform the system management functions. The platform is object oriented and uses the standard CORBA-compliant object model for use with diverse and distributed platforms. This feature is somewhat similar to the Sun Enterprise Manager [Sun Enterprise], which also uses the CORBA-compliant OSI object model and standard CMIP protocol. In the management framework, Tivoli management agents reside in the managed host applications. Although the TME Enterprise system does not use SNMP as the management protocol, NetView handles SNMP traps.

The Tivoli Enterprise Manager monitors network, systems, and applications in a distributed architecture, as in most other systems. Event management has a built-in rule-based engine that correlates events and diagnoses problems. It further has an automated or operator-initiated response mechanism

correct problems wherever possible, using a decision support system. Service desk technology is integrated with network and system management technology in the service-level management framework.

Tivoli service management comprises problem management, asset management, and change management. The problem management module tracks customer requests, complaints, and problems. The asset management module handles inventory management. The change management module incorporates and manages business change processes.

The Tivoli Enterprise Manager framework contains an applications manager module (global enterprise manager) that coordinates business applications residing in diverse multiple platforms. An API is provided to permit third-party vendors to integrate their application software into the Tivoli Enterprise Manager framework. The applications manager also measures the response and throughput performance of applications.

The security management module provides encryption and decryption capabilities (optional). A software distribution module automates software distribution and updates. Operations can be scheduled by the workload scheduler module.

9.5.4 Telecommunications Management Systems

Unlike an enterprise network, a telecom network provides commercial services to subscribers. The operator has legal responsibilities to the subscribers. Telecom operators are usually subject to stringent regulations.

Telecommunications network management includes monitoring and managing the network with certain distinctive features. Apart from supporting standard management features such as low level, as well as consolidated, fault and performance management, there are certain features that are specific to telecom:

1. A typical telecommunications network often includes equipment acquired over several years, and a telecommunications NMS must be capable of managing these heterogeneous systems from a single point.
2. Most telecom devices that implement the ITU-T standard protocols (such as CMIP/CMISE) differ in details and this makes development of the management application more complex.
3. Some devices have only an EMS, which often has a proprietary interface, and the NMS must communicate with the EMS rather than with the device agent directly. Thus, often the NMS manages some NEs via an EMS and some directly.
4. Typical telecom networks include multivendor multitechnology equipment, supporting diverse protocols. SDH and related transport technologies (SONET, DWDM, etc.) continue as the de facto technology for delivering reliable and scalable andwidth services over the last decade, and service providers have large amounts of deployed fiber that has been laid out over a period of time.
5. Provisioning of bandwidth is a common and important requirement in telecom networks. Provisioning bandwidth in optical networks in an optimal manner without causing fragmentation of the available bandwidth is a challenge. For example, if there is a requirement to provision an E1 circuit (2 Mbps), it would not be desirable to "break" an STM1 link (64 Mbps) to do the provisioning.
6. A related requirement is the maintenance of the latest inventory in the database so that when there is an incoming bandwidth-provisioning request, reliable and fresh information about the availability of bandwidth can be accessed. Usually, this information is maintained and updated manually. However, a telecom network is a dynamically changing entity, so automatic discovery

with periodic rediscovery to synchronize the inventory database with the actual network is becoming a critical requirement for efficient and optimal management of the network. While autodiscovery of IP networks using ping and/or SNMP is a standard feature in data NMS, the discovery of circuits in telecom networks is a far less straightforward issue.

7. A typical telecom provider assures customers of quality of service via service level agreements (SLAs). In a telecom environment this includes parameters such as call completion rate for voice calls, signal strength on wireless links, call completion for data calls, etc. The management system must provide the support for SLA management.
8. RCA to diagnose the actual source of a problem is another important feature to be supported.
9. Networks tend to be large, often with hundreds of thousands of NEs, and have a wide geographic spread. The telecom operator may have a hierarchical organization with different administrators responsible for different regions of the network. The NMS architecture must be scalable and should support the hierarchy of the operator.

In this section we describe CygNet, a commercial telecom NMS deployed at leading telecom service providers in India. CygNet is the flagship product of NMSWorks Software Pvt. Ltd., a technology company that is a part of the TeNeT (Telecommunications and Networks) group of the Indian Institute of Technology Madras, India.

CygNet NMS is a multivendor multitechnology management system that has been designed to meet the needs of telecommunication management. CygNet architecture accommodates most of the needs of a telecommunications management system.

Architecture of the CygNet Core. Figure 9.39 depicts the architecture of the core CygNet NMS. We describe the major components below.

Managed Element Modeling, Storage, and Depiction The elements server is responsible for this function. Network elements that are managed by the CygNet NMS are stored in the topology (topo) database. Elements to be managed by the CygNet NMS can either be added manually or by automatic discovery. A discovery configurator allows the operator to specify a range of IP addresses for discovering

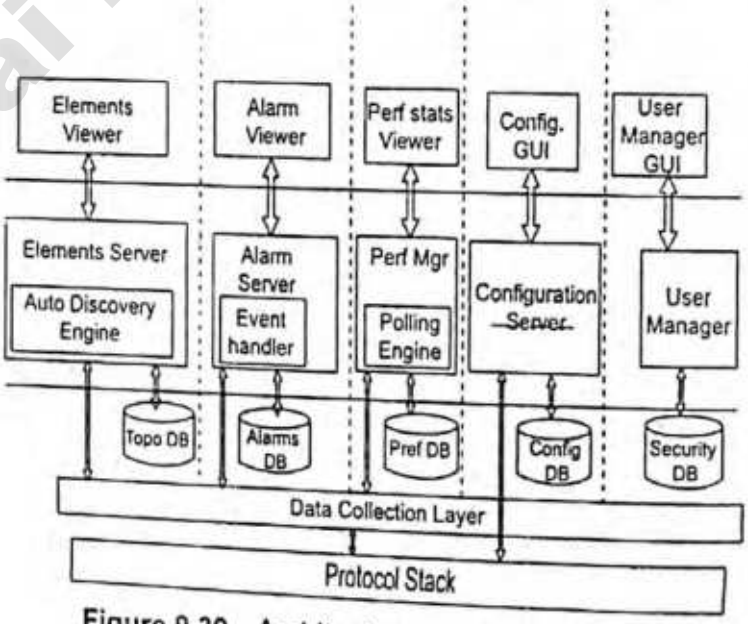


Figure 9.39 Architecture of the CygNet Core

th identifier (VPI) comprises virtual circuit identifiers (VCIs). Thus, establishing the route from St
n A to Station Z in our example is to do a look-up of the VPI–VCI tables. The price that we pay f
opting this mechanism is that some VCIs may remain idle during non-busy traffic period, and th
ste the bandwidth. However, this wasted bandwidth is a lot less than that for dedicated physical link
a circuit-switched transmission mode.

The VP–VC can be established on a per session basis or on a permanent basis between a pair of en
ations that carry large volumes of traffic. In the former case, the circuit is established as and whe
eeded and torn down after the session is over. This is called the switched virtual circuit (SVC). Whe
e connection is established for long periods of time and not switched between sessions, a permanen
irtual circuit (PVC) is established.

12.2.2 ATM Packet Size

ATM packets are of fixed size, each 53 bytes long. A fixed-size packet was chosen so that fast and ef
icient switches can be built. Many switches can operate in parallel if they all perform switching on th
ame-size packets.

The ATM packet size of 53 bytes has a header of 5 bytes and a payload of 48 bytes. This size wa
arrived at by optimizing between two factors. The packet size should be as small as possible to reduc
the delay in switching and packetization. However, it should be large enough to reduce the overhead o
the header relative to the payload.

12.2.3 Integrated Service

The main challenge in integrating the three services is to meet the different requirements of each. Voic
and video traffic require low tolerance on variations in delay and low end-to-end (roundtrip) delay
for good interactive communication. Once voice data are lost or delayed, real-time communication i
garbled and we cannot reproduce it. Thus, it has to be given the highest priority of service in transmis
sion. This is true with the voice portion of the information in video transmission. Further, the voice an
video have to be synchronized. Otherwise, it will be like watching a movie with the conversation lagging
behind the mouth motion due to wrong threading of the film. Pure video without sound can have les
priority than audio.

Data traffic can have a much higher tolerance on latency. It is primarily a store and forward technol
ogy and the traffic itself is inherently bursty in nature. However, data speed is important for large da
transmission applications, although it has the lowest priority in transmission.

It is possible to set the priority in ATM switches by assigning priority to the different services. This i
accomplished by guaranteeing a quality of service (QoS) for each accepted call setup. A traffic descrip
tor is specified by the user of the service; and the system ensures that the service requested could be m
by the virtual circuit that is set up.

There are four main classes of traffic defined to implement quality of service. They are the consta
bit rate (CBR), the real-time variable bit rate (VBR-rt), the non-real-time variable bit rate (VBR-nrt
and the available bit rate (ABR). Voice is assigned CBR. Streaming video such as real-time video o
the Internet is assigned VBR-rt. The VBR-nrt is applicable to transmission of still images. The IP dat
traffic gets the lowest bandwidth priority with ABR.

There are two markets for ATM switches using ATM technology, public and private. A public network i
the network that is established by the service providers. A private network is primarily a campus networ
Network management clearly distinguishes between these two markets, as we shall see in Section 12.3.

12.2.4 WAN/SONET

Although analog high-frequency multiplexing is still in vogue for WAN transportation in legacy systems, digital transmission is the predominant mode of transportation. The basic voice band, 0–4 kHz, is converted to 64 Kbps digital signal universally. However, multiplexing hierarchy of the basic signal has evolved differently in North America, Europe, and Japan. For example, T1 transmission carrier shown in Figure 2.28(a) has a data rate of 1.544 Mbps carrying 24 voice channels. Equivalent of this in Europe is the E1 transmission at a data rate of 2.048 Mbps carrying 30 channels. Thus, whenever digital transmission happens across the "pond" between Europe and North America, there is an expensive conversion involved between the two types of systems.

The digital hierarchy has been brought into synchronization across the world using 155.52 Mbps as the basic data rate in carrier technology using fiber optics. However, the names are different in Europe and North America. In Europe, it is called SDH and in North America SONET. The units of SDH are Synchronous Transport Signal (STS-n) where n is the hierarchical level. The optical carrier starts with the unit of OC-1 (Optical carrier-level 1), which is 51.84 Mbps and thus the basic SONET is level OC-3.

12.2.5 ATM LAN Emulation

It was once considered possible that ATM would be at all desktop workstations. However, this has not been the case and IP over Ethernet has become the most used LAN.

The services provided by ATM differ from conventional LAN in three ways. First, ATM is connection oriented. Second, ATM makes one-to-one connection between pairs of workstations in contrast to the broadcast and multicast mode in conventional LAN. Third, a LAN MAC address is dedicated to the physical network interface card and is independent of network topology. The 20-byte ATM address is not.

In order to use ATM in the current LAN environment, it has to fit into the current TCP/IP LAN environment. Because of the basic differences mentioned in the previous paragraph, although the ATM Forum has developed ATM specifications for LAN emulation (LE or LANE) that emulate services of the current LAN network across an ATM network, it has been discontinued. Hence, we will not discuss this any further.

12.3 ATM NETWORK MANAGEMENT

Broadband network management consists of managing the WAN using ATM technology, as well as access networks from the central office to the home. We will discuss the former in this section. We will discuss access technology management in the next chapter.

WAN facilities are provided by public service providers, who perform the following management functions: operation, administration, maintenance, and provisioning (OAMP). Typically, a large enterprise or corporation services its private network. However, they too use the public service providers' facilities to transport information over a long distance. This is referred to as public network. ATM networks are classified as private and public networks, as shown in Figure 12.3. The standards for the management of each and the interactions between them have been addressed by the ATM Forum, which is an international organization accelerating cooperation on ATM technology. The user interface to the private network is the private user-network interface (UNI), and the interface to the public network is the Public UNI.

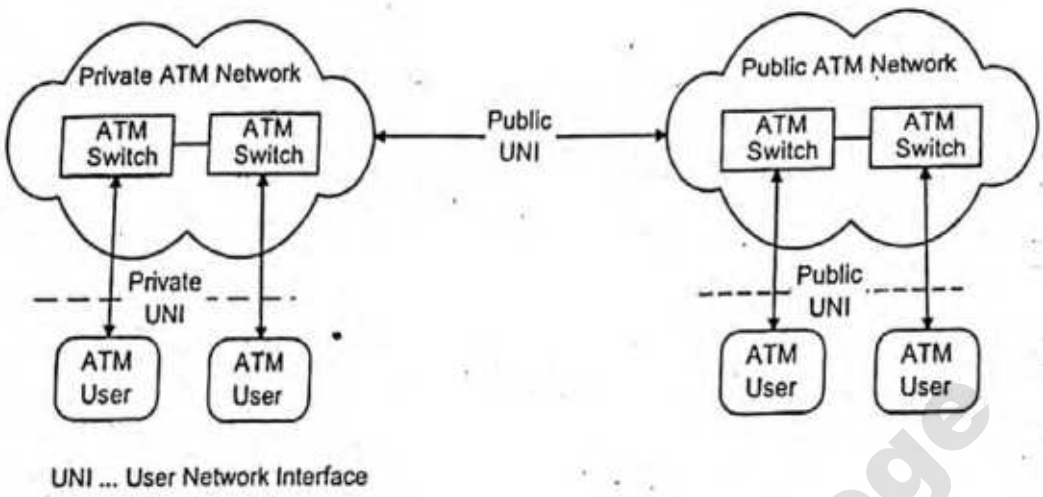


Figure 12.3 Private and Public ATM Network User Network Interfaces

12.3.1 ATM Network Reference Model

The ATM Forum has defined a management interface architecture, ATM network reference model, a shown in Figure 12.4. Private networks are managed by private network managers or private networ management systems (NMSs). Public network managers or public network management systems man age the public networks. To distinguish between a human manager and the management system, w will refer to the network manager in the context of a system as the NMS, unless explicitly stated. There are five interfaces between systems and networks. M1 and M2 are, respectively, the interfaces between private NMS and either end user or private network. The end user can be a workstation, ATM switch, o any ATM device. A private ATM network is an enterprise network.

A private NMS can access its own network-related information in a public network via an M3 interface to the public NMS. The public NMS, which manages the public network, responds to the private NMS via the M3 interface with the relevant information or takes the appropriate action requested.

M4 is the interface between a public NMS and a public network. M5 is the interface between NMSs of two service providers. The ATM Forum has not yet specified this interface.

12.3.2 Integrated Local Management Interface

Beside the M-interfaces, Figure 12.4 also shows interfaces between an ATM end user or device and an ATM network, as well as interfaces between ATM networks. These are distinct from the M-interfaces between NMSs and networks or end users. While the M-interfaces provide a top-down management view of network or device, the ATM Forum defines the ATM link-specific view of configuration and fault parameters across a UNI. These are the UNI interfaces presented in Figures 12.3 and 12.4. The specifications for these are contained in the integrated local management interface (ILMI), which we will discuss further in Section 12.3.4.

The "I" in ILMI originally stood for "Interim," not "Integrated." See af.ilmi.0065.000. Its specifications were supposed to have been replaced by IETF specifications. However, it turned out that some were and others were not. Hence, "I" in ILMI now designates Integrated. The ILMI fits into the overall model for an ATM device, as shown in Figure 12.5. The ATM management information is communicated across the UNI or the network-to-network interface (NNI). These interfaces are with ATM devices (end-systems, switches, etc.) that belong to either a private or a public network. Any interface with the

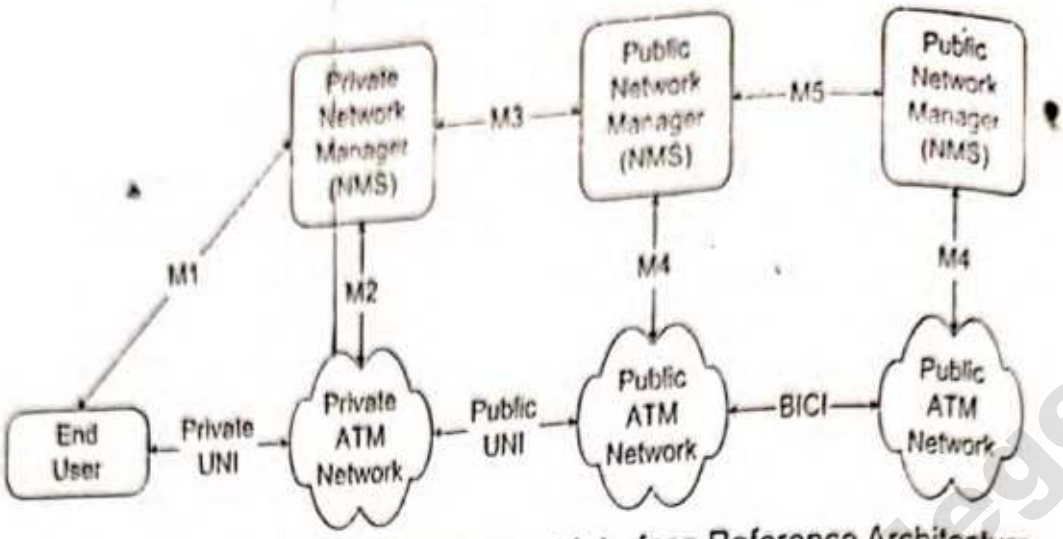


Figure 12.4 ATM Forum Management Interface Reference Architecture

public ATM network is a public UNI or a public NNI. Any interface with a private ATM network is a private UNI or a private NNI. The devices communicate across UNI and NNI via an ATM interface management entity (IME) module in the entity. There are three versions of IME—user, network, and system based on where it is used.

Figure 12.5 shows the various physical connections, virtual connections, and the ILMI communication links between the devices and the networks. The ILMI communication occurs over both physical and virtual links using SNMP or AAL5 protocols. We will discuss the MIB related to management in the following sections.

Two public carrier networks interface with each other via a broadband intercarrier interface (BICI), as shown in Figure 12.4. BICI is also known as network-to-network interface (NNI).

12.3.3 ATM Management Information Base

The MIB and the structure of management of information that are required for the management of the ATM network are specified between two sets of documents defined by IETF and the ATM Forum. The global view of the Internet MIB tree associated with the ATM is presented in Figure 12.6. The two major branches are *mib-2* and *atmForum* (under enterprises). The structure of management information is defined using the ASN.1 syntax. The MIB associated with the ATM is primarily concerned with the ATM sublayer parameters. The parameters associated with higher layers are handled using the standard MIB discussed earlier in Chapter 4. The documents that address the various groups are listed in Table 12.3.

There are five nodes shown under mib-2 in Figure 12.6. We described system and interfaces groups in Chapter 4. the interfaces group has evolved to handle the sublayers, such as ATM; the details are described in RFC 2863. The transmission group contains subgroups for each medium of transmission. The ATM objects, as defined in the *atmMIBObjects* group under *atmMIB*, are specified in RFC 1695.

The *atmForum* group is subnode 353 under the enterprises node. The *atmForum* group contains five subgroups, as shown in Figure 12.6: *atmForumAdmin*, *atmForumUni*, *atmUniDxi*, *atmLanEmulation* and *atmForumNetworkManagement*. The ATM administrative (*atmForumAdmin*) and ATM UNI (*atmForumUni*) groups are defined in the integrated local management interface (ILMI) specification, af-ilmi-0065.000. The ATM DXI (*atmUniDxi*) group is the Data Exchange Interface and is discussed in Section 12.3.9. It is the ATM interface between DTE and DCE and is described in af-dxi-0014.000. The MIB for M4 interface (*atmForumNetworkManagement*) is covered in af-nm-0095.001.

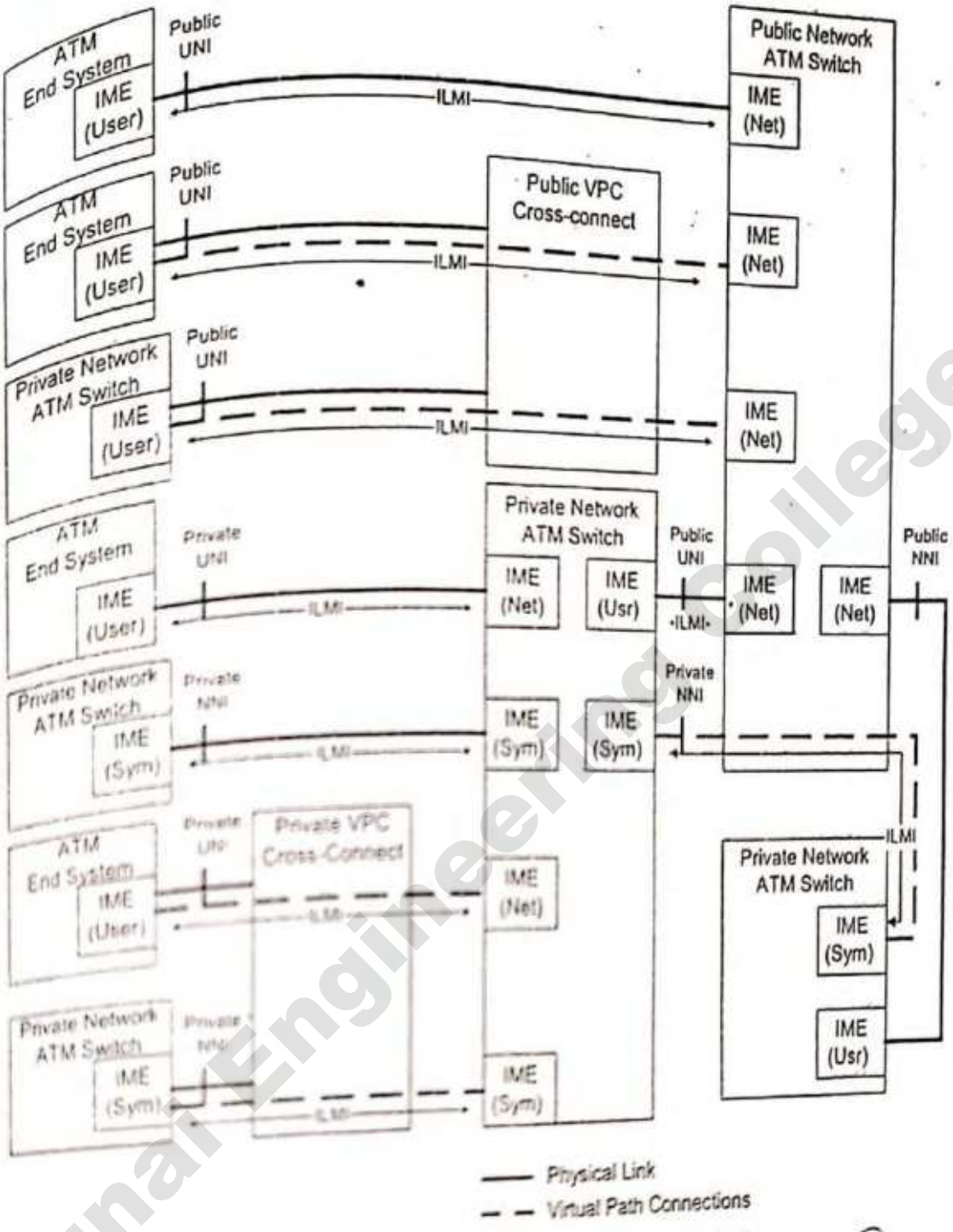


Figure 12.5 Definitions and Context of ILMI

12.3.4 Role of SNMP and ILMI in ATM Management

Although ILMI was conceived as interim specifications, it has become permanent. The ATM network management uses both SNMP MIB and ATM Forum MIB. Figures 12.7 and 12.8 (af-ilmi-0065.000 and Section 4 of UNI) conceptually present the role of the two network management protocols. Figure 12.7 presents the M1 interface. An SNMP agent is shown embedded in an ATM device, and the NMS communicates with it using SNMP protocol and IETF MIB modules. RFC 2863 specifies the interface parameters and types, including the additional tables to manage the ATM sublayer. RFC 1695 specifies the ATM objects. The transport MIB module is dependent on the transmission medium.

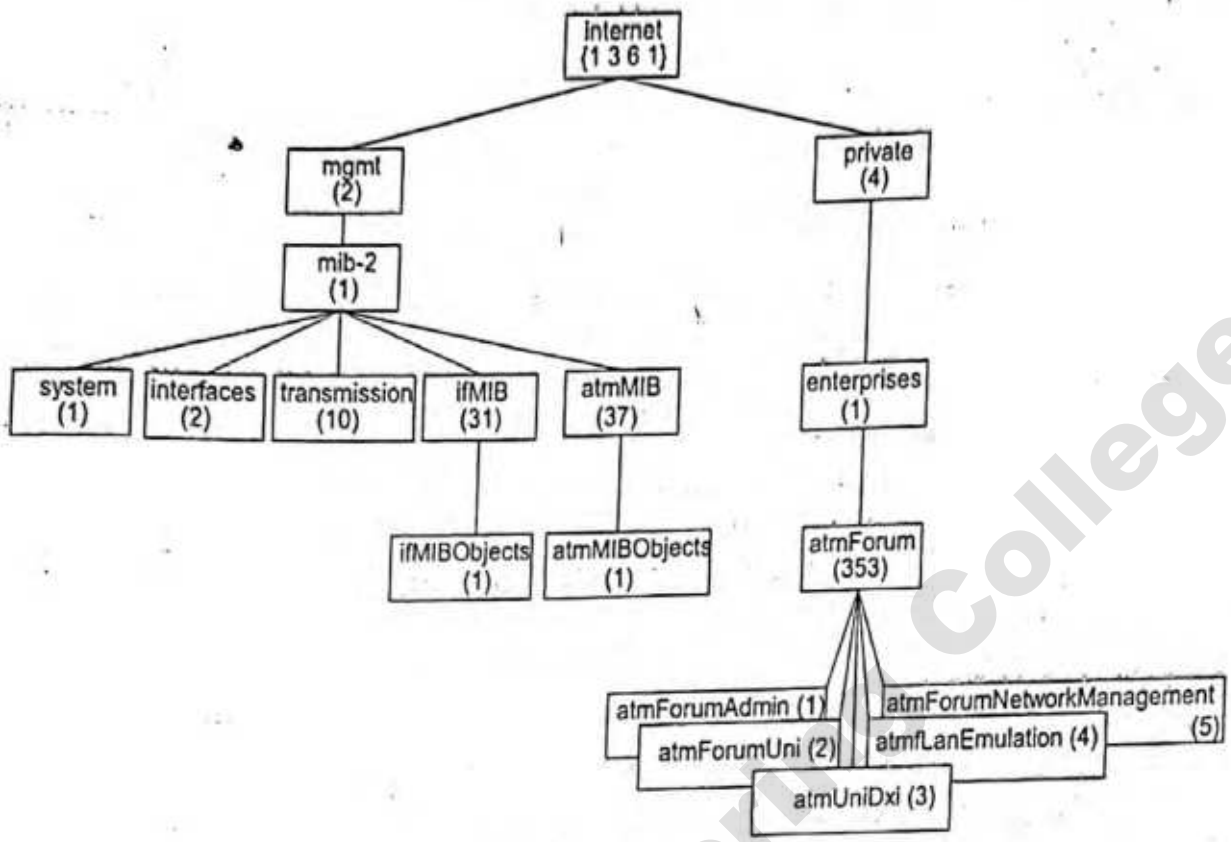


Figure 12.6 Internet ATM MIB

Table 12.3 Internet ATM MIB Groups and Documents

ENTITY	OID	DESCRIPTION	DOCUMENT
system	mib-2 1	System	RFC 1213
interfaces	mib-2 2	Interfaces (modified)	RFC 1213
RFC 1573			
ifMIB	mib-2 31	Interface types	RFC 1573
transmission	mib-2 10	Transmission	RFC 1213
ds1	transmission 18	DS1 carrier objects	RFC 1406
ds3	transmission 30	DS3/E# interface objects	RFC 1407
sonetMIB	transmission 39	SONET MIB	RFC 1595
atmMIB	mib-2 37	ATM objects	RFC 1695
atmForum	enterprises 353	ATM Forum MIB/M3 specification	af-nm-0019.000
		M4 interface	af-nm-0020.000
			af-nm-0020.001
		CMIP specification for M4 interface	af-nm-0027.000
		M4 network-view interface	af-nm-0058.000
		AAL management for the M4 NE View	af-nm-0071.000
		Circuit emulation service internet-working requirements, logical and CMIP MIB	af-nm-0072.000
		M4 network-view CMIP MIB Spec v1.0	af-nm-0073.000
		M4 network-view requirements and logical MIB addendum	af-nm-0774-000

Table 12.3 (continued)

ENTITY	OID	DESCRIPTION	DOCUMENT
atmForumAdmin	atmForum 1	ATM administrative	af-ilmi-0065.000
atmForumUni	atmForum 2	ATM user network interface	af-ilmi-0065.000
atmUniDxi	atmForum 3	Data exchange interface (DXI) specification	af-dxi-0014.000
		Multiprotocol over ATM	af-mpoa-0087.000
		Multiprotocol over ATM Version 1.0 MIB	af-mpoa-0092.000

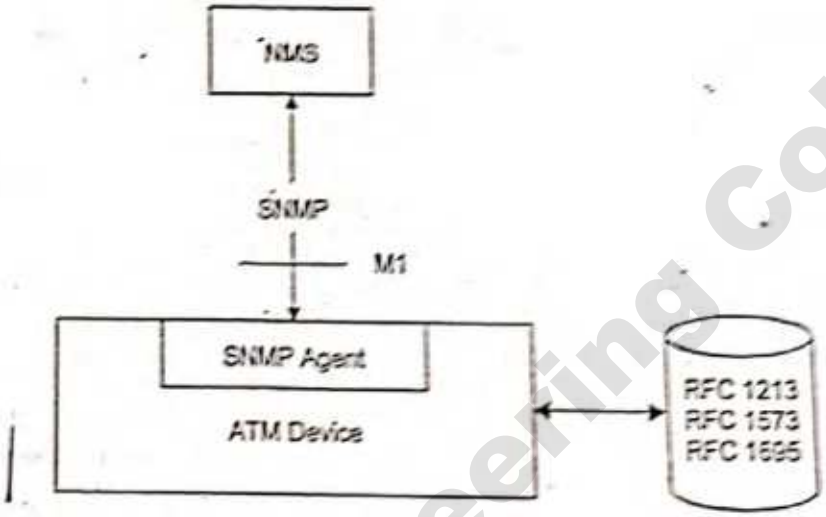


Figure 12.7 SNMP ATM Management (M1 Interface)

Figure 12.8, which shows the M2 interface, comprises the network of two ATM devices. The NMS manages the network with an interface to device A. The ILMI protocol is used for communication between the agent management entity (AME) in device A and the AME in device B. A proxy agent that resides in device A does the translation between ILMI MIB and SNMP MIB.

12.3.5 M1 Interface: Management of ATM Network Element

The M1 interface, as mentioned earlier, is between an SNMP management system and an SNMP agent in an ATM device, as shown in Figure 12.7. Four entities, *ifInNUcastPkts*, *ifOutNUcastPkts*, *ifOutQLen*, and *ifSpecific* have been deprecated. The interfaces (interfaces) and *ifMIB* (IF MIB) groups under the *mgmt* node are shown in Figure 12.9. Four tables have been added to handle sublayers. They are shown in Figure 12.9 under *ifMIBObjects*. Table 12.4 gives a brief description of the functions that each table performs.

Figure 12.10 shows the three transmission modes that are used for the ATM. They are DS1 (1.544-Mbps twisted-pair cable), DS3 (44.736-Mbps coaxial cable), and SONET (nx155.52-Mbps optical fiber). DS1 and DS3 are transmitted over T1 and T3 carriers, respectively. Only one of these MIBs

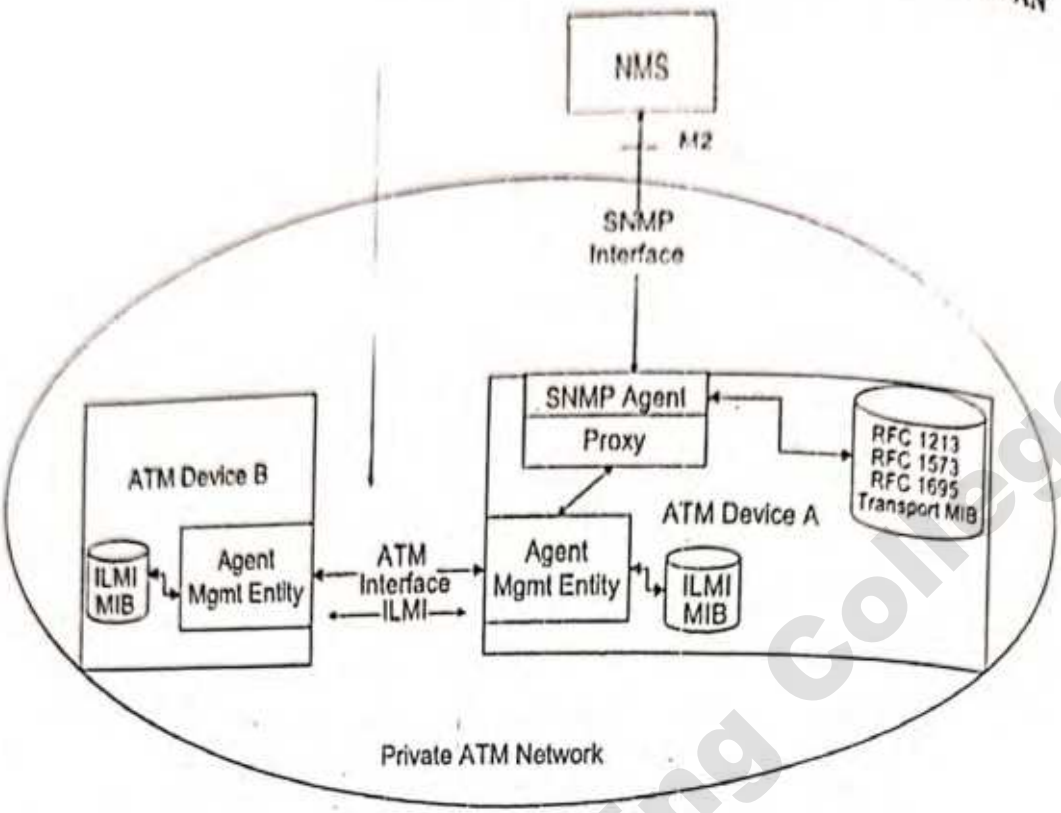


Figure 12.8 Role of SNMP and ILMI in ATM Management (M2 Interface)

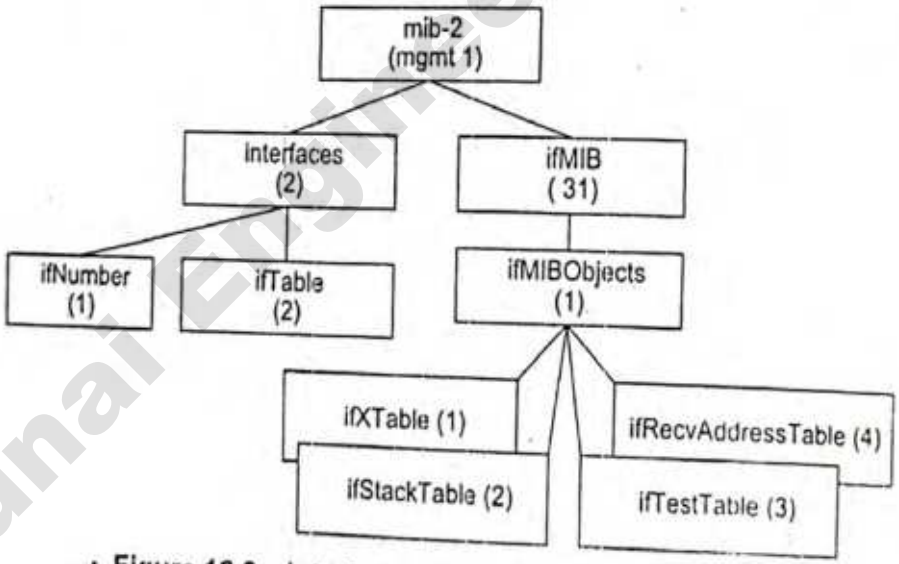


Figure 12.9 Interfaces Group Tables for Sublayers

Table 12.4 Interfaces Group Tables for Sublayers

ENTITY	OID	DESCRIPTION (BRIEF)
ifXTable	ifMIBObjects 1	Additional objects for the interface table
ifStackTable	ifMIBObjects 2	Information on relationship between sublayers
ifTestTable	ifMIBObjects 3	Tests that NMS instructs the agent to perform
ifRecvAddressTable	ifMIBObjects 4	Information on type of packets/frames accepted on an interface

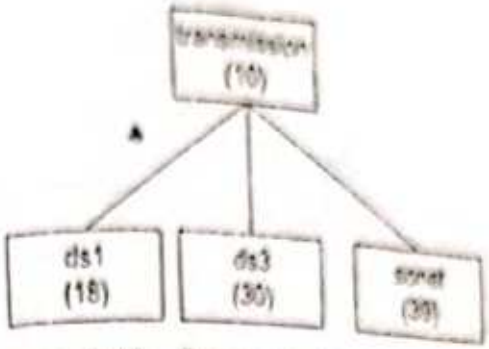


Figure 12.10 Transmission Groups for ATM

...to be implemented in the agent based on which transmission medium is used. RFCs dealing with ... transmission group MIB modules are listed in Table 12.3.

Figure 12.11 and Table 12.5 show the ATM MIB objects group. This group contains information to manage the ATM sublayer entities: traffic descriptors, DS3 physical-layer convergence parameters (PLCP), transmission convergence (TC) sublayer parameters, virtual path link/virtual channel link and their associated cross-connect tables, and performance parameters for AAL5 (ATM adaptation layer).

12.5.6 M2 Interface: Management of a Private Network

The M2 interface for ATM management is shown in Figure 12.8. The management information on ATM links between devices is gathered from ILMI MIB. The relative roles of each are shown in Figure 12.8. Detailed UNI and NNI for both private and public interfaces, specified in af-ilmi-0065.000, are shown in Figure 12.5 and were discussed in Section 12.3.2.

The ILMI specifications define the administrative and UNI groups of the ATM Forum MIB. The administrative group defines a general-purpose registry for locating ATM network services such as the ATM name answer server (ANS). Other subgroups under the administrative group have been deprecated and handled by IETF specifications.

Figure 12.12 and Table 12.6 show the ATM UNI MIB object group. They define the management objects associated with the ATM layer and the physical layer. The statistics group is deprecated. The parameters associated with virtual path/virtual connections as well as the adjustable bandwidth rate (ABR) and QoS are covered.

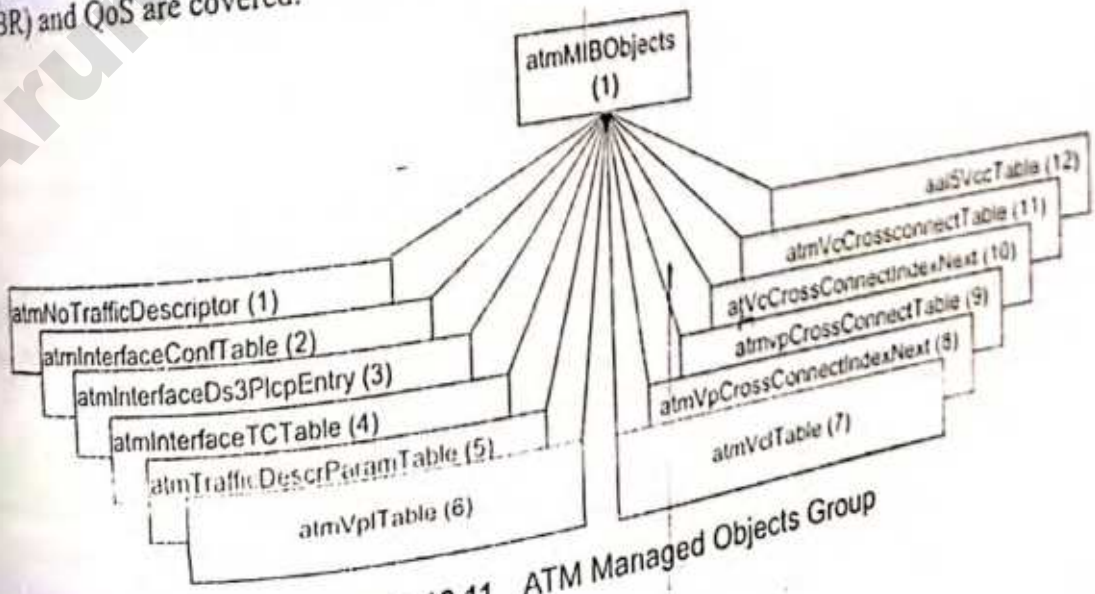


Figure 12.11 ATM Managed Objects Group

Table 12.5 ATM Managed Objects Group

ENTITY	OID	DESCRIPTION (BRIEF)
atmNoTrafficDescriptor	atmMIBObjects 1	ATM traffic descriptor type
atmInterfaceConfTable	atmMIBObjects 2	ATM local Interface configuration parameter table
atmInterfaceDs3PlcpEntry	atmMIBObjects 3	ATM Interface DS3 PLCP parameters and state variables table
atmInterfaceTCTable	atmMIBObjects 4	ATM TC sublayer configuration and state parameters table
atmTrafficDescrParamTable	atmMIBObjects 5	ATM traffic descriptor type and associated parameters
atmVplTable	atmMIBObjects 6	Virtual path link table
atmVclTable	atmMIBObjects 7	Virtual channel link table
atmVpCrossConnectNext	atmMIBObjects 8	Index for virtual path cross-connect table
atmVpCrossConnectTable	atmMIBObjects 9	Virtual path cross-connect table
atmVcCrossConnectNext	atmMIBObjects 10	Index for virtual channel cross-connect table
atmVcCrossConnectTable	atmMIBObjects 11	Virtual cross-connect table
aal5VccTable	atmMIBObjects 12	AAL VCC performance parameters table

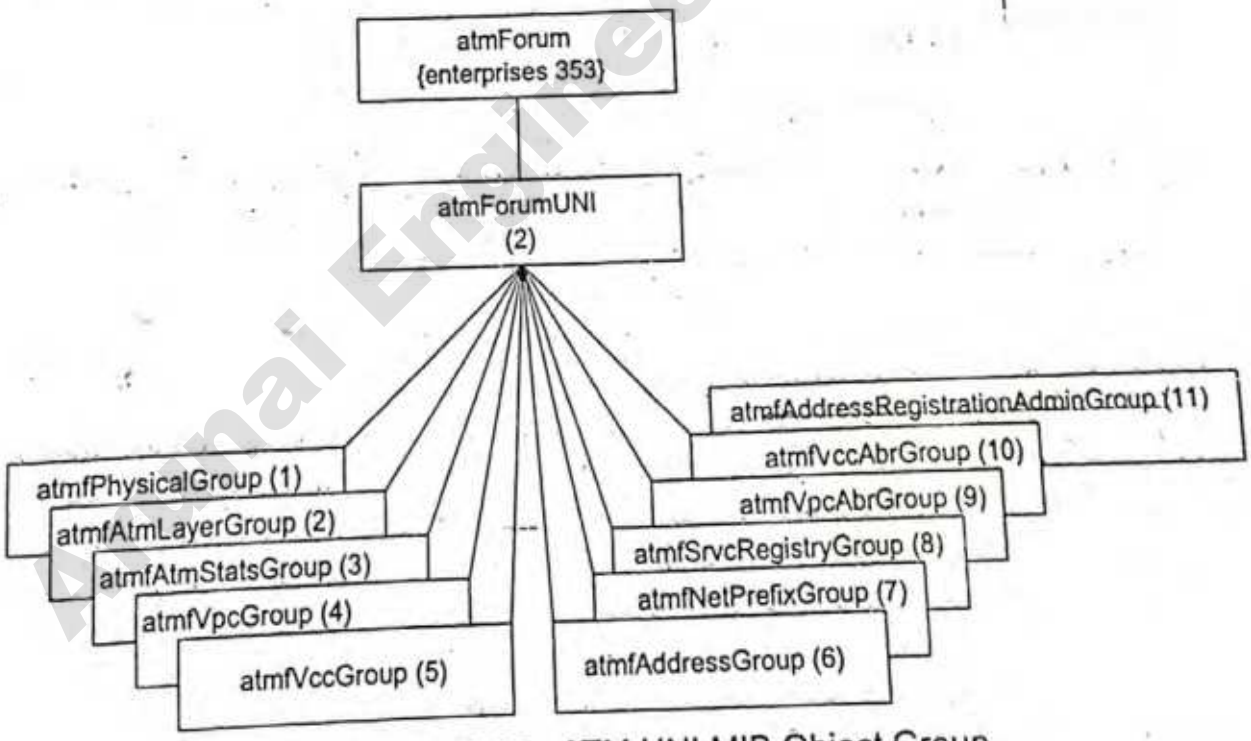


Figure 12.12 ATM UNI MIB Object Group

Table 12.6 ATM UNI MIB Object Group

ENTITY	OID	DESCRIPTION (BRIEF)
atmfPhysicalGroup	atmForumUni 1	Defines a table of physical-layer status and parameter information
atmfAtmLayerGroup	atmForumUni 2	Defines a table of ATM-layer status and parameter information
atmfAtmStatsGroup	atmForumUni 3	Deprecated
atmfVpcGroup	atmForumUni 4	Defines a table of status and parameter information on the virtual path connections
atmfVccGroup	atmForumUni 5	Defines a table of status and parameter information on the virtual channel connections
atmfAddressGroup	atmForumUni 6	Defines the network-side IME table containing the user-side ATM-layer addresses
atmfNetPrefixGroup	atmForumUni 7	Defines a user-side IME table of network prefixes
atmfSrvcRegistryGroup	atmForumUni 8	Defines the network-side IME table containing all services available to the user-side IME
atmfVpcAbrGroup	atmForumUni 9	Defines a table of operational parameters related to ABR virtual path connections
atmfVccAbrGroup	atmForumUni 10	Defines a table of operational parameters related to ABR virtual channel connections
AtmfAddressRegistrationAdminGroup	atmForumUni 11	

12.3.7 M3 Interface: Customer Network Management of a Public Network

M3 is the management interface between the private NMS and the public service provider NMS. It allows the customer to monitor and configure their portion of the public ATM network. The M3 interface specifications are defined in the ATM Forum document af-nm-0019.000. Networks show the typical configuration, how a customer would interact with the public service provider network via the carrier management system. There are two classes of M3 requirements in the figure—status and configuration monitoring (Class I) and virtual configuration control (Class II).

Class I requirements are those which a public network service provider offers to the customer. These include the customer performing monitoring and management of configuration, fault, and performance management of a specific customer's portion of a public ATM network. This service is offered only for PVC configuration. Examples of this service are (a) retrieving performance and configuration information for a UNI link and (b) public service NMS reporting an alarm or trap message to the user NMS on a UNI-link failure.

Class II service provides greater capability to the user. The user can request the service provider to add, delete, or change virtual connections between a pair of customer's UNIs. An example of this would be the customer wanting to establish a new virtual path or increase the number of virtual circuits in a given virtual path.

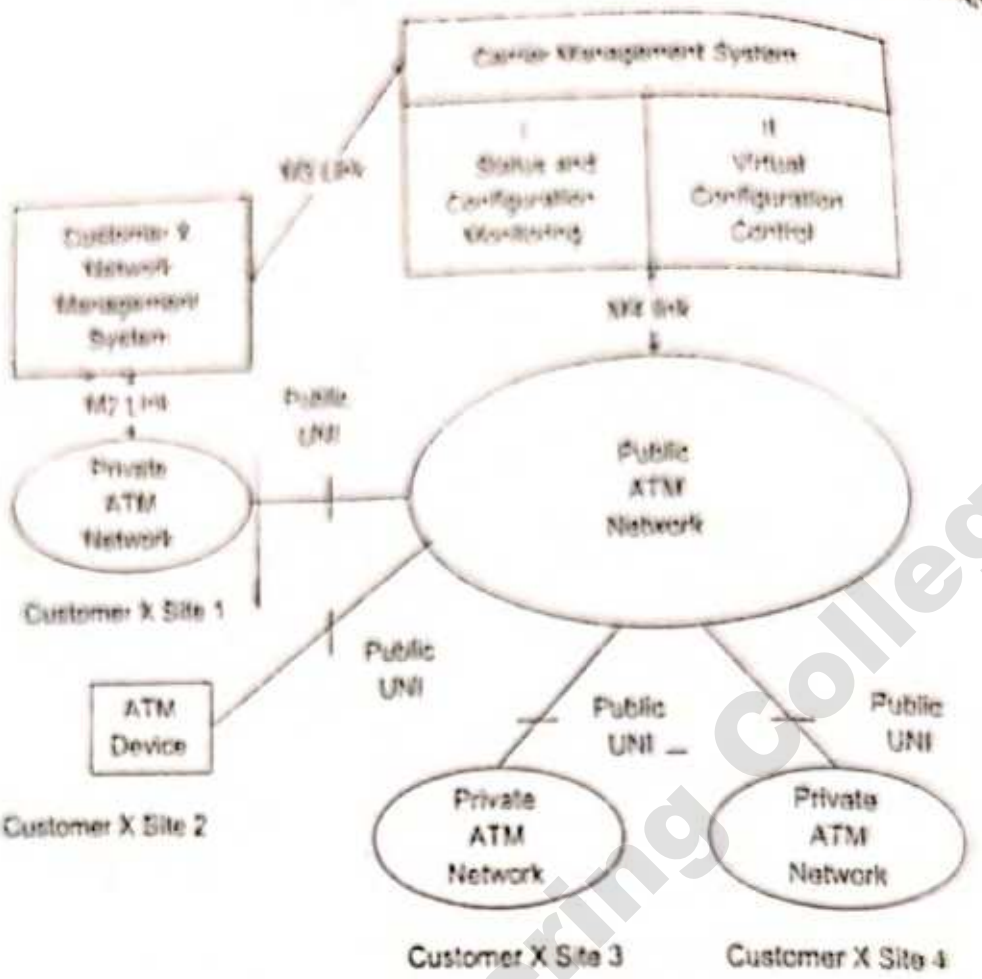


Figure 12.13 Customer Management of Private and Public Networks

A customer network management (CNM) agent residing in the private service provider's NMS provides the M3 service. As mentioned, the service is limited to the portion of the public service provider's network that the user's circuit traverses. If the user's circuit traverses multiple service providers, a separate interface with each provider is needed. The CNM sends requests to the carrier management system (see Figure 12.13), which acts as an agent to the CNM. The carrier management system then invokes the request on the network elements (NE) or other NMS and returns the responses to CNM.

The requirements for M3 and M4 are specified as mandatory or required, conditionally required, and optional. Class I requirements are mandatory, and Class II requirements are optional.

Class I Interface Management Functions. Table 12.7 presents M3 Class I requirements and the MIB groups that are used to obtain the information. The request has SNMP "read-only" capability. The public network service provider should give the CNM customer the ability to retrieve all the information listed in Table 12.7

Class II Interface Management Functions. M3 Class II functionality is divided into three groups so that the provider can implement one or more subgroups. They are (1) ATM-level subgroup, (2) VPC/VCC-level subgroup, and (3) traffic subgroup.

The ATM-level subgroup should provide the CNM the ability to modify the ATM Level Information Configuration Information.

The VPC/VCC-level subgroup provides the CNM the ability to modify:

1. Virtual path link configuration and status information.
2. Virtual channel link configuration and status information.

Table 12.7 M3 Class I Interface Requirements and MIB

General UNI protocol stack information	System group [RFC 1213], Interfaces group, including ifTable and ifStackTable [RFC 1213, 2863], SNMP group [RFC 1213]
ATM performance information on customer's UNI physical-layer performance and status information	ifTable [RFC 2863] All tables except dsx3ConfigTable [RFC 1407], all tables except dsx1ConfigTable [RFC 1406], all tables except the configuration tables and VT tables of SONET MIB [RFC 1595], atmInterfaceDs3PlcpTable/atmInterfaceTCTable of ATM MIB [RFC 1695]
ATM-level information configuration information Physical-layer configuration information	atmInterfaceConfTable of ATM MIB [RFC 1695] dsx3ConfigTable [RFC 1407] dsx1ConfigTable [RFC 1406] all configuration tables except the sonetVtConfigTable of SONET MIB [RFC 1595]
ATM-layer virtual path link configuration and status information	atmVplTable of ATM MIB [RFC 1695]
ATM-layer virtual channel link configuration and status information	atmVclTable of ATM MIB [RFC 1695]
ATM-layer virtual path connection configuration and status information	atmVpCrossConnectTable and atmVpCrossConnectIndexNext of ATM MIB [RFC 1695]
ATM-layer virtual channel connection configuration and status information	atmVcCrossConnectTable and atmVcCrossConnectIndexNext of ATM MIB [RFC 1695]
ATM-layer traffic characterization (traffic descriptors for customer's UNIs) information	atmTrafficDescrParamTable of ATM MIB [RFC 1695]
Event notifications on ATM link going up or down	warmStart, coldStart, linkUp, linkDown of SNMP group [RFC 1695]

3. Virtual path connection configuration and status information.
4. Virtual channel connection configuration and status information.

The traffic subgroup shall provide the CNM the ability to modify:

1. Traffic descriptors and information objects for virtual channel connections.
2. Traffic descriptors and information objects for virtual path connections.

Table 12.8 presents the M3 Class II requirements and the MIB objects in the ATM MIB group.

12.3.8 M4 Interface: Public Network Management

The management of public ATM network is primarily the responsibility of network service providers—carriers and Postal Telephone and Telegraph (PTT) companies. They have the challenge of not only managing the public network, but also keeping up with new technology. To help this process, ITU-T has

Table 12.8 M3 Class II Interface Requirements and MIB

ATM-level information configuration information	atmInterfaceConfTable in ATM MIB [RFC 1695]
Virtual path link configuration and status information	atmVplTable in ATM MIB [RFC 1695]
Virtual channel link configuration and status information	atmVclTable in ATM MIB [RFC 1695]
Virtual path connection configuration and status information	atmVpCrossConnectTable and atmVpCrossConnectIndexNext of ATM MIB [RFC 1695]
Virtual channel connection configuration and status information	atmVcCrossConnectTable and atmVcCrossConnectIndexNext of ATM MIB [RFC 1695]
Traffic descriptors and information objects for virtual path and channel connections	atmTrafficDescrParamTable in ATM MIB [RFC 1695]

defined M.3010, a five-layer model of operations, telecommunications management network (TMN), which we discussed in detail in Chapter 10. The relationship of ATM to TMN is shown in Figure 12.14. The top two layers, the business management layer and the service management layer, deal with the business and service aspects of TMN and are not addressed by the ATM Forum.

The element layer (EL) contains NEs. The NEs specific to ATM technology are components, such as ATM workstation, ATM switches, ATM transport devices (cross-connect systems and concentrators), etc. The element management layer (EML) manages NEs. The network management layer (NML) has the responsibility to manage the network either directly or via the EML.

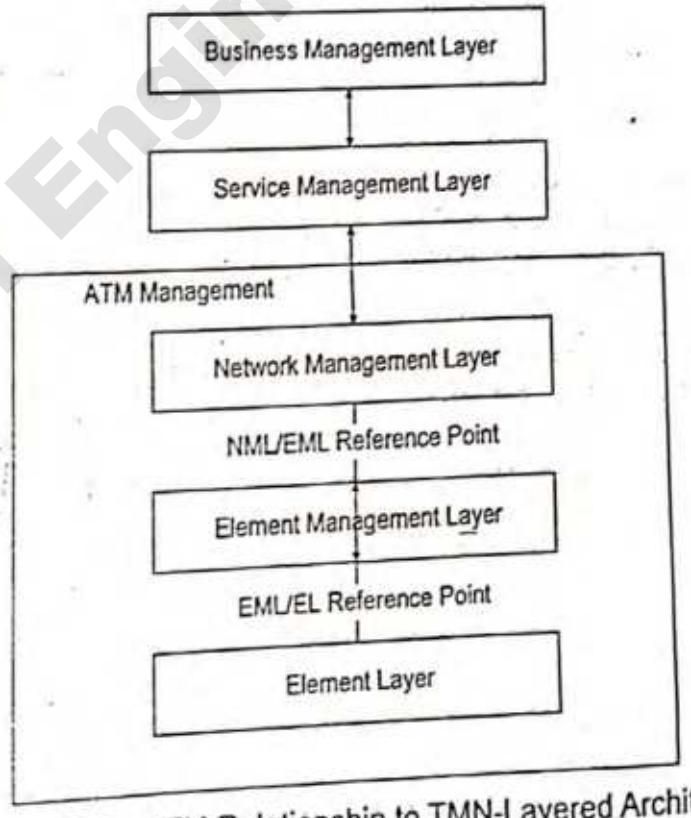


Figure 12.14 ATM Relationship to TMN-Layered Architecture

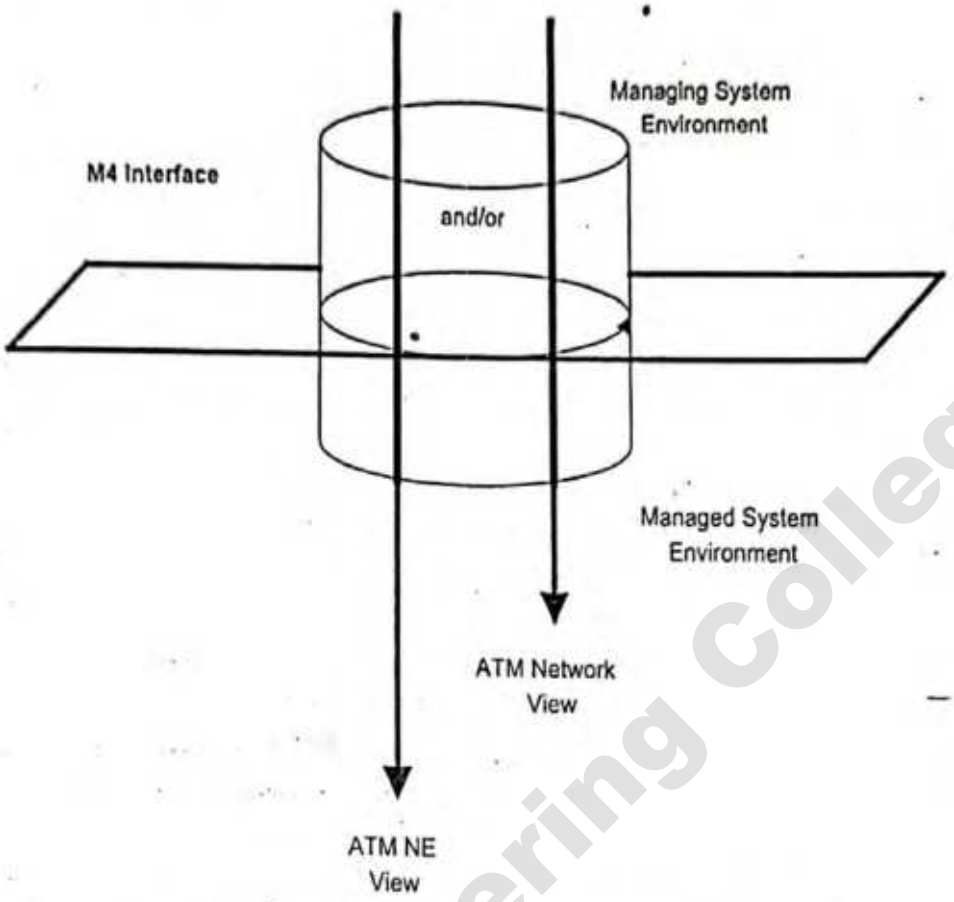


Figure 12.15 Dual Views of the M4 Interface

Figure 12.15 shows the dual view of M4 interface (af-nm-0058.000). Both views are present in the architecture across the M4 interface plane. It should be noted that this is a conceptual view, and the physical connections can be the same for both views.

In the NE-level management architecture, shown in Figure 12.16, the NMS environment, consisting of one or more NMSs, directly interfaces with the ATM NE and manages them. There is a single M4 interface between ATM NE and the NMS environment. The figure shows links between NE, but the NMS directly communicates with each ATM network element. In an actual implementation, it is likely that the NMS is interfacing with another NMS, which is managing the NE, but can still present a network view to the higher-level management system.

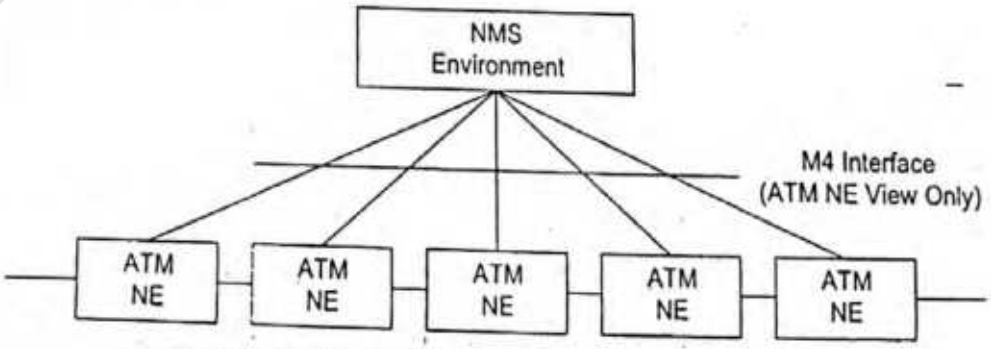


Figure 12.16 NE-View Management Architecture

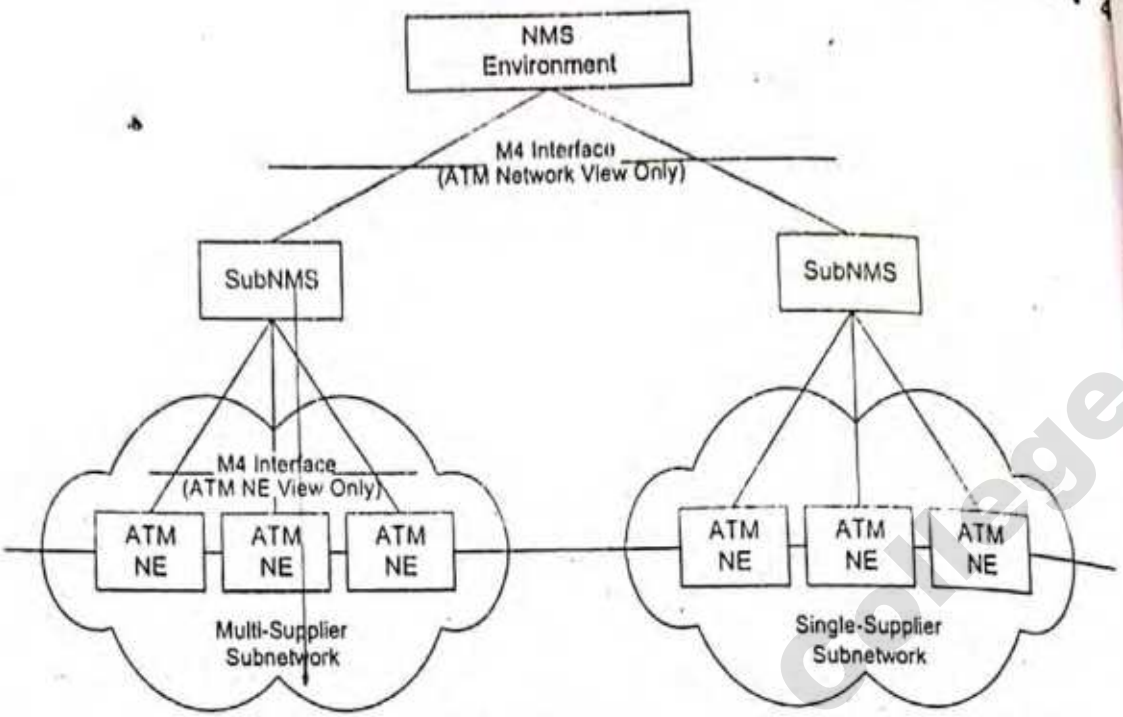


Figure 12.17 Example of Network-View Management Physical Configuration

Figure 12.17 presents an example of network-view management physical configuration. It consists of two ATM networks, one a single-supplier subnetwork and the other a multi-supplier subnetwork. Each subnetwork has its own subNMS managing its NE. In the single-supplier subnetwork shown on the right side, the subNMS has only an ATM NE view. In the multi-supplier subnetwork environment shown on the left side, the subNMS is presented only an ATM NE view, although it may actually be communicating to lower-level NMSs of each supplier. This is similar to the manager-of-manager architecture we discussed in Chapter 4. This is explicitly shown in the figure. Both subNMSs in Figure 12.16 present an ATM network view only to the NMS environment shown at the top. Thus, the top-level NMS sees only the network view of the subnetworks and not the NE view.

The NMS environment can manage both NE and networks, as shown in the architectural view in Figure 12.18. We can visualize such a hybrid situation in some remote ATM devices that do not need to be under a group's NMS, but are managed by an enterprise NMS.

M4 Network Element-View Requirements and Logical MIB. ATM Forum M4 network-element-view specifications support PVCs. Based on the OSI application model, the specifications are confined to configuration management, fault management, and performance management. Basic security features are also included to ensure the authorization and authentication of the user and the protection and privacy of data, while M3 interface responds to queries. The ATM security framework is similar to the security framework discussed for management in Chapter 7. It is covered in af-sec-0096-000. The network management considerations are presented in af-nm-0103.000, which includes security requirements and logical MIB.

The MIB specifications are specified in a logical format in af-nm-0020.000 and updated in af-nm-0020.001. They could be implemented using either CMIP or SNMP protocols. The CMIP specifications are presented on af-nm-0027.000, af-nm-0058.000, af-nm-0071.000, af-nm-0072.000, af-nm-0073.000 and af-nm-0074.000. The SNMP specifications are detailed in af-nm-test-0080.000 and af-nm-0095.001. We will now summarize the protocol-independent specifications of M4 interface and the logical MIB.

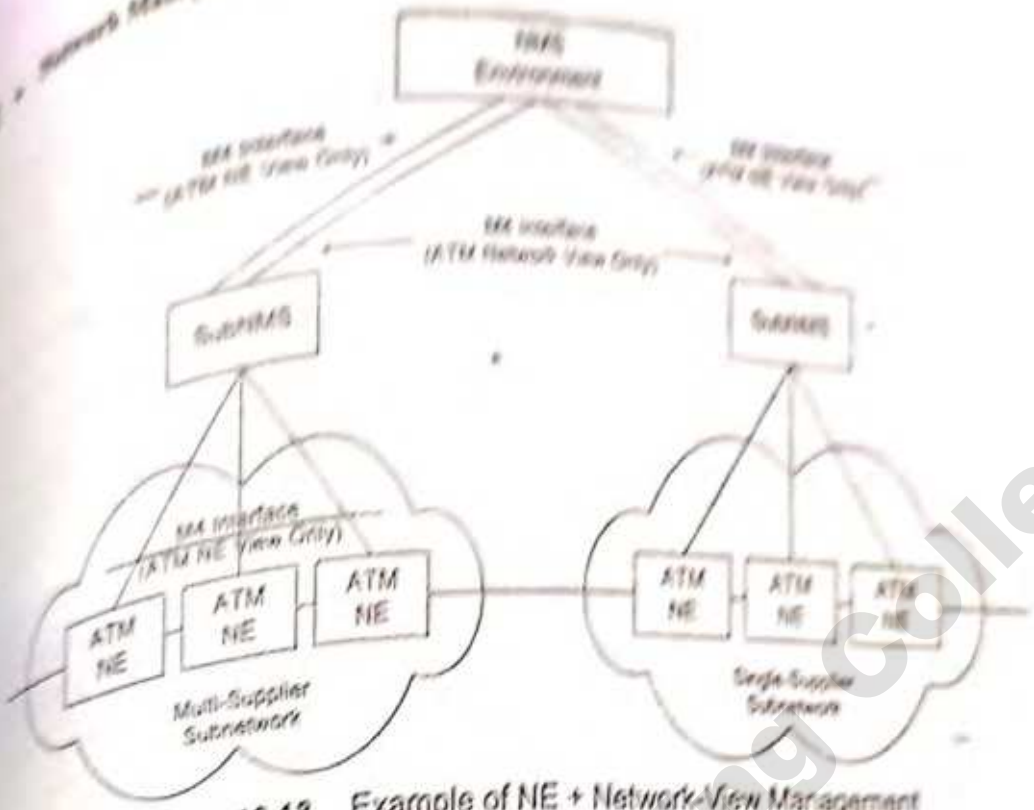


Figure 12.18 Example of NE + Network-View Management

Configuration Management. Configuration management provides the following list of functions to manage NEs:

1. ATM NE configuration identification and change reporting, which involves:
   (a) Operations performed over the craft interface
   (b) Human intervention (removal/insertion of equipment modules)
   (c) Customer control channels (e.g., ILMI)
   (d) Network failures
   (e) Protection switching events
   (f) Sub-ATM NE component initialization
   (g) Secondary effects of atomic operations performed by the management system
2. Configuration of UNIs, BICIs, and BISSIs.
3. Configuration of VPL/VCL termination points and cross-connections.
4. Configuration of VPC and VCC OAM segment end-points.
5. Event flow control—event forwarding discriminator function.

Fault Management. The following set of functions is specified to detect, isolate, and correct abnormal operation:

1. Notifying the NMS of a detected failure.
2. Logging failure reports.
3. Isolating faults via demand testing.

The specific functions are:

1. Failure reporting of the various alarms listed in Table 12.9. The generic troubles that cause the alarm are also listed in the table
2. Operations, administration, and maintenance (OAM) cell loopback testing.

Table 12.9 Generic Troubles in ATM NEs

ALARM CATEGORY	GENERIC TROUBLE
Communication alarms	Alarm indication signal (AIS)
	Loss of cell delineation (LCD)
	Loss of frame (LOF)
	Loss of pointer (LOP)
	Loss of signal (LOS)
	Payload type mismatch
	Transmission error
	Path trace mismatch
	Remote defect indication (RDI)
	Signal label mismatch
Equipment alarms	Back-plane failure
	Cell establishment error
	Congestion
	External interface device problem
	Line card problem
	Multiplexer problem
	Power problem
	Processor problem
	Protection path failure
	Receiver failure
	Replaceable unit missing
	Replaceable unit problem
	Replaceable unit type mismatch
	Timing problem
	Transmitter failure
	Trunk card problem
Processing error alarms	Storage capacity problem
	Memory mismatch
	Corrupt data
	Software environment problem
	Software download failure
	Version mismatch
Environmental alarms	Cooling fan failure
	Enclosure door open
	Fuse failure
	High temperature
General	Vendor specific

Performance Management. The functions of performance monitoring for an ATM network are:

1. Performance monitoring.
2. Traffic management.
3. UPC (user parameter control)/NPC (network parameter control) disagreement monitoring.
4. Performance management control.
5. Network data collection.

To accomplish these general functions, the following specific functions are specified:

1. Physical-layer performance monitoring.
2. ATM cell-level protocol monitoring.
3. UPC/NPC disagreement monitoring.

Network-View Requirements and Logical MIB. The M4 network view for the management of an ATM public network is concerned with NML information. It addresses the different perspectives of the service providers, each of whom need both network management and service management capabilities.

The ATM Forum document af-nm-0058.000 details the requirements for the ATM network management across the M4 network view interface. The associated MIB is specified in logical form and is not management protocol specific. The functional areas addressed in the specifications are:

1. Transport network configuration provisioning (including subnetwork provisioning and link provisioning).
2. Transport network connection management (including set up/reservation/ modification for subnetwork connection, link connection, trails, and segments).
3. Network fault management (including congestion monitoring and connection and segment monitoring).
4. Network security management.

Managed entities have not been defined in the MIB for meeting all the above requirements.

Transport Network Provisioning/Layered Network Provisioning. The transport network provisioning includes subnetwork provisioning of network nodes and links. Specifically:

1. Subnetwork provisioning: addition and monitoring information on addition, deletions, and changes in NEs and their configuration.
2. Link provisioning: set up, modify, and release subnetwork links.

Subnetwork Connection Management. The M4 network-view managed entities support subnetwork management of reservation and modification of subnetwork connections, link connections, trails, and segments. Specifically, this involves:

1. Point-to-point subnetwork connection: VP–VC subnetwork connection between pair of end points.
2. Multipoint subnetwork connection: Multipoint VP–VC connections between pair-wise end points.
3. Link connection set-up: VP–VC connections between subnetworks.
4. Segment setup: Set up and support VP–VC segment termination end points.
5. Trail setup: Support and set up trails containing information on subnetwork connections and links.

When a part of the ATM trail spans multiple administrative domains, each NMS is responsible for its domain setup and maintains its trails.

Connection Release. The connections release across the M4 interface involves the management of resources to be made available after use. Specifically, the network management should support

1. Release subnetwork connections and release resources of both point-to-point and multipoint connections.
2. Release link connections between subnetworks.

Subnetwork State Management. The NMS needs to be aware of the operational status of the subnetworks with regard to the network being ready to perform its intended functions, including link connections, trail operational changes, and network components.

Transport Network Fault Management. The M4 interface management is required to report network-view alarms and provide testing capability to isolate the problem. Specifically, this includes

1. Log network alarms within a subnetwork to be retrieved by the NMS.
2. Autonomously notify failures, such as termination point failures.
3. Provide loopback-testing capability that supports OAM cell loopback along a subnetwork connection or a segment of it.

Network Security Management. The security framework for ATM networks is described in the ATM Forum document af-sec-0096.000. It addresses the security concerns from the perspectives of customers, public communities, and network operators. The main security objectives are (1) confidentiality (confidentiality of stored and transferred information), (2) data integrity (protection of stored and transferred information), (3) accountability (for all ATM network service invocations and for ATM network management activities), and (4) availability (correct access to ATM facilities).

Seven generic threats are considered in the threat analysis of an ATM network. They are (1) masquerade or spoofing (pretence by an entity to be a different entity), (2) eavesdropping (breach of confidentiality by monitoring communication), (3) unauthorized access, (4) loss or corruption of information, (5) repudiation (an entity involved in a communication exchange subsequently denies the fact), (6) forgery, and (7) denial of service (failure to fulfill its functions by an entity or preventing others from fulfilling).

Table 12.10 maps the threats to the objectives. Not all threats affect all security objectives. For example, masquerade and unauthorized access threatens all security objectives, whereas eavesdropping only affects confidentiality of information.

A set of functional security requirements is identified to deal with the generic threats, and security services should be built to address these requirements. Table 12.11 maps the security requirements to the services needed to fulfill them. You may refer to af-sec-0096.000 for specification of each of the services. Security recovery and management of security do not have associated services but are part of the requirements.

The security framework is specifically applied across the M4 interface. The network management M4 security requirements and the logical MIB are documented in af-nm-0103.000. The MIB is defined independent of protocol implementation and is used for the transfer of information across ATM management interfaces. Both resources and services are defined in the MIB as *managed entities* in the ATM network element.

Table 12.10 Mapping of Threats and Objectives

THREAT	CONFIDENTIALITY	DATA INTEGRITY	ACCOUNTABILITY	AVAILABILITY
Masquerade	x	x	x	x
Eavesdropping	x	–	–	–
Unauthorized access	x	x	x	x
Loss or corruption of information	–	x	x	–
Repudiation	–	–	x	–
Forgery	x	–	x	–
Denial of service	–	–	–	x

Table 12.11 Mapping of Security Requirements and Services

FUNCTIONAL SECURITY REQUIREMENTS		SECURITY SERVICES
Verification of identities		User authentication
Peer entity authentication		
Data origin authentication		
Controlled access and authorization		Access control
Protection of confidentiality	Stored data	Access control
	Transferred data	Confidentiality
Protection of data integrity	Stored data	Access control
	Transferred data	Integrity
		Non-repudiation
Strong accountability		Security alarm, audit trail, and recovery
Activity logging		Security alarm, audit trail, and recovery
Alarm reporting		Security alarm, audit trail, and recovery
Audit		–
Security recovery/Management of Security		–

12.8.9 ATM Digital Exchange Interface Management

The Digital Exchange Interface (DXI) is an interface between a Digital Terminal Equipment (DTE) and a Digital Circuit Equipment (DCE) that connects to a public data network. In the ATM network, the public data network is a network of ATM switches. Figure 12.19 shows a high-level view of the DXI interface. Typically, a DTE will be a hub or the router and the DCE is a Digital Service Unit (DSU), which interfaces to an ATM switch. More details on this interface and management of it can be found in the ATM Forum document on Data Exchange Interface Specification, af-dxi-0014.000.

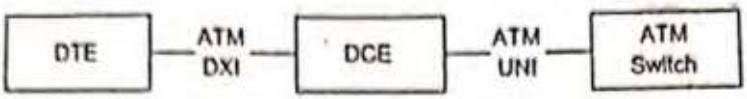


Figure 12.19 The ATM DXI Interface

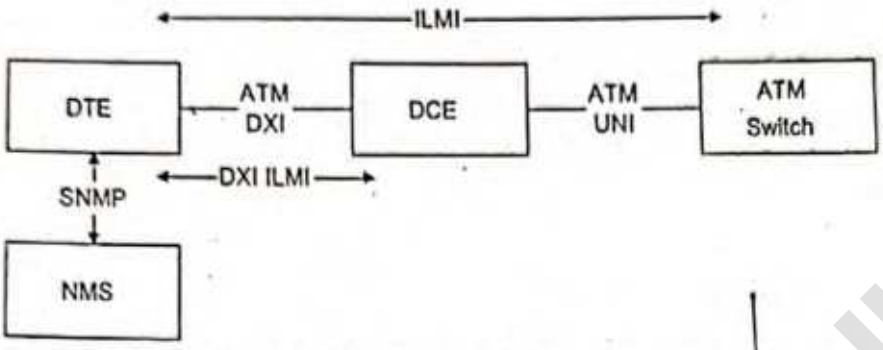


Figure 12.20 ATM DXI Local Management Interface

Figure 12.20 shows the ATM DXI Local Management Interface (LMI). The ATM LMI defines the protocol for exchange of information across the DXI interface, and supports DXI, AAL, and UNI-specific management information. The LMI protocol supports the SNMP management system and the ILMI Management Entity (IME) running on an ATM switch. The ATM DXI LMI MIB (af-dxi-0014.000) supports IETF ATM MIB and the ATM Forum UNI MIB.

12.4 MPLS NETWORK TECHNOLOGY

Multiprotocol Label Switching (MPLS) is a fast-emerging WAN technology that replaces pure IP and ATM networks [RFC 2702]. It combines the richness of IP and the performance of ATM networks.

12.4.1 MPLS Network

One of the important developments in WAN transport technologies is the evolution of the MPLS protocol from IP and ATM. An IP-based network is feature rich because of its extensive implementation and compatibility with Ethernet LAN. It routes packets intelligently. However, it is slow in performance as it has to open each packet at the layer 3 level to determine its next hop and its output port. The simultaneous transport of real-time and non-real-time traffic is difficult.

In contrast to IP, the ATM protocol is a high-performance cell-based protocol switching cells at the layer 2 level. It is capable of handling real-time and non-real-time traffic simultaneously and thus is superior to the IP-based network. It has been deployed extensively in the WAN. However, its address incompatibility with the popular Ethernet LAN, along with difficult end-to-end circuit provisioning, has limited its usage at the customer premises network and hence related applications.

The MPLS protocol evolved to combine the richness of IP and the performance of ATM networks. It is being deployed in convergent network for broadband services handling real-time and non-real-time traffic simultaneously, thus achieving high quality of service transport. It can be deployed in the legacy network of either IP- or ATM-base. A simplified representation of an MPLS domain network with an ingress MPLS router and an egress MPLS router is shown in Figure 12.21. This shows the MPLS header,

13.5 DOCSIS STANDARDS

DOCSIS standards have been evolving. Most of the treatment in this chapter has so far been based on the basics of CM technology, which is DOCSIS 1.0. Later standards are oriented toward interoperability and technological improvements in performance and added services. A subset of documents associated with various versions of DOCSIS standards is given in Table 13.3.

13.5.1 DOCSIS 1.0

DOCSIS 1.0, as we just mentioned, focused on basic technology. Each vendor's CM would communicate only with its own CMTS, as communication between them was proprietary. Downstream transmission was TDM broadcast mode and upstream was hybrid of random access, TDMA, and dedicated bandwidth allocation. There was primitive security and privacy provided by the BPI, and QoS features were also limited. *docsIfMib* and *docsTrCmMib* shown in Figure 13.11 form the basis of all managed objects associated with CM technology.

13.5.2 DOCSIS 1.1

There was significant improvement in features in DOCSIS 1.1 compared to DOCSIS 1.0. Eight levels of QoS were defined to handle real-time and non-real-time video and data traffic. Several implementation improvements were introduced that enhanced performance. IP multicast using Internet Gateway Multicast Protocol (IGMP) was specified to handle multicast transmission. SNMPv3 was added to make management data secure. Service assurance was introduced to handle fault management.

Another major enhancement in DOCSIS 1.1 is the introduction of enhanced security as per BPI+. The intent of the BPI+ specification is to describe the MAC-layer security services between CMTS and CM communications. BPI+ security goals are twofold: they provide CM users with data privacy across the cable network, and they provide MSOs with service protection; i.e., prevent unauthorized users from gaining access to the network's RF MAC services. The protected RF MAC data communications services fall into three categories: (1) best-effort, high-speed, IP data services; (2) QoS (e.g., constant bit rate) data services; and (3) IP multicast group services. The earlier BPI specification [ANSI/SCTE 22-2] had "weak" service protection because the underlying key management protocol did not authenticate CMs. BPI+ strengthens this service protection by adding digital certificate-based CM authentication to its key exchange protocol.

13.5.3 DOCSIS 2.0

Two major enhancements in DOCSIS 2.0 are performance improvement and the introduction of IPv6. DOCSIS 2.0 CM support IPv6 provisioning and management. The term 2.0+IPv6 CM is used to represent such CMs. The use of IPv6 allows MSOs to conserve IPv4 addresses. This Technical Report provides guidelines for CM only. DOCSIS 2.0+IPv6 CM is required to support SNMP over IPv6.

Performance improvement is accomplished by providing options to use either TDMA or synchronous CDMA (S-CDMA) MAC protocol in upstream traffic. Note that there is no random access protocol in this version. Downstream traffic is TDM. The management system should support these protocols.

13.5 DOCSIS 3.0

DOCSIS 3.0 introduces a number of features that build upon features introduced in previous versions of DOCSIS. They include key new features for OSSI based on requirements established with both the introduction of new DOCSIS 3.0 features and enhancements to management capabilities that are designed to improve operational efficiencies for the MSO. Table 13.7 summarizes new requirements that support new 3.0 features and enhancements to existing management features. The table shows management features along with the traditional network management functional areas (fault, configuration, accounting, performance, and security) for the network elements (NE), CM, CMTS, and the corresponding OSI layer where those features operate.

It needs to be noted that pre-3.0 DOCSIS, network management models used IETF RFCs that were defined to use only IPv4. After the introduction of IPv6, IETF IPv6-compliant MIBs are not backward compatible with IPv4-based MIBs required by pre-3.0 DOCSIS. In contrast, provisioning system backward compatibility is a key requirement for management. To accommodate these two conflicting requirements (backward compatibility and IPv6 support using combined v4/v6 MIBs), DOCSIS 3.0 requires maintaining backward compatibility for provisioning but not monitoring.

Special mention needs to be made for accounting management, which in general includes collection of usage data and permits billing the customer based on the subscriber's use of network resources. CMTS is the NE that is responsible for providing usage statistics to support billing. Subscriber account management interface specification (SAMIS) is defined to enable prospective vendors of CM and CMTS to address the operational requirements of subscriber account management in a uniform and consistent manner.

13.6 DSL ACCESS NETWORK

The main motivating factor to employ xDSL (x digital subscriber line) for access technology in multimedia services is the pre-existence of local loop facilities to most households. Information capacity of a 3,000 Hz analog voice channel of 30 dB signal-to-noise ratio based on Shannon limit is 30,000 bits per second [Tanenbaum, 1996]. However, an unloaded twisted pair of copper wire from the central office to a residence can carry a digital T1/DS1 signal at 1.544 Mbps up to 18,000 feet, and a STS-1 signal at 51.840 Mbps up to 1,000 feet. Thus, Shannon's fundamental limitation of data rate that is prevalent in an analog modem can be overcome by direct digital transmission. This is the basic concept behind xDSL technology, which we will now review. You are referred to numerous books on the subject [e.g., Gorlaski, 2001] for an in-depth treatment.

Distance can be increased for analog telephony if we use loaded cables that compensate for loss and dispersion. However, they cannot support the DSL as the loaded coils attenuate high frequencies. Many modern communities have been cabled with fiber coming to the curb with the digital multiplexer at the end of the fiber. The length limitation of the copper cable in this configuration is practically eliminated. This is being taken advantage of in later releases of xDSL such as very high data rate DSL (VDSL).

Basic asymmetric digital subscriber line (ADSL) architecture consists of an unloaded pair(s) of wires connected between a transceiver unit at the central office and a transceiver unit at the customer premises. This transceiver multiplexes and demultiplexes voice and data and converts the signal to the format suitable for transmission on the ADSL link. Table 13.8 [Broadband Forum] shows various forms of DSL and their characteristics. ADSL 2 and ADSL 2+ are enhancements to ADSL.

Table 13.7 Management Features Requirements for DOCSIS 3.0

FEATURES	MANAGEMENT FUNCTIONAL AREA	OSI LAYER	NE	DESCRIPTION
Multiple upstream channels per port	Configuration	PHY	CMTS	Provisioning physical upstream ports that support multiple upstream receivers according to their capabilities
Plant topology		PHY, MAC (Data link)	CMTS	Provisioning flexible arrangements of US/DS channels for channel-bonding configuration to reflect HFC plant topology
Enhanced diagnostics	Fault	PHY, MAC, network	CMTS	Detailed log of different conditions associated with the CM registration state and operation that may indicate plant problems affecting service availability
Enhanced performance data collection	Performance	PHY, MAC, network	CMTS	IPDR streaming of large statistical data sets such as CMTS CM status information with less performance impact on the CMTS resources
Enhanced signal quality monitoring		PHY	CMTS	Gathers information on narrowband ingress and distortion affecting the quality of the RF signals
Usage-based billing	Accounting	PHY, MAC, network	CMTS	Update SAMIS to 3.0 specification requirements
Enhanced security	Configuration, fault, performance, security	MAC, network	CM/CMTS	Updates to management models to support DOCSIS 3.0 security features
IPv6	Configuration, fault, performance	Network	CM/CMTS	Updates to management models to support IPv6 provisioning, CM IP stack management, CMTS and CM IP filtering requirements
Channel bonding	Configuration, fault, performance	PHY, MAC	CM/CMTS	Update existing management models and include new events to support DS and US channel bonding
IP multicast	Configuration, fault, performance	MAC, network	CM/CMTS	Update existing management models to support new multicast capabilities such as SSM, IGMP v3, MLD v1 and v2

Table 13.8 DSL Technologies

NAME	MEANING	MAX DATA RATE*	MODE	CABLE	APPLICATIONS
ADSL/ADSL2/ ADSL2+	Asymmetric digital subscriber line	7/12/24 Mbps 0.8/1/1 Mbps	Down Up	1-pair	Most common type
SHDSL	Symmetric high data rate DSL	5.6 Mbps	Duplex Duplex	2-pair	Business connections
VDSL 1 km	Very high data rate digital subscriber line	55 Mbps 15 Mbps	Down Up	2-pair	Triple play (no QoS)
VDSL2- long reach 3 km	Very high data rate digital subscriber line	55 Mbps 30 Mbps	Down Up	2-pair	Triple play
VDSL short reach 500 m	Very high data rate digital subscriber line	100 Mbps	Down Up	2-pair	Triple play

*Max data rate as per broadband forum

Symmetric high data rate digital subscriber line (SHDSL) operates at T1 or E1 data rate in a duplex mode with two pairs of wires [RFC 3276]. The duplex mode is defined as two-way communication with the same speed in both directions. Symmetric HDSL operates with two pairs, one for each direction. SHDSL typically operates at rates from 256 Kb/s to 6 Mb/s upstream and downstream.

VDSL operates asymmetrically. As in ADSL, the downstream signal has a larger bandwidth and is at the high end of the spectrum, whereas the upstream is at the lower end of the spectrum with a lower bandwidth for the signals. VDSL 2 has long- and short-range implementation, the former being asymmetrical and the latter being symmetrical.

13.7 ASYMMETRIC DIGITAL SUBSCRIBER LINE

Among all the xDSLs, the asymmetric digital subscriber line (ADSL) is the technology that is being deployed now in most of the world. A simplified access network using ADSL is shown in Figure 13.15 and consists of an ADSL transmission unit (ATU) and splitter at each end of the ADSL line. The ATU acronym has also been expanded in print as the ADSL transceiver unit as well as the ADSL terminating unit, although ADSL TR-001 defines it as the ADSL transmission unit. The ATU at the central office is ATU-C and the one at the customer residence is ATU-R. The ATU is also called the ADSL modem. The data and video signal from the broadband network is converted to an analog signal by the ATU-C and multiplexed and demultiplexed. The splitter at the central office combines the plain old telephone service (POTS) voice signal and the broadband signal. The reverse process occurs at the splitter and ATU-R at the customer premises (residence). There are modems available that embed the splitter and thus eliminate a separate splitter at the customer site. This configuration is referred to as ADSL-Lite, also known as GLite.

As mentioned above, upstream and downstream signals are placed asymmetrically in the frequency spectrum, as shown in Figure 13.16. The POTS signal is always allocated the baseband of 4 kHz and separated from the broadband signal by a guard band. There are two schemes for separating the upstream and downstream frequency bands: frequency division multiplexing (FDM) or echo cancellation.

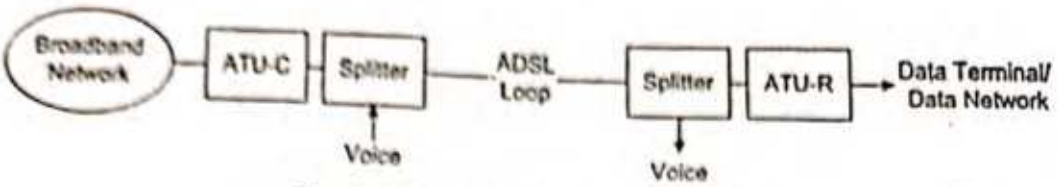


Figure 13.15 ADSL Access Network

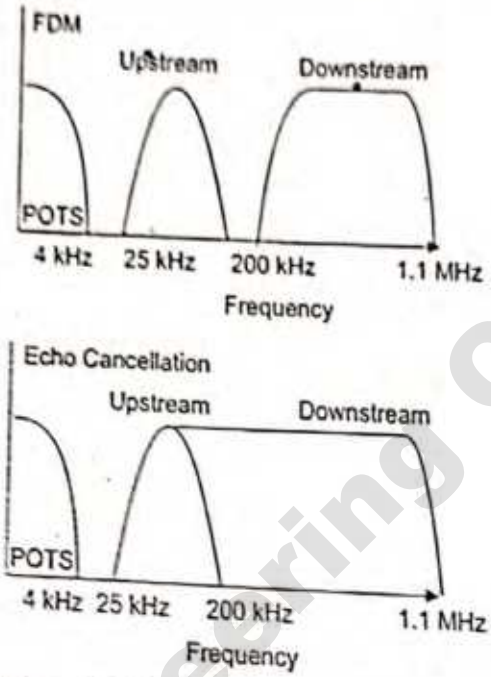


Figure 13.16 ADSL Spectrum Allocation

In FDM, after separating the upstream and downstream bands, each band is then divided into one or more high-speed channels and one or more low-speed channels. In echo cancellation, upstream and downstream bands overlap, but are separated by a technique known as echo cancellation. Using echo cancellation, the low frequency end of the spectrum is made available for downstream, thus increasing the overall downstream spectral band.

Within the upstream and downstream bands, individual channels are allocated a multiple of 4 kHz band using either the standard discrete multitone (DMT) or carrierless amplitude phase (CAP) modulation. The former modulation scheme is more efficient, but more complex and costly. Both schemes are currently in use. You may consult the reference for further details on spectrum allocation schemes [Goralski, 2001].

Standards are addressed by various standards organizations for XDSL including the American National Standards Institute (ANSI). T1-413 is the ANSI standard for XDSL at the physical-layer protocol level. In order to accelerate interoperability and implementation of ADSL, the industry had established a consortium, the ADSL Forum, to address issues associated with end-to-end system operation, management, and security. This organization is now called the Broadband Forum.

Not all residential customers could enjoy the privilege of getting ADSL service. Only those who have direct copper connection from the central office could be served with ADSL. Telephone loop facilities with loaded coils do not qualify for the service. Besides, many newer residential complexes have fiber cable to the neighborhood (FTTN) and twisted pair from FTTN to the residence. For these residences,

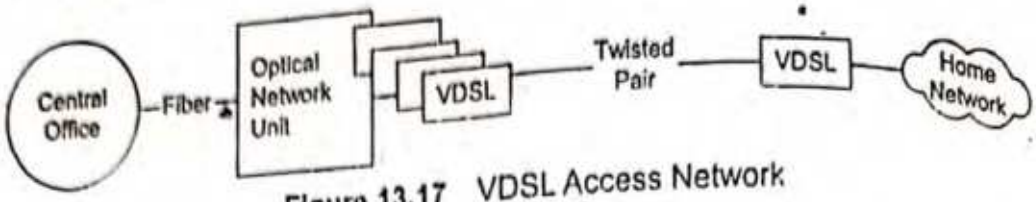


Figure 13.17 VDSL Access Network

telephone companies would offer VDSL service, as shown in Figure 13.17. This has the benefit of providing greater bandwidth. The signal traverses the optical fiber medium from the central office to the optical network unit (ONU) in the neighborhood carrying multiple channels. It is demultiplexed at the ONU and fed through VDSL modems and twisted-pair cable to the residence. A bandwidth of 12.96–55.2 Mbps could be achieved for downstream and up to 15 Mbps for the upstream using a single-pair cable. There are other configurations proposed from ONU to home, which we do not plan to deal with here.

Much of the latest documentation on ADSL is available on TRs published by the ADSL Forum, RFCs released by IETF, and ITU-T standards. The approved documentation list of the ADSL Forum is available in the public domain of the Broadband Forum's Web page, www.broadband-forum.org. Some TRs that are relevant to the basics and management of DSL that we refer to are listed in Table 13.9. ITU-T standards G.xx and RFCs relating to the MIBs are included in the Bibliography at the end of the book.

13.7.1 ADSL Access Network in Overall Network

Broadband Forum's view [TR-001] of how ADSL access network fits into the overall network for broadband services is presented in Figure 13.18. It shows the components of the overall network comprising private, public, and premises network and the role that ADSL access network plays in it. The networking side of the service providers consists of service systems, different types of networks that are behind the access node, the operation systems (OS) that perform the operations, administration, and maintenance (OAM) of the networks and access nodes, and the ATU-Cs. The customer premises network comprises ATU-R, premises distribution network (PDN), various service modules (SM), and terminal equipment (TE). On the bottom of Figure 13.18 are shown five transport modes that depict an evolutionary process from a primitive synchronous transfer mode (STM) to an all ATM mode.

Table 13.9 ADSL Management Documents

TR-001	ADSL Forum System Reference Model	May 1996
TR-005	ADSL Network Element Mgmt	March 1998
TR-015	CAP Line Code-Specific MIB	February 1999
TR-024	DMT Line Code-Specific MIB	June 1999
TR-027	SNMP-based ADSL LINE MIB	September 1999
TR-028	CMIP Specification for ADSL Network Element Mgmt	May 1999
TR-066	ADSL Network Element Mgmt (Update to TR-005)	March 2004
TR-090	Protocol-Independent Object Model for Managing Next Generation ADSL Technologies	December 2004
TR-113	MCM-Specific Managed Objects in VDSL Network Element	December 2005
TR-128	Addendum to TR-090	September 2006

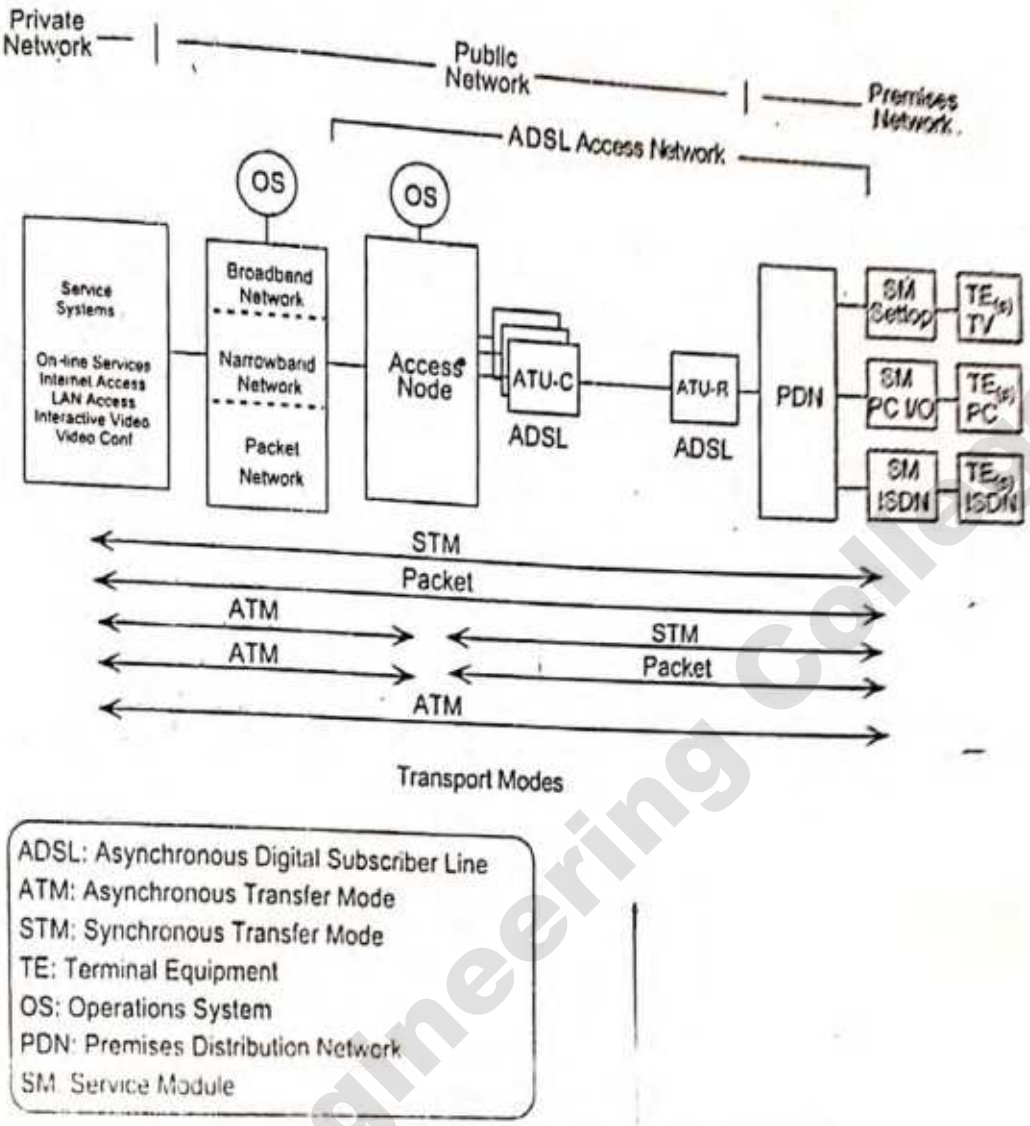


Figure 13.18 Overall Network and ADSL

The service systems are on a private network providing on-line services, Internet access, LAN access, interactive video, and video conference services. The private network interfaces with the public network, which is broadband (such as SONET/SDH), narrowband (such as T1/E1), or packet network (such as IP). The access node is the concentration point for broadband, narrowband data, and packet data. It is either located in the central office or a remote location such as ONU. The access node could include ATU-Cs, such as in a digital subscriber loop access multiplexer (DSLAM). The access network commences at the access node and extends up to PDN in the customer premises.

The premises network starts from the network interface at the output of ATU-R. The PDN, which is part of the home network, could be a choice of a LAN, twisted-pair cable of telephone network, consumer electronics bus (CEBus) which distributes signal over power lines, coaxial cable, optical fiber (in future homes), or a combination of these. SM, such as set-top boxes and ISDN, perform the termination adaptation functions to the TE.

There are five transport modes presented in Figure 13.18. At the top is what the ADSL Forum terms synchronous transfer mode (STM), which is the bit synchronous transmission mode. An example of this is the bit pipe such as T1/E1, ISDN, or a simple modem. In this mode, the PDN outputs strictly bits out of the SM, and the access node delivers and receives bits to and from the narrowband network.

The second transport scheme is the end-to-end packet mode such as IP packets. In this mode, SM are expected to deliver packets to the ADSL access network through PDN. This is probably one of the most common usages of the Small Office Home Office (SOHO) network. Digital data terminals are interconnected via an Ethernet LAN PDN, and packets are delivered to the ADSL access network via a router. The reverse process occurs at the access node to the network interface.

The next two transport modes are hybrid modes. Output to the network from the access node is the ATM. The SM at the premises network delivers either a bit synchronous output or a packet output. There is a conversion involved in the access node. For example, the access node to the broadband network could be an emulated LAN, wherein IP packets could be transferred as ATM cells to the network.

The fifth, and last, mode of transport scheme is the end-to-end ATM, where SM put out cells instead of packets. We would expect the home network in this case to be wired with optical fiber.

13.7.2 ADSL Architecture

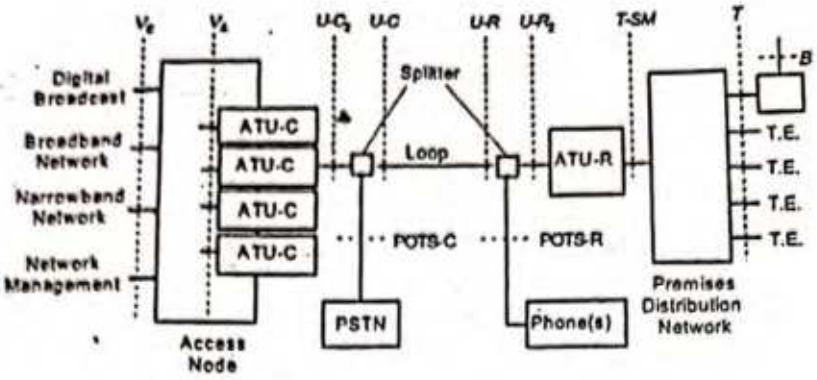
Let us now look at the system architecture details of the ADSL access network presented in Figure 13.15. The ADSL Forum's ADSL system reference model [TR-001] is shown in Figure 13.19. We have already discussed some components in Section 13.7.1. Additional components are splitters at the central office and customer premises, which separate low-frequency telephony from video and digital data. Public-switched telephone network (PSTN) is the switch connected at the central office, while telephones are off the splitter at the customer end. We notice the digital broadcast and network management interfacing with the access node. Digital broadcast is the typical broadcast video. Network management could be treated as one of the operations system components. We will be going into more details on the operations system interfaces and functions in Section 13.8.

Interesting aspects of the ADSL system reference model shown in Figure 13.19 are the interfaces between components of the ADSL network and interfaces between ADSL access network and external networks. There are five basic interfaces: V, U, T, B, and POTS. V_C is the interface between the access node and the network and is usually a physical interface. An interface could have multiple physical connections (as shown in the figure), or multiple logical interfaces can be connected through a physical interface. V_A is the logical interface between ATU-C and the access node. Network management is implemented through the V_C interface. All network monitoring of the central office and the home network component has to go through this interface.

There are several U interfaces shown in Figure 13.19. They are all off the splitters. In fact, these U interfaces may disappear when ADSL-Lite is implemented, although it is highly unlikely for a long time. POTS interfaces are also from the splitters as shown. The B interface is for auxiliary data input; for example, a satellite feed directly into a SM such as a set-top box.

13.7.3 ADSL-Channeling Schemes

There are two perspectives in discussing transport channels in an ADSL access network. The first perspective is the traditional transport bearer channels as they are defined in ISDN. For ADSL transport frames, there are seven "AS" bearer channels defined for the downstream signal operating in a simplex mode. The AS bearer channels are in multiples (one, two, three, or four) of T1 rate of 1.536 Mbps or E1 rate of 2.048 Mbps. In addition to downstream AS channels, there could be three additional "LS" duplex channels carrying the signal in both downstream and upstream directions. The LS bearer channels are 160, 384, or 576 Kbps. The reader is referred to [Goralski, 2000] for a detailed discussion of these. Incidentally, "AS" and "LS" are not specific acronyms.



Interfaces:
B: Auxiliary data input such as a satellite feed to Service Module (TE)
POTS-C: Interface between PSTN and POTS splitter at network end
POTS-R: Interface between phones and POTS splitter at premises end
T: Interface between Premises Distribution Network and Service Modules
T/SM: Interface between ATU-R and Premises Distribution Network
U-C: Interface between Loop and ATU-C (analog)
U-C2: Interface between POTS splitter and ATU-C
U-R: Interface between Loop and ATU-R (analog)
U-R2: Interface between POTS splitter and ATU-R
V_a: Logical interface between ATU-C and Access Node
V_c: Interface between Access Node and network

TE: Terminal Equipment
POTS: Plain Old Telephone Service
PSTN: Public-Switched Telephone Network

Figure 13.19 ADSL System Reference Model

The second perspective in discussing the channels is how the signal is buffered while traversing the ADSL link. This is represented in Figure 13.20. Real-time signals, such as audio and real-time video, use a fast buffering scheme and hence are referred to as the *fast channel*. Digital data that could tolerate delay use slow buffers that are interleaved between the fast signals. The digital data channel is referred to as the *interleaved channel*. Thus, a physical interface would carry both the fast channel and the interleaved channel and needs to be addressed in the network management of interfaces. We will discuss the interface types of the physical, fast, and interleaved channels more in Section 13.8.

13.7.4 ADSL-Encoding Schemes

ADSL management is dependent on the line-encoding scheme used, and hence we will briefly discuss the two types here. There are two encoding schemes used in ADSL line encoding. They are carrierless amplitude and phase (CAP) modulation and discrete multitone (DMT) technology. Both are based on the QAM scheme that we discussed in Section 13.2. In both cases, the basic approach is to separate the

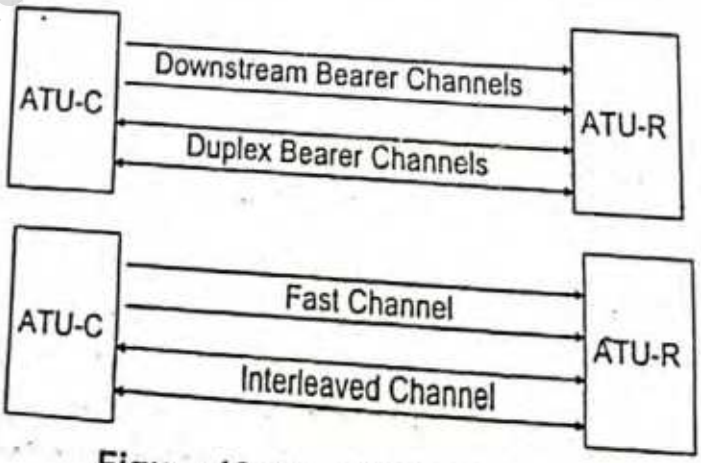


Figure 13.20 ADSL Channeling

POTS band (0–4 kHz), as shown in Figure 13.16. It also shows two views, one that uses FDM to separate the upstream signal from the downstream signal by a guard band. In the second view, upstream and downstream signals overlap, but are distinguished by the echo cancellation technique.

An echoing phenomenon occurs in the telephone system due to crosstalk between neighboring pairs of wires in a bundle. Two signals transmitted from the central office could couple with each other, which is termed *near-end crosstalk*. Two signals traversing in opposite directions could also interfere with each other, which is *far-end crosstalk*. Both of these are mitigated using the echo cancellation technique. The same technique is used to separate the overlapping band between the upstream and downstream shown in the echo cancellation view of Figure 13.16.

Although the ANSI has recommended the use of DMT for ADSL, there currently exist deployed systems that use the CAP system. CAP, as you may recall, is carrierless. In other words, the signal is quadrature amplitude modulated at a specific carrier frequency; the carrier is suppressed at the transmitter, and then sent. The carrier is regenerated at the receiver to detect the signal bits. In CAP, the entire local loop bandwidth (25–200 kHz for upstream or 200 kHz to 1.1 MHz for downstream) is used in the encoding.

In DMT, the entire bandwidth of approximately 1.1 MHz is split into 256 subchannels, each of approximately 4 kHz band. Subchannels 1–6 are used for voice signals and the rest for broadband signals. There are 32 (7–38) upstream subchannels. The number of downstream subchannels is either 250 if echo cancellation is used or 218 if no echo cancellation is performed.

13.8 ADSL MANAGEMENT

The general framework for ADSL management is described in ADSL Forum TR-005 and updated in TR-066. TR-027 presents SNMP-based ADSL Line MIB, and TR-028 contains CMIP specification for ADSL NE management. TR-024 and TR-015 document DMT Line Code-Specific (LCS) MIB and CAP LCS MIB, respectively. Management documentation is specific to ADSL and is a supplement to standard management MIB.

Figure 13.21 shows the ADSL system reference model that is used in the ADSL management framework. It is similar to the one shown in Figure 13.19, but has additional components identifying the switching and physical-layer functions explicitly. Management functions addressed in ADSL-specific documents deal with physical-layer functions. The management of data layers is addressed by the conventional NMSs. Low- and high-pass filters are also explicitly shown in the figure.

13.8.1 ADSL Network Management Elements

ADSL network management deals with parameters, operations, and protocols associated with configuration, fault, and performance management. Security and accounting management are not explicitly dealt with, although these are important management functions and are addressed by other models (for example, SNMP security management discussed in Chapters 7 and 11).

The management of the ADSL network involves the following five NEs: (1) management communications protocol across the network management subinterface of the V interface; (2) management communications protocol across the U interfaces between ATU-C and ATU-R; (3) parameters and operations with the ATU-C; (4) parameters and operations within the ATU-R; and (5) ATU-R side of the T interface. All management functions in the ADSL network are accomplished via the V interface. Thus, the management of elements 2–5 is accomplished via the V interface and not the U interface.

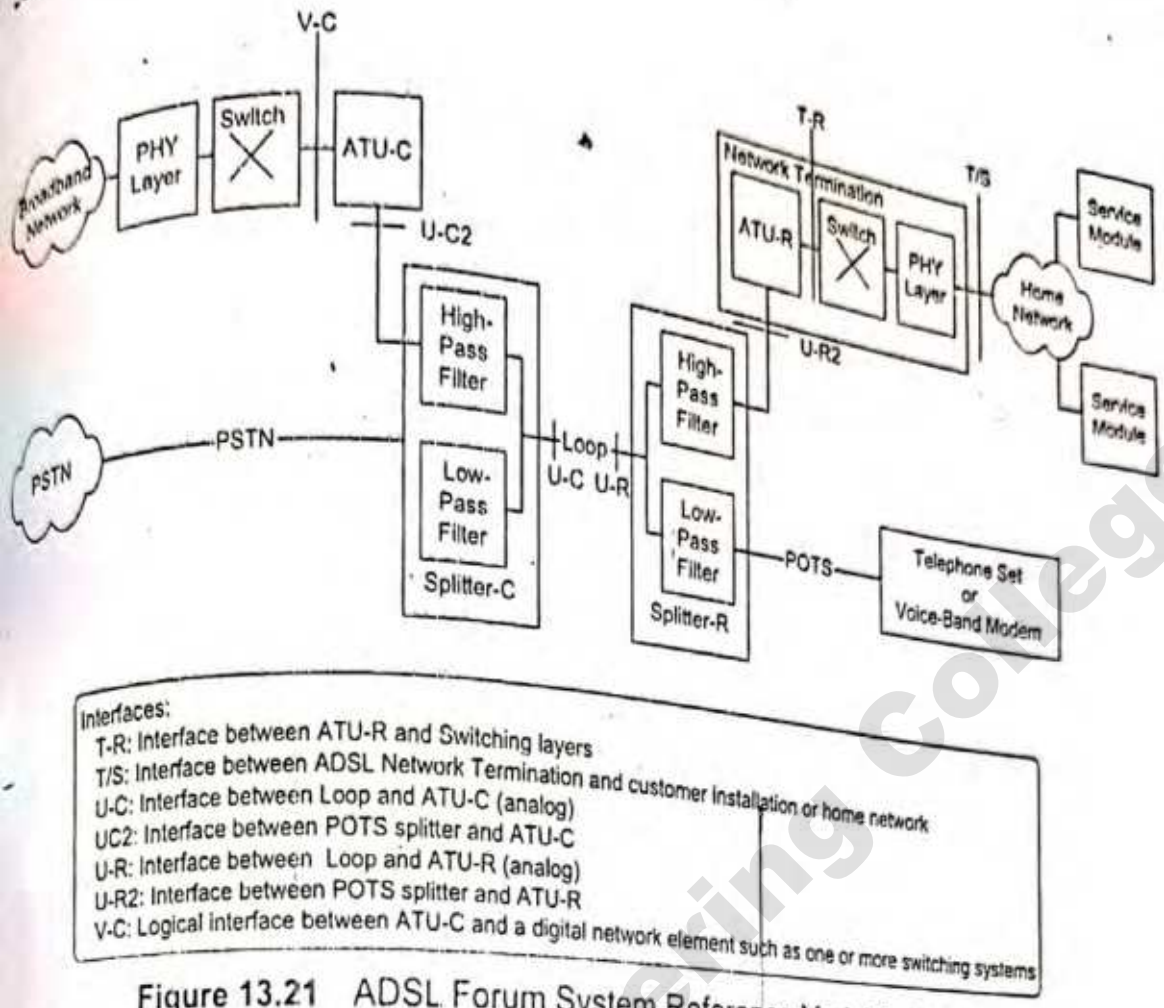


Figure 13.21 ADSL Forum System Reference Model for Management

As discussed in Section 13.7.3, the management function at the physical layer involves three entities: physical channel, fast channel, and interleaved channel. Fast and interleaved channels need to be managed separately. These two use the physical transmission medium that also needs to be managed. Besides the management of physical links and channel parameters, the parameters associated with the type of line coding need to be monitored. We will look at various parameters associated with configuration, fault, and performance management in the next section.

13.8.2 ADSL Configuration Management

Various parameters that need to be managed for configuration are listed in Table 13.10. The table lists the component that the parameter is associated with, as well as whether it pertains to the physical line or fast or interleaved channel. A brief description of each parameter is given in the last column.

It is to be noted that the link could be configured in one of five options: no separation of channels, fast, interleaved, either, or both.

There are five levels of noise margin—from the highest defined by the maximum noise margin to the lowest defined by the minimum noise margin. The transmitted power of the modem needs to be decreased or increased, respectively, based on these thresholds. The transmission rate can be increased if the noise margin goes above a threshold level, which is beneath the maximum noise margin threshold. Similarly, the transmission rate should be decreased if the noise margin falls below a certain threshold, which is higher than the minimum noise margin. Right at the middle of all these thresholds is the steady-state operation. These levels are shown in Figure 13.22.

Table 13.10 ADSL Configuration Management Parameters

PARAMETER	COMPONENT	LINE	DESCRIPTION
ADSL line type	ADSL Line	N/A	Five types: no channel, fast, interleaved, either, or both
ADSL line coding	ADSL Line	N/A	ADSL coding type
Target noise margin	ATU-C/R	Phy	Noise margin under steady state (BER = $<10^{-7}$)
Max. noise margin	ATU-C/R	Phy	Modem reduces power above this threshold
Min. noise margin	ATU-C/R	Phy	Modem increases power below this margin
Rate adaptation mode	ATU-C/R	Phy	Mode 1: Manual Mode 2: Select at start-up Mode 3: Dynamic
Upshift noise margin	ATU-C/R	Phy	Threshold for modem increases data rate
Min. time interval for upshift rate adaptation	ATU-C/R	Phy	Time interval to upshift
Downshift noise margin	ATU-C/R	Phy	Threshold for modem decreases data rate
Min. time interval for downshift rate adaptation	ATU-C/R	Phy	Time interval to downshift
Desired max. rate	ATU-C/R	F/I	Max rates for ATU-C/R
Desired min. rate	ATU-C/R	F/I	Min. rates for ATU-C/R
Rate adaptation ratio	ATU-C/R	Phy	Distribution ratio between fast and interleaved channels for available excess bit rate
Max. interleave delay	ATU-C/R	F/I	Max. transmission delay allowed by interleaving process
Alarm thresholds	ATU-C/R	Phy	15-minute count threshold on loss of signal, frame, power, and error seconds
Rate-up threshold	ATU-C/R	F/I	Rate-up change alarm
Rate-down threshold	ATU-C/R	F/I	Rate-down change alarm
Vendor ID	ATU-C/R	Phy	Vendor ID assigned by T1E1.4
Version No.	ATU-C/R	Phy	Vendor-specific version
Serial No.	ATU-C/R	Phy	Vendor-specific serial no.

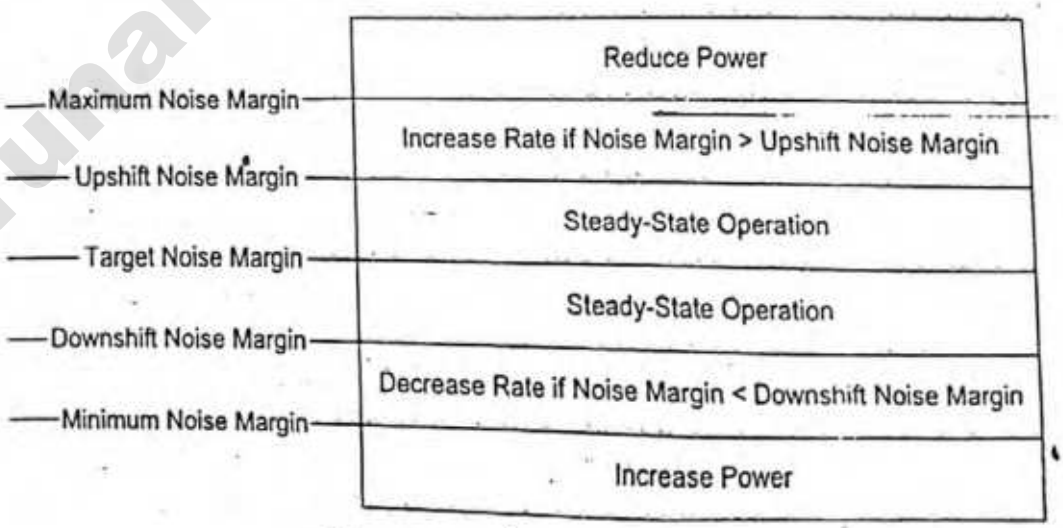


Figure 13.22 Noise Margins

Some modems support rate adaptation modes. There are three modes. Mode 1 is manual in which the rate is changed manually. In mode 2 the rate is automatically selected at start-up, but remains at that level afterwards. The last mode is mode 3 where the rate is dynamic based on the noise margin.

13.8.3 ADSL Fault Management

Fault management parameters are shown in Table 13.11 and should be displayed by the NMS. After the automatic indication of faults, ATU-C and ATU-R self-tests as specified in T1.413 could be used to assist in the diagnostics.

The ADSL line status shows the current state of the line as to whether it is operational, or there is a loss of any of the parameters on frame, signal, power, or link. It also indicates initialization errors. Alarms are generated when the preset counter reading exceeds 15 minutes on loss of signal, frame, power, link, and error seconds.

13.8.4 ADSL Performance Management

Table 13.12 shows the parameters associated with ADSL performanee management. Each ATU's performance in terms of line attenuation, noise margin, total output power, current and previous data rate, along with the maximum attainable rate, channel data block length (on which CRC check is done), and interleave delay can be monitored. In addition, statistics are gathered for a 15-minute interval and a 1-day interval on the error-seconds statistics. Two counters are maintained by each ATU for each error condition to measure these. Error statistics are maintained for loss of signal seconds, loss of frame seconds, loss of power seconds, loss of link seconds, error seconds, transmit blocks, receive blocks, corrected blocks, and uncorrectable blocks.

13.8.5 SNMP-based ADSL Line MIB

There are both SNMP- [RFCs 2662, 3440] and CMIP- [TR-016] based specifications that have been developed for ADSL. We will discuss the updated SNMP-based MIB [Table 13.9; RFC 2662; TR-015; TR-024; TR-027] in this section. ADSL SNMP MIB is presented in Figure 13.23.

There are three nodes defined under *adslLineMib {adslMIB 1}*. RFC [2662] specifies *adslMibObjects {adslLineMib 1}*, which are shown in part in Figure 13.23.

Notice that there are complimentary objects for the link in terms of physical and channel objects. For example, there are *adslAtucPhysTable* and corresponding *adslAtucChanTable*. The former specifies

Table 13.11 ADSL Fault Management Parameters

PARAMETER	COMPONENT	LINE	DESCRIPTION
ADSL line status	ADSL Line	Phy	Indicates operational and various types of failures of the link
Alarms thresholds	ATU-C/R	Phy	Generates alarms on failures or crossing of thresholds
Unable to initialize ATU-R	ATU-C/R	Phy	Initialization failure of ATU-R from ATU-C
Rate change	ATU-C/R	Phy	Event generation when rate changes when crossing of shift margins in both upstream and downstream

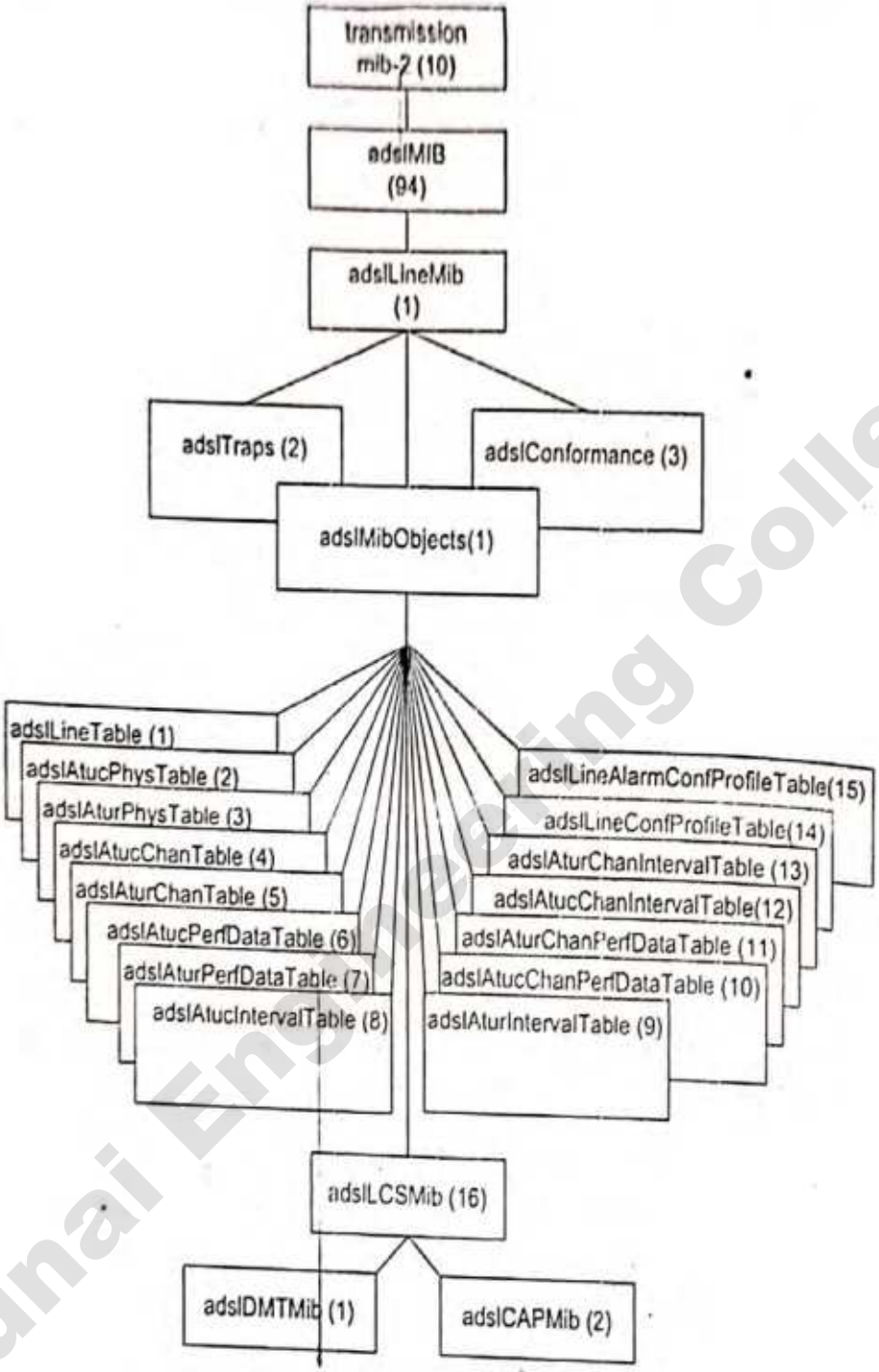


Figure 13.23 ADSL SNMP MIB

a table with each row containing the physical-layer parameters associated with the link on an interface. The latter specifies a table in which each row contains parameters associated with a channel on that interface. These interfaces are defined based on RFC 1213 for *interfaces {mib-2 2}* and RFC 2863 for *ifMIB {mib-2 31}*.

Figure 13.23 contains objects under *adslMibObjects* that pertain to configuration management parameters defined in Table 13.10, fault management parameters defined in Table 13.11, and performance management parameters in Table 13.12.

Table 13.12 ADSL Performance Management Parameters

PARAMETER	COMPONENT	LINE	DESCRIPTION
Line attenuation	ATU-C/R	Phy	Measured power loss in dB from transmitter to receiver ATU
Noise margin	ATU-C/R	Phy	Noise margin in dB of the ATU with respect to the received signal
Total output power	ATU-C/R	Phy	Total output power from the modem
Max. attainable rate	ATU-C/R	Phy	Max. currently attainable data rate by the modem
Current rate	ATU-C/R	F/I	Current transmit rate to which the modem is adapted
Previous rate	ATU-C/R	F/I	Rate of the modem before the last change
Channel data block length	ATU-C/R	F/I	Data block on which CRC check is done
Interleave delay	ATU-C/R	F/I	Transmit delay introduced by the interleaving process
Statistics	ATU-C/R	Phy F/I	15 minute/1 day failure statistics

13.8.6 MIB Integration with Interfaces Group in MIB-2

The ADSL LINE MIB specifies detailed attributes of a data interface. It is integrated with IF-MIB with the following *ifType*(s) relative to ADSL in the following manner:

adslPhysIf ::= {transmission 94}
adslInterIf ::= {transmission 124}
adslFastIf ::= {transmission 125}

Each MIB branch would have the appropriate tables for that interface type and would *augment* the interfaces table with *ifIndex* in *ifEntry* as the accessing index.

Table 13.13 presents the objects needed for ADSL, which are part of the mandatory *ifGeneralGroup* [RFC 2863]. They are applicable to the line, not either end in particular. "NORMAL" means the variable is used normally as specified in MIB-II. Designations "i," "j," and "k" indicate three arbitrary *ifIndex* values corresponding to the physical, interleaved, and fast entries for a single ADSL line.

The *ifStackTable* {*ifMIB.ifMIBObjects 2*}, which is the table containing information on the relationships between multiple sublayers of network interfaces, is used to associate the fast and interleaved channels with the physical line. The top of Figure 13.24 shows the logical representation of the channels. Their relationship with each other and with the higher layer is shown in the bottom of Figure 13.24. The fast channel and the interleaved channel, which are at the same level, are stacked on top of the physical layer. They interface above with a higher layer, for example ATM if ATM is over ADSL.

13.8.7 ADSL Operational and Configuration Profiles

Table 13.14 shows the configuration of the operational file for *ifType* for each modem in setting up the fast and interleaved channels. An *ifIndex* is associated with each channel.

In a typical configuration of an ADSL system, the access node shown in Figure 13.19 has hundreds of ATU-Cs. It would be impractical to provision all the parameters for each ATU-C individually. There are two MIB tables to address this issue—one for configuration profile and another for the performance

Table 13.13 Use of Interfaces Table for ADSL

MIB VARIABLE	PHYSICAL LINE (I)	INTERLEAVED CHANNEL (J)	FAST CHANNEL (K)
ifDescr	NORMAL	NORMAL	NORMAL
ifType (IANA)	94	124	125
ifSpeed	ATU-C line Tx rate	ATU-C channel Tx rate	ATU-C channel Tx rate
ifPhyAddress	NULL	NULL	NULL
ifAdminStatus	NORMAL	NORMAL	NORMAL
ifOperStatus	NORMAL	NORMAL	NORMAL
ifLastChange	NORMAL	NORMAL	NORMAL
ifLinkUpDownTrapEnable	NORMAL (default: Enable)	NORMAL (default: Enable)	NORMAL (default: Enable)
ifConnectPresent	True	False	False
ifHighSpeed	NULL	NULL	NULL

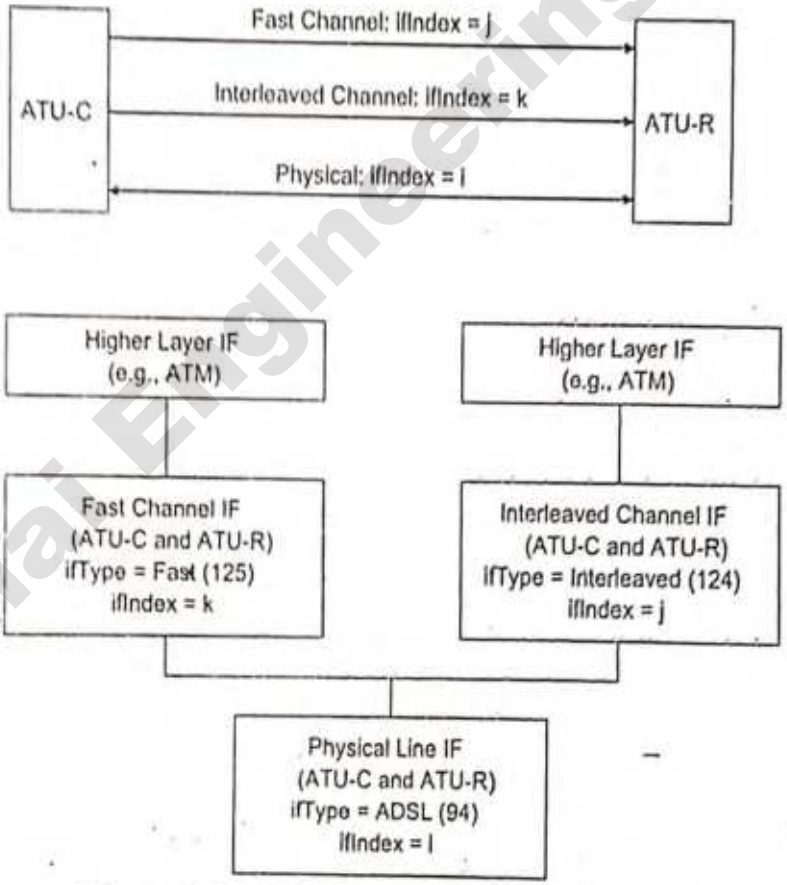


Figure 13.24 Relationship between ADSL Entries

profile. One of the tables is *adslLineConfProfileTable {adslMibObjects 14}*, which contains information on the ADSL line configuration shown in Table 13.10. One or more ADSL lines may be configured to share common profile information. Figure 13.25 shows the dynamic mode, MODE-I, configuration profile scheme. Profile tables are created and indexed 1 to *n*. Each ADSL line interface, with the given

interfaces. The major difference is with respect to security requirements. The interface requirements for other interfaces are not available in the standards. In the case of F interface, this work has recently begun within ITU.

Three classes of applications are introduced in this chapter. The interactive class of applications exchange management information using either a command/response format or asynchronous event reports. The file transfer class is concerned with managing large volumes of data stored in files. TMN Directory is a third class of application which is a support service for successful operation of TMN. The information exchanges is not specific to management, it supports determining information about the managing and managed systems required for successful communication between these entities.

The protocol requirements are grouped into lower and upper layers according to the OSI Reference Model. The TMN interfaces use many different protocols at the lower layers including the Internet protocol TCP/IP. The upper layer requirements are the same irrespective of the protocol stack (also called a profile in Q.811) selected for use on a specific interface implementation. The various protocol profiles at the lower layers are further grouped in terms of connection-oriented and connectionless protocols.

The application layer is complex and can be considered to be made up of building blocks that may be reused. The application layer structure, specifically the components of the application entity, are discussed in this chapter for the interactive class of applications. The reuse concept is seen from the repeated occurrence of ACSE to support the three classes of applications.

Security requirements for Q3 and X interfaces are discussed according to what is available in Recommendation Q.812 and the newly approved standard Q.813, STASE-ROSE.

This chapter discussed requirements for all the layers with more details specified at the lower layers. The two application services for interactive and file-oriented classes are discussed in the next chapter.

SNMPv1 Network Management: Organization and Information Models

OBJECTIVES

- *ETF SNMP standard*
  - *History*
  - *RFC, STD, and FYI*
- *Organization model*
  - *2- and 3-tier models*
  - *Manager and agent*
- *Management messages*
- *Structure of management information, SMI*
- *Object type and instance*
- *Scalar and aggregate managed objects*
- *Management information base, MIB*
- *NMS physical and virtual databases*
- *IETF MIB-2 standard*

SNMP management is also referred to as Internet management. We have chosen to call it SNMP management since it has matured to the level that it manages more than the Internet, for example, intranet and telecommunications networks. Any network that uses TCP/IP protocol suite is an ideal candidate for SNMP management. SNMP network management systems (NMSs) can manage even non-TCP/IP network elements through proxy agents.

SNMP management is the most widely used NMS. Most network components that are used in enterprise network systems have network agents built in them that can respond to an SNMP NMS. Thus, if a new component, such as a host, a bridge, or a router that has an SNMP agent built in, is added to a managed network, the NMS can automatically start monitoring the added component. The ease of adding components and configuring them for management has contributed to the acceptance and popularity of the SNMP management system. To quote Marshall Rose [Rose, 1996], who is one of the early architects of SNMP management, the fundamental axiom is, "the impact of adding network management to managed nodes must be minimal, reflecting a lowest common denominator."

SNMP management got started as an interim set of specifications, the ultimate standard being
management. Since that did not materialize, SNMP specifications were enhanced by the developm
SNMPv2 and SNMPv3. The first version of SNMP is informally referred to as SNMPv1, as it is
in this chapter. SNMPv2 and SNMPv3 are covered in Chapters 6 and 7, respectively.

We start by giving a real-world example of a managed network in Section 4.1 and show the kin
detailed information one could gather from an NMS. We then learn what SNMP management is and
that enables us to obtain that kind of information. The history of SNMP management goes back to
in managing the Internet. The Internet Engineering Task Force (IETF) has the responsibility to dev
Internet standards including network management standards. The standards documents are avail
free in Request for Comments documents (RFCs). These are covered in Sections 4.2 and 4.3.

The SNMP management model is introduced in Section 4.4 and addresses primarily organiza
information, and communication. An NMS comprises management process, agent process, and net
elements. We discuss various possible configurations in Section 4.5. There are three messages trans
ted by the manager and two by the agent for a total of five messages. Management data are obtaine
the manager by polling the agents. Agents respond with requested data. They also generate a few ala
when needed. This simple architecture of SNMP management is described in Section 4.6.

SNMP information model, described in Section 4.7, comprises the Structure of Management In
mation (SMI) and the Management Information Base (MIB). SMI uses ASN.1 syntax to define mana
objects. SMIv1 documented the specifications distinct from the formal ASN.1 definition as it was
expected that OSI would be the future standard. However, that did not happen. Hence, SMIv2 mer
the two parts into a concise document. The MIB defines the relationship between managed objects
groups of related objects into MIB modules. MIB-II is a superset of MIB-I and is used in SNMPv1.

SNMP architecture, administration, and access policies, which fall under the communication mo
are discussed in the next chapter.

4.1 MANAGED NETWORK: CASE HISTORIES AND EXAMPLES

Let us look at some of the real-world experiences that demonstrate the power of network manageme
before learning how it is accomplished. As with any good technology, the power of technology cou
result in both positive and negative results. Atomic energy is a great resource, but an atomic bomb is no
An NMS is a powerful tool, but it could also bring your network down, when not "managed" properl

As part of my experience in establishing a network operations center, as well as in teaching a netwo
management course, we made several visits to see how various corporations and institutions mana
their networks. One of the visits was to an AT&T Network Control Center, which monitored the netwo
status of their network in the entire eastern half of the United States. We could see the network of nod

and links on a very large screen, mostly in green indicating that the network was functioning well. The display screen would automatically refresh every few minutes. We saw nodes or links change color to yellow or red, indicating a minor or major alarm. We would also then see them turn back to green without interference by any human being. What we were seeing was monitoring of a national network from a central monitoring center. Monitoring was done by the NMSs and operations support systems without any human intervention. Even the healing of the network after a failure was accomplished automatically—self-healing network as it is called. Any persistent alarm was pursued by the control center, which tested the network remotely using management tools to isolate and localize the trouble. It was an impressive display of network management capability.

In another visit to a major international news network world headquarters, we were shown the monitoring of not only network failures, but also the performance of networks around the globe. Network Operations Center personnel were able to look at networks in various continents separately, as well as in the global integrated network. The system was putting out not only alarms, but also the cause of the failure, which was accomplished using artificial intelligence built in the system.

On a more intimate level, one of the directors of Information Technology was narrating his experience of resolving a network failure problem using the discovery tool that identified new components in the network. A newly added host interface card was the culprit! This was done from a network operations center, without sending an engineer to the remote site.

There is also another side to the power of this awesome discovery tool, which was an experience of another network manager. He was once asked by one of the departments in the campus to shut off the discovery tool as it was flooding the network and degrading performance. Thus, powerful network management tools also need to be managed to avoid degradation of network performance. There are horror stories of the network coming down when turning on NMSs.

When asked what is the most benefit that he got out of an NMS, one of the managers answered that it is the consistency of administering, for example, configuring the network. This came across as an extremely interesting comment to me as I was once involved in automating the installation and maintenance of a telephone network. One of the operating telephone company managers, who helped in specifications then, commented that what we were trying to accomplish was an impossible task. He said that there were no standard operations procedures for the company that could be automated, and even in one single operations center, no two groups were following the same procedures! Believe it or not, the project was a success.

Let us now illustrate what an NMS could do by monitoring a subnetwork using a commercial NMS. The addresses of network components have been intentionally modified for security reasons.

Figure 4.1 shows a managed LAN that was discovered by an NMS. We show here only a subnetwork of a larger network managed by the NMS. As we mentioned above, an NMS can automatically discover any component in the network as long as the component has a management agent. The management agent could be as simple as a TCP/IP suite that responds to a ping by the NMS. However, agents in the modern network components are more sophisticated. We will study how NMS does an autodiscovery of elements in the network in Chapter 12. Let us accept for now that it has been accomplished somehow.

The managed subnetwork that we are discussing here is an Ethernet LAN that is shown below the backbone cloud in Figure 4.1. It consists of a router and two hubs and is connected to the backbone network. The LAN IP address is 172.16.46.1, and the two hub addresses have been configured as 172.16.46.2 and 172.16.46.3. The LAN IP address, 172.16.46.1, is the address assigned to the interface card in the router. Interface cards in the router and the interface card in each of the hubs are connected by a cat-5 cable forming the Ethernet LAN.

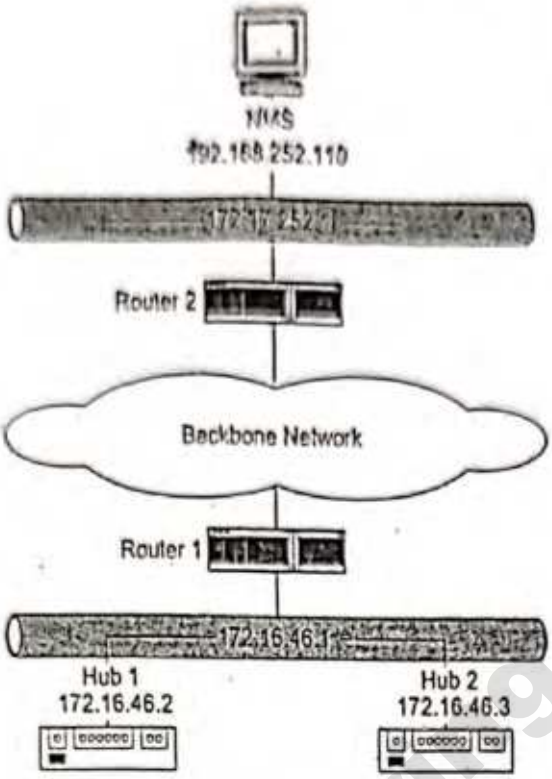


Figure 4.1 Managed LAN Network

The NMS, whose IP address is 192.168.252.110, is physically and logically located remotely from the 172.16.46.1 LAN. It is configured on the LAN 192.168.252.1 and is connected to the backbone network. Information system managers establish conventions to designate a network and a subnetwork. A 0 in the fourth decimal position of an IP address designates a network, and a subnetwork is designated with a 1 in the fourth position of the dotted decimal notation. Thus, 172.16.46.1 is a LAN subnetwork in the network 172.16.46.0.

Once network components have been discovered and mapped by the NMS, we can query and acquire information on system parameters and statistics on the network elements. Figure 4.2 presents system information on three network elements in the managed LAN gathered by the NMS sending specific queries asking for system parameters.

Figure 4.2(a) shows that the network element is designated by 172.16.46.2. No specific title or name has been assigned to it. System description indicates that it is a hub made by a 3Com vendor, with its model and software version. It also gives the system object ID and how long the system has been up without failure. The format of the System ID follows the format shown in Figure 3.10 with the 3Com node under enterprises node being 43. The last three node numbers, 1.8.5, following 43 describe the private MIB of 3Com. The *System Up Time* indicates that the system has been operating without failure for over 286 days. The number in parenthesis is in units of one hundredths of a second. Thus, the hub designated by the IP address 172.16.46.2 has been up for 2,475,380,437 hundredths of seconds, or for 286 days, 12 hours, 3 minutes, 24.37 seconds. System Description and System Object ID are factory set and the rest are user settable.

Figure 4.2(b) shows similar parameters for the second hub, 172.16.46.3, on the LAN. Figure 4.2(c) presents system information sent by the router on the network to the NMS's queries. The system name for the router has been configured and hence the query received the response of the name, router1. gatech.edu.

```
Title: System Information : 172.16.46.2
Name or IP Address: 172.16.46.2

System Name          :
System Description   : 3Com LinkBuilder FMS, SW version:3.02
System Contact       :
System Location      :
System Object ID     : .iso.org.dod.internet.private.enterprises.43.1.8.5
System Up Time       : (2475380 437) 286 days, 12:03:24.37
```

(a) System Information on 172.16.46.2 Hub

```
Title: System Information: 172.16.46.3
Name or IP Address: 172.16.46.3

System Name          :
System Description   : 3Com LinkBuilder FMS, SW version:3.12
System Contact       :
System Location      :
System Object ID     : .iso.org.dod.internet.private.enterprises.43.1.8.5
System Up Time       : (3146735182) 364 days, 4:55:51.82
```

(b) System Information on 172.16.46.3 Hub

```
Title: System Information: router1.gatech.edu
Name or IP Address: 172.16.252.1

System Name          : router1.gatech.edu
System Description   : Cisco Internetwork Operating System Software
                     : IOS (tm) 7000 Software (C7000-JS-M), Version
                     : 11.2(6),RELEASE SOFTWARE (ge1)
                     : Copyright (c) 1986-1997 by Cisco Systems, Inc.
                     : Compiled Tue 06-May-97 19:11 by kuong
System Contact       :
System Location      :
System Object ID     : iso.org.dod.internet.private.enterprises.cisco.ciscoProducts.
                       cisco 7000
System Up Time       : (315131795) 36 days, 11:21:57.95
```

(c) System Information on Router

Figure 4.2 System Information Acquired by an NMS

Figures 4.3(a), (b), and (c) present the data acquired by the NMS from the interface cards on the two hubs and the router, which are on LAN 172.16.46.1. They are addresses associated with each interface. At the top of each figure are the titles and IP address or name of the network interface card used by the NMS to access the network component. Thus, in Figure 4.3(a), the title and the name or IP address are 172.16.46.2. Note that the IP address 172.16.46.3 is the address as seen by the NMS traversing the router. In Figure 4.3(b), the IP address 172.16.46.3 is the access address of the second hub on the

Title: Addresses: 172.16.46.2
Name or IP Address: 172.16.46.2

Index	Interface	IP Address	Network Mask	Network Address	Link Address
1	3Com	172.16.46.2	255.255.255.0	172.16 46.0	0x08004E07C25C
2	3Com	192.168.101.1	255.255.255.0	192.168.101.0	<none>

(a) Addresses on 172.16.46.2 Hub Ports

Title: Addresses: 172.16.46.3
Name or IP Address: 172.16.46.3

Index	Interface	IP Address	Network Mask	Network Address	Link Address
1	3Com	172.16.46.3	255.255.255.0	172.16 46.0	0x08004E0919D4
2	3Com	192.168.101.1	255.255.255.0	192.168.101.0	<none>

(b) Addresses on 172.16.46.3 Hub Ports

Title: System Information: router1.gatech.edu
Name or IP Address: 172.16.252.1

Index	Interface	IP Address	Network Mask	Network Address	Link Address
23	LEC.1.0	192.168.3.1	255.255.255.0	192.168.3.0	0x00000C3920B4
25	LEC.3.9	192.168.252.1	255.255.255.0	192.168.252.0	0x00000C3920B4
13	Ethernet2/0	172.16.46.1	255.255.255.0	172.16.46.0	0x00000C3920AC
16	Ethernet2/3	172.16.49.1	255.255.255.0	172.16.49.0	0x00000C3920AF
17	Ethernet2/4	172.16.52.1	255.255.255.0	172.16.52.0	0x00000C3920B0
9	Ethernet1/2	172.16.55.1	255.255.255.0	172.16.55.0	0x00000C3920A6
2	Ethernet 0/1	172.16.56.1	255.255.255.0	172.16.56.0	0x00000C39209D
15	Ethernet2/2	172.16.57.1	255.255.255.0	172.16.57.0	0x00000C3920AE
8	Ethernet1/1	172.16.58.1	255.255.255.0	172.16.58.0	0x00000C3920A5
14	Ethernet2/1	172.16.60.1	255.255.255.0	172.16.60.0	0x00000C3920AD

(c) Addresses on Router Ports (Partial List)

Figure 4.3 Addresses Information Acquired by an SNMP NMS

172.16.46.1 LAN. Figure 4.3(c) shows the title and name or IP address as router1.gatch.edu. By using a network lookup command, the IP address of router1.gatech.edu can be recognized as 172.16.252.1. This is the backbone interface address of the router and is the interface on the router as seen by the NMS traversing the backbone network.

In Figures 4.3(a), (b), and (c), we notice that there are six columns of data. The first column is the index, which identifies the row in the matrix. Each row is a collection of various addresses associated with an interface. The second column describes the port id. For example, hubs 1 and 2 have 3Com cards in them. Column 2 of Figure 4.3(c) identifies the card and the port of the interface. For example, the

row with index 2 identifies Ethernet 0 card/port 1. The IP address of the interface card is presented in the third column of the matrix. The IP address in the third column and the network mask address in the fourth column are "and-ed" in modula-2 arithmetic to obtain the network address presented in the fifth column. This implies that all packets destined for network address 172.16.46.0 will be accepted by hub 1. The sixth and the last column in Figure 4.3, the link address, contains the MAC address. In the first row of Figure 4.3(a), 08004E07C25C is the MAC address of the hub 1 interface card. Link addresses in the second rows of Figures 4.3(a) and (b) are presented as "none," as they are non-LAN interfaces.

The Figure 4.3(c) matrix has many rows, as it is a router with many interface cards, each with multiple ports. For example, each Ethernet card has four physical ports. LEC 1.0 and LEC 3.9 are ATM LAN Emulation Card interfaces.

4.2 HISTORY OF SNMP MANAGEMENT

SNMP management began in the 1970s. Internet Control Message Protocol (ICMP) was developed to manage Advanced Research Project Agency NETwork (ARPANET). It is a mechanism to transfer control messages between nodes. A popular example of this is Packet Internet Groper (PING), which is part of the TCP/IP suite now. We learned to use this in the exercises in Chapter 1. PING is a very simple tool that is used to investigate the health of a node and the robustness of communication with it from the source node. It started as an early form of network-monitoring tool.

ARPANET, which started in 1969, developed into the Internet in the 1980s with the advent of UNIX and the popularization of client- server architecture. Data were transmitted in packet form using routers and gateways. TCP/IP-based networks grew rapidly, mostly in defense and academic communities and in small entrepreneurial companies taking advantage of the electronic medium for information exchange. National Science Foundation officially dropped the name ARPANET in 1984 and adopted the name Internet. Note that the Internet is spelled with a capital I and is limited to a TCP/IP-based network. An Internet Advisory Board (IAB) was formed to administer Internet activities, which are covered in the next section.

With the growth of the Internet, it became essential to have the capability to remotely monitor and configure gateways. Simple Gateway Monitoring Protocol (SGMP) was developed for this purpose as an interim solution. The IAB recommended the development of SNMP that is a further enhancement of SGMP. Even SNMP management was intended to be another interim solution, with the long-term solution being migration to the OSI standard CMIP/CMIS. However, due to the enormous simplicity of SNMP and its extensive implementation, it has become the de facto standard. SNMPv2 was developed to make it independent of the OSI standard, as well as adding more features. SNMPv2 has only partially overcome some of the limitations of SNMP. The final version of SNMPv2 was released without one of its major enhancements on its security feature due to strong differences in opinion. SNMPv3 addresses the security feature.

4.3 INTERNET ORGANIZATIONS AND STANDARDS

4.3.1 Organizations

We mentioned in the previous section that the IAB recommended the development of SNMP. The IAB was founded in 1983 informally by researchers working on TCP/IP networks. Its name was formally

changed from the Internet Advisory Board to the Internet Architecture Board in 1989 and was designated with the responsibility to manage two task forces—the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF).

The IRTF is tasked to consider long-term research problems in the Internet. It creates focused, long-term, and small research groups working on topics related to Internet protocols, applications, architecture, and technology.

With the growth of the Internet, the IETF organization has grown to be the protocol engineering, development, and standardization arm of the IAB.

The Internet Network Information Center (InterNIC) is an organization that maintains several archives that contain documents related to the Internet and the IETF activities. They include among other documents, Request for Comments document (RFC), Standard RFC (STD), and For Your Information RFC (FYI). The latter two are subseries of RFCs (more about these in the next section).

The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols. It is the clearinghouse to assign and coordinate use of numerous Internet protocol parameters. The Internet protocol suite contains numerous parameters, such as Internet addresses, domain names, autonomous system numbers (used in some routing protocols), protocol numbers, port numbers, MIB object identifiers (including private enterprise numbers), and many others. The common use of Internet protocols by the Internet community requires that these values be assigned uniquely. It is the task of IANA to make those unique assignments as requested and to maintain a registry of currently assigned values.

4.3.2 Internet Documents

Originally, RFC was just what the name implies—Request for Comments. Early RFCs were messages between ARPANET architects about how to resolve certain problems. Over the years, RFC has become more formal. It had reached the point that they were being cited as standards, even when they were not. To help clear up some confusion, there are now two special subseries within the RFCs: FYIs and STDs. The "For Your Information" RFC subseries was created to document overviews and topics that are introductory. Frequently, FYIs are created by groups within the IETF User Services Area. The STD RFC subseries was created to identify those RFCs that do in fact specify Internet standards. Every RFC, including FYIs and STDs, has an RFC number by which they are indexed and can be retrieved. FYIs and STDs have FYI numbers and STD numbers, respectively, in addition to RFC numbers. This makes it easier for a new Internet user, for example, to find all of the helpful, informational documents by looking for the FYIs among all the RFCs. If an FYI or STD is revised, its RFC number will change, but its FYI or STD number will remain constant for ease of reference.

RFC documents are available in public libraries and can be accessed via the Internet. Some sources that are in the public domain to access RFC and other Internet documents are:

ftp://ftp.internic.net/rfc
ftp://nic.mil/rfc
ftp.nic.it
http://nic.internic.net/

A novice to SNMP management could easily be confused as to which RFC document refers to what, namely, SMI, MIB, and SNMP, etc. It is confusing because the management field and associated documents are continuously evolving. Figure 4.4 portrays a high-level view of various document paths and

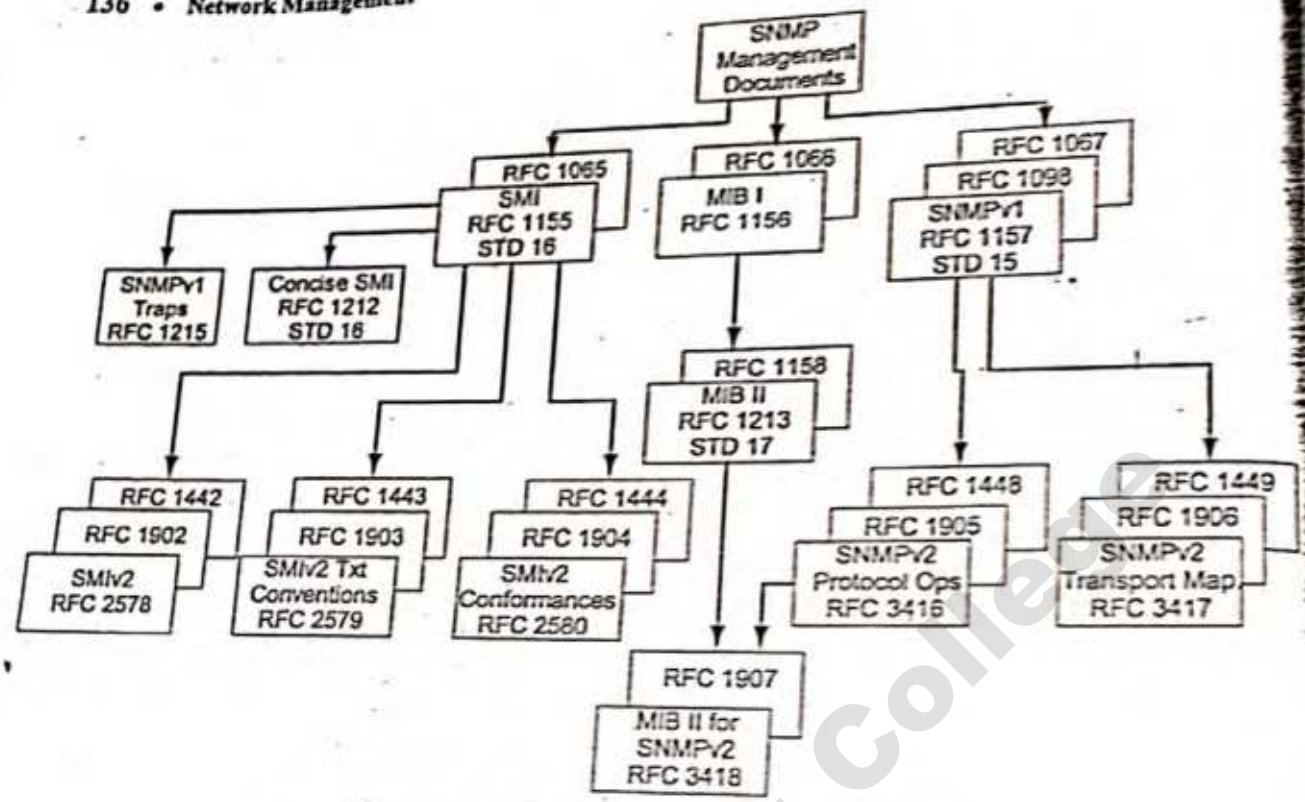


Figure 4.4 SNMP Document Evolution

documents that are relevant to SNMPv1 and SNMPv2. Documents associated with SNMPv3 will be described in Chapter 7. It is not intended to be a complete list, but to identify major core documents. There are three series of RFC and STD documents. They are: SMI, MIB, and SNMP Protocol. There are three standard documents, STD 15, 16, and 17 that have been approved by the IETF. STD 15/RFC 1157 defines the SNMP protocol. RFCs 1905, on protocol operations, and 1906, on transport mappings, are expanded updates of RFC 1157. These have been updated to RFC 1448 and RFC 1449 and subsequently, with the evolution of SNMPv3, to RFC 3416 and RFC 3417, respectively. In Figure 4.4, RFCs in the back of the cascades are earlier versions of the draft that have become obsolete. For example, RFC 1448 has been replaced by RFC 1905.

Structure of Management Information (SMI) forms the contents of RFC 1155, shown in Figure 4.4. A more concise version of SMI is given in RFC 1212 and is a supplement to RFC 1155. They both comprise STD 16 document. RFC 1155 did not address trap events, which is covered in RFC 1215.

SMIv2 is next in the evolution of SMI specifications, which are covered as STD 58 with the three documents RFC 2578–RFC 2580 describing SMIv2 data definition language, textual conventions, and conformance, respectively.

MIB has gone through a few iterations. RFC 1213/STD 17 is the version that is currently in use. It is backward compatible with MIB I specified in RFC 1156, which is obsolete now. Legacy systems that have implemented MIB I can continue to be used with MIB II implementation.

SNMP protocol has gone through modification and is part of SNMPv2. RFC 1907 is an early version of MIB II for SNMPv2 and the latest version is RFC 3418, which has gone through only minor changes from RFC 1907.

4.4 SNMP MODEL

We described an example of a managed network in Section 4.1. We saw that numerous management functions were accomplished in that example. We will now address how this is done in SNMP management. An NMS acquires a new network element through a management agent or monitors the ones it has acquired. There is a relationship between manager and agent. Since one manager is responsible for managing the designated functions of many agents, it is hierarchical in structure. The infrastructure of the manager–agent and the SNMP architecture that it is based on form the organization model.

Information is transmitted and is received by both the manager and the agent. For example, when a new network element with a built-in management agent is added to the network, the discovery process in the network manager broadcasts queries and receives positive response from the added element. The information must be interpreted both semantically and syntactically by the agent and the manager. We covered the syntax, ASN.1, in Section 3.7. Definition of semantics and syntax form the basis of the information model. We present a detailed definition of a managed object, rules for the SMI, and a virtual information database, MIB, which groups managed objects and provides a relational framework.

Communication between the manager and agents has to happen before information can be exchanged. The TCP/IP protocol suite is used for the transport mechanism. SNMP is defined for the application layer protocol and will be presented in Chapter 5.

Functions and services are not explicitly addressed in SNMP management. Security management is covered in the administration model as part of communication. Services are covered as part of SNMP operations.

The organization model, which has gone through an evolutionary process, is described in the next section.

4.5 ORGANIZATION MODEL

The initial organization model of SNMP management is a simple two-tier model. It consists of a network agent process, which resides in the managed object, and a network manager process, which resides in the NMS and manages the managed object. This is shown in Figure 4.5(a). Both the manager and the agent are software modules. The agent responds to any management system that communicates with it using SNMP. Thus, multiple managers can interact with one agent as shown in Figure 4.5(b)

We can question the need of multiple managers in a system when it is easy to monitor all objects in a network with standard messages. However, to configure a system in detail, more intimate knowledge of the object is needed, and hence an NMS provided by the same vendor would have more capabilities than another vendor's NMS. Thus, it is common practice to use an NMS to monitor a network of multiple

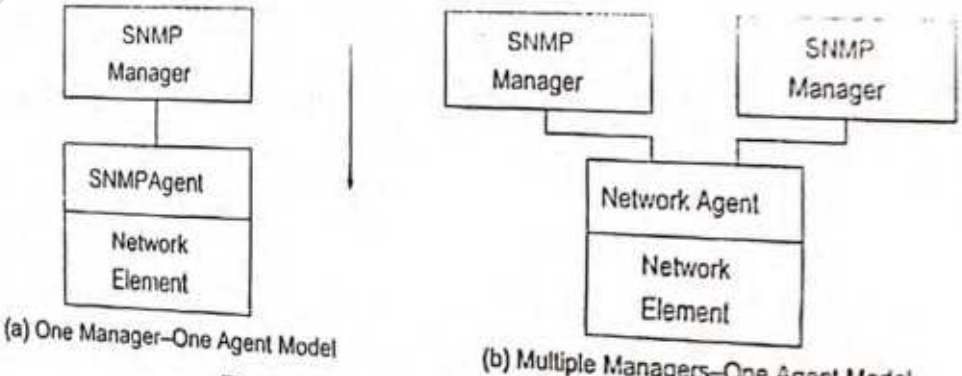


Figure 4.5 Two-Tier Organization Model

vendor products, and several vendors' NMSs to configure respective network elements. Further, during fault tracking, a vendor's NMS can probe in more depth the source of failure—even to the level of identification of a component on a printed circuit board.

In two-tier models, the network manager receives raw data from agents and processes them. It is sometimes beneficial for the network manager to obtain pre-processed data. For example, we may want to look at traffic statistics, such as input and output packets per second, at an interface on a node as a function of time. Alternatively, we may want to get the temporal data of data traffic in a LAN. Instead of the network manager continuously monitoring events and calculating the information—for example, data rate—an intermediate agent called Remote Monitoring (RMON) is inserted between the managed object and the network manager. This introduces a three-tier architecture as shown in Figure 4.6. The network manager receives data from managed objects, as well as from the RMON agent about the managed objects. The RMON function, implemented in a distributed fashion on the network, has greatly increased the centralized management of networks.

A pure SNMP management system consists of SNMP agents and SNMP managers. However, an SNMP manager can manage a network element, which does not have an SNMP agent. Figure 4.7 shows

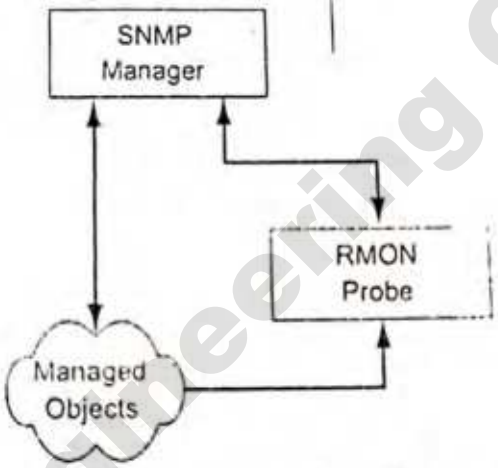


Figure 4.6 Three-Tier Organization Model

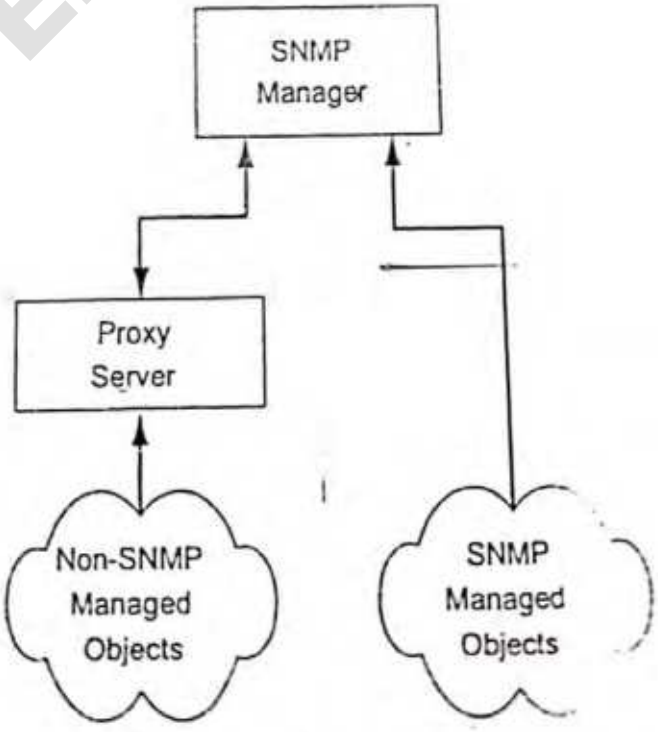


Figure 4.7 Proxy Server Organization Model

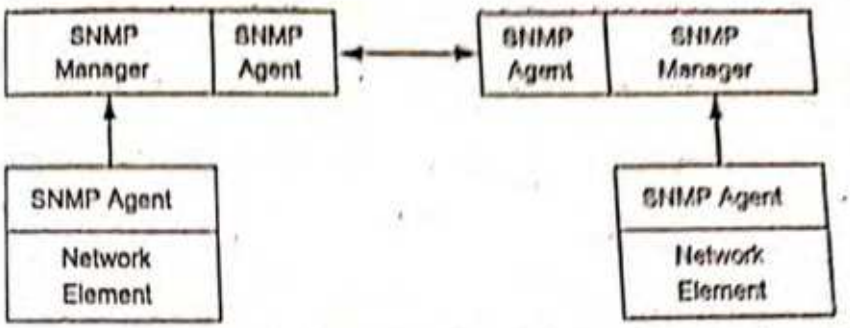


Figure 4.8 NMS Behaving as a Manager and an Agent

the organizational model for this case. This application occurs in many situations, such as legacy systems management, telecommunications management network, managing wireless networks, etc. In all these cases, they are part of an overall network that have to be managed on an integrated basis. As an example in a legacy case, we may want to manage outside plant and customer premises equipment for a Hybrid Fiber Coax (HFC) access system in broadband services to home. There are amplifiers on the outside cable plant, which do not have SNMP agents built in them. The outside cable plant uses some existing cable technology and has monitoring tools built into it, as for example transponders that measure various amplifier parameters. Information from the amplifiers could be transmitted to a central (head end) location using telemetry facilities. We can have a proxy server at the central location that converts data into a set that is SNMP compatible and communicates with the SNMP manager.

An SNMP management system can behave as an agent as well as a manager. This is similar to client–server architecture, where a host can function as both a server and a client (see Figure 1.8). In Figure 4.6, RMON, while collecting data from network objects, performs some functions (network monitoring) of a network manager. However, pre-processed data by RMON may be requested by the network manager or sent unsolicited by RMON to the network manager to integrate with the rest of the network data and to display it to the user. In the latter situation, RMON acts as a network agent. Another example of a system acting as both an agent and a manager is when two NMSs managing two autonomous networks exchange information with each other when the networks are connected via a gateway. This model is presented in Figure 4.8 and is applicable to two telecommunication service providers managing their respective wide area networks. To provide end-to-end service to customers, service providers may need to exchange management information between them.

4.6 SYSTEM OVERVIEW

Now that we have learned the relationship between the network (management) agent and manager and the different ways they can be configured, let us consider SNMP management from a system point of view. We have opted to do this prior to discussing details of the other three models—information, communication, and functional, because it would help us to understand them better if we have the big picture first.

Figure 4.9 shows SNMP network management architecture. It portrays the data path between the manager application process and the agent application process via the four transport function protocols—UDP, IP, Data Link Control (DLC), and Physical (PHY). The three application layers above the transport layer are integrated in the SNMP process.

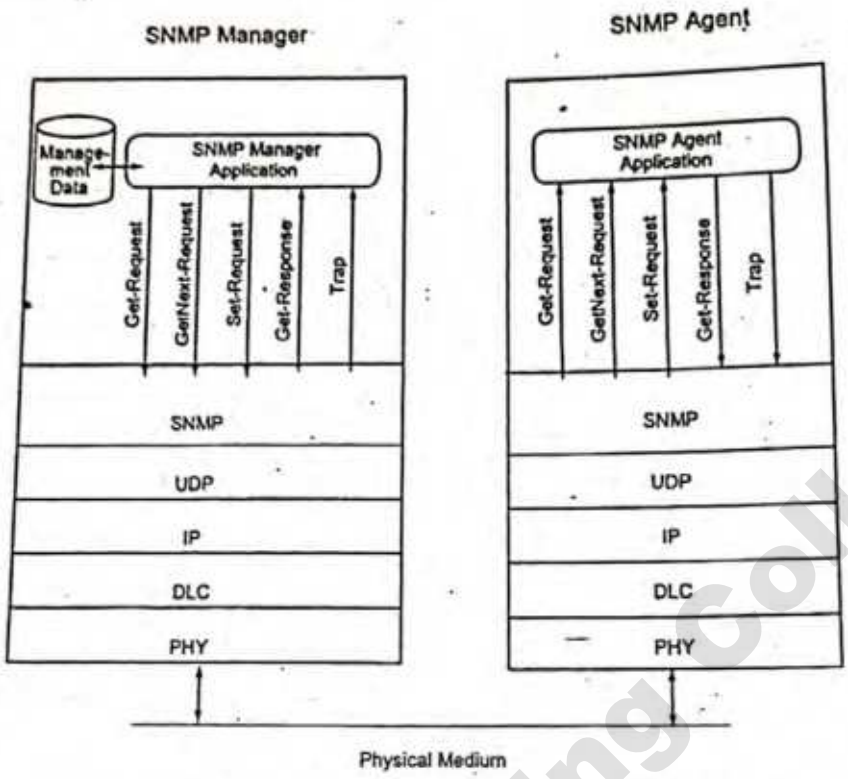


Figure 4.9 SNMP Network Management Architecture

As we stated in Chapter 1, the Internet is only concerned with the TCP/IP suite of protocols and does not address layers above or below it. Thus, layers 1 (PHY) and 2 (DLC) in the transport layers can be anything of users' choice. In practice, SNMP interfaces to the TCP/IP with UDP as the transport layer protocol.

RFC 1157 describes SNMP system architecture. It defines SNMP "by which management information for a network element may be inspected or altered by logically remote users." Two companion RFCs are *RFC 1155*, which describes the structure and identification of management information, and RFC 1156, which addresses the information base that is required for management.

As the name implies, SNMP protocol has been intentionally designed to be simple and versatile; this surely has been accomplished as indicated by its success. Communication of management information among management entities is realized through exchange of just five protocol messages. Three of these (get-request, get-next-request, and set-request) are initiated by the manager application process. The other two messages, get-response and trap, are generated by the agent process. Message generation is called an event. In the SNMP management scheme, the manager monitors the network by polling the agents as to their status and characteristics. However, efficiency is increased by agents generating unsolicited alarm messages, i.e., traps. We will summarize the messages here and describe structures associated with their Packet Data Units (PDUs) later. RFC 1157 defines the original specifications.

The get-request message is generated by the management process requesting the value of an object. The value of an object is a scalar variable. System group parameters in Figure 4.2 are single-instance values and are obtained using the get-request message.

The get-next-request, or simply called get-next, is very similar to get-request. In many situations, an object may have multiple values because of multiple instances of the object. For example, we saw in

Figure 4.3 that an interface could have multiple addresses associated with a given row. Another exam is the routing table of a router, which has multiple values (instances) for each object. In such situatio get-next-request obtains the value of the next instance of the object.

The set-request is generated by the management process to initialize or reset the value of an ob variable. The configuration parameters in Figure 4.2 that are settable can be set using the set-requ message.

The get-response message is generated by an agent process. It is generated only on receipt o get-request, get-next-request, or set-request message from a management process. The get-respo process involves filling the value of the requested object with any success or error message associa with the response.

The other message that the agent generates is trap. A trap is an unsolicited message generated an agent process without any message or event arriving from the manager process. It occurs whe observes the occurrence of a preset parameter in the agent module. For example, a node can send tr when an interface link goes up or down. Or, if a network object has a threshold value set for a parame such as the maximum number of packets queued up, a trap could be generated and transmitted by agent application whenever the threshold is crossed in either direction.

The SNMP manager, which resides in the NMS, has a database that polls managed objects for ma agement data. It contains two sets of data—one on information about the objects, MIB, and a second the values of the objects. These two are often confused with each other. MIB is a virtual data (inform tion) base and is static. In fact, it needs to be there when an NMS discovers a new object in the netwo It is compiled in the manager during implementation. If information about the managed object is n in the manager, it could still detect the object but would mark it as unidentifiable. This is because t discovery process involves a broadcast PING command by the NMS and responses to it from netwo components. Thus, a newly added network component would respond if it has a TCP/IP stack th normally has a built-in ICMP. However, the response contains only the IP address. MIB needs to implemented in both the manager and the agent to acquire the rest of the information, such as the syste group information shown in Figure 4.2.

The second database is dynamic and contains measured values associated with the object. This a true database. While MIB has a formalized structure, the database containing actual values can implemented using any database architecture chosen by the implementers.

It is worth noting in Figure 4.9 that the SNMP manager has a database, which is the physical database and the SNMP agent does not have a physical database. However, both have MIBs, which are compile into the software module and are not shown in the figure.

4.7 INFORMATION MODEL

The information model deals with SMI and MIB, which are discussed in the following subsections.

4.7.1 Introduction

Figure 4.9 shows the information exchange between the agent and the manager. In a managed network, there are many managers and agents. For information to be exchanged intelligently between manager and agent processes, there has to be common understanding on both the syntax and semantics. The syntax used to describe management information is ASN.1 and a general introduction to it was given in Chapter 3. In this section, we will address SNMP-specific syntax and semantics of management information.

We discussed the types of messages in the previous section and will discuss more in Chapter 5 when we consider the communication model. In this section we will address the specification and organizational aspects of managed objects. This is called the Structure of Management Information, SMI, and is defined in RFC 1155. Specifications of managed objects and the grouping of, and relationship between, managed objects are addressed in the MIB [RFC 1213].

There are generic objects that are defined by IETF and can be managed by any SNMP-compatible NMS. Objects that are defined by private vendors, if they conform to SMI defined by RFC 1155 and MIB specified by RFC 1213, can also be managed by SNMP-compatible NMSs. There are other RFCs that address specialized network objects, such as FDDI [RFC 1285], OSPF [RFC 1253], ATM [RFC 1695], etc. Private vendor objects are specified in private MIBs provided by vendors for their specific products.

4.7.2 Structure of Management Information

A managed object can be considered to be composed of an object type and an object instance, as shown in Figure 4.10. SMI is concerned only with the object type and not the object instance. That is, the object instance is not defined by SMI. For example, Figures 4.2(a) and (b) present data on two 3Com hubs. They are both identical hubs, except for a minor software release difference. The object types associated with both hubs are represented by the identical object ID, iso.org.dod.internet.private.enterprises.43.1.8.5. Hub 1 with an IP address 172.16.46.2 is an instance of the object.

Figure 4.11 shows the situation where there are multiple instances of an object type. In Figures 4.2(a) and (b), hub 1 with an IP address 172.16.46.2 and hub 2 with an IP address 172.16.46.3 are two instances of the object.

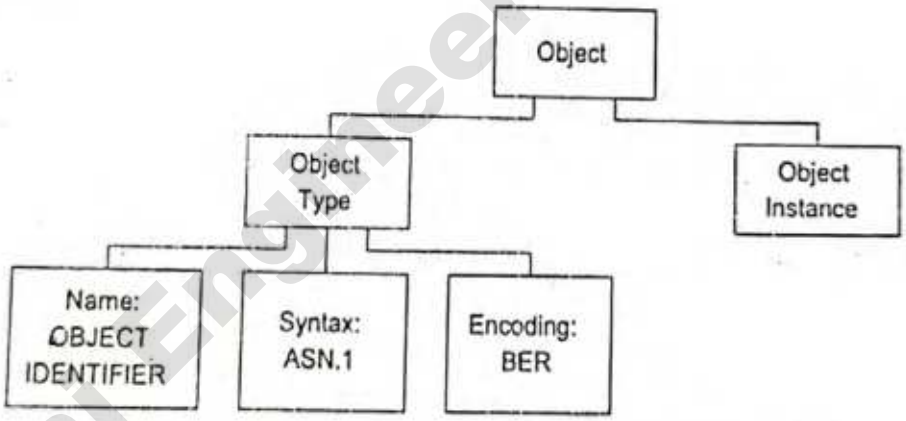


Figure 4.10 Managed Object: Type and Instance

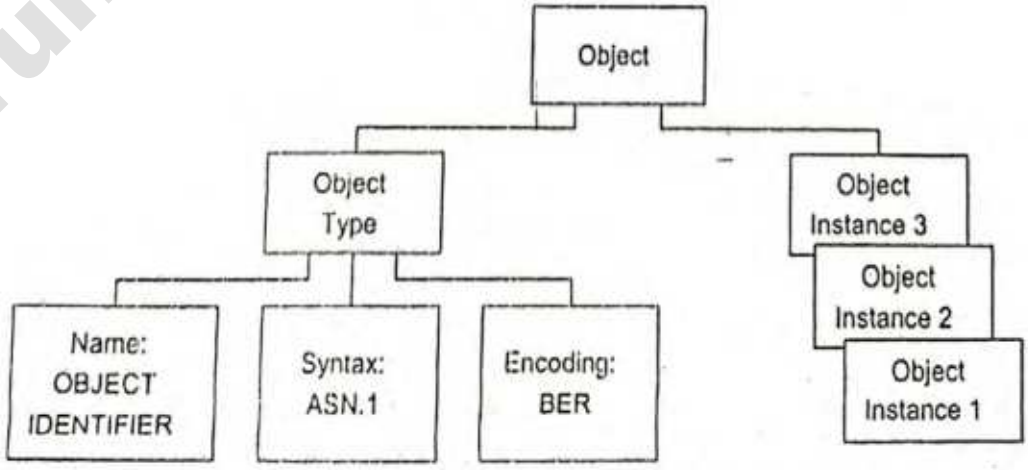


Figure 4,11 Managed Object: Type with Multiple Instances

A managed object need not be just a network element, it could be any object. For example, the Internet as an organization has an object name, "internet," with OBJECT IDENTIFIER 1.3.6.1. Of course, there can only be one instance of it! Thus, a managed object is only a means of identifying an object, whether it is physical or abstract.

The object type, which is a data type, has a name, a syntax, and an encoding scheme as discussed in Section 3.7. The name is represented uniquely by a descriptor and an object identifier. The syntax of an object type is defined using the abstract data structure ASN.1. Basic encoding rules (BER) have been adopted as the encoding scheme for transfer of data types between agent and manager processes, as well as between manager processes. We will next discuss each of these for SNMP-managed objects in detail.

Names. Every object type, i.e., every name, is uniquely identified by a DESCRIPTOR and an associated OBJECT IDENTIFIER. DESCRIPTOR and OBJECT IDENTIFIER are in uppercase since they are ASN.1 keywords. The DESCRIPTOR defining the name is mnemonic and is all in lowercase letters—at least it begins with lowercase letters, as we just described the Internet object as "internet." Since it is mnemonic and should be easily readable, uppercase letters can be used as long as they are not the beginning letter. For example, the object IP address table is defined as *ipAddrTable*. OBJECT IDENTIFIER is a unique name and number in the Management Information Tree (MIT), as we discussed in Section 3.4.1. We will henceforth use the term Management Information Base (MIB) for the Internet MIT. Thus, the Internet MIB has its OBJECT IDENTIFIER 1.3.6.1, as shown in Figure 3.5. It can also be defined in a hybrid mode, for example,

```
internet OBJECT IDENTIFIER ::= {iso org(3) dod(6) 1 }.
```

Information inside the curly brackets can be represented in various ways. This is shown in Figure 4.12. We can use any combination of the unique name and the unique node number on the management tree.

Any object in the Internet MIB will start with the prefix 1.3.6.1 or *internet*. For example, there are four objects under the *internet* object. These four objects are defined as:

```
directory       OBJECT IDENTIFIER ::= {internet 1}
mgmt            OBJECT IDENTIFIER ::= {internet 2}
experimental    OBJECT IDENTIFIER ::= {internet 3}
private         OBJECT IDENTIFIER ::= {internet 4}
```

The first line in this example states that the object, directory, is defined as the first node under the object internet. The four subnodes under the "internet" node are shown in Figure 4.13. We will discuss objects in the MIB tree in the next section.

The *directory*(1) node is reserved for future use of OSI Directory in the Internet. The *mgmt*(2) node is used to identify all IETF recommended and IAB-approved subnodes and objects. As of now the only

```
internet OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) internet(1) }
internet OBJECT IDENTIFIER ::= { 1 3 6 1 }
internet OBJECT IDENTIFIER ::= { iso org  dod internet }
internet OBJECT IDENTIFIER ::= { iso org  dod(6) internet(1) }
internet OBJECT IDENTIFIER ::= { iso(1) org(3) 6 1 }
```

Figure 4.12 Different Formats of Declaration of OBJECT IDENTIFIER

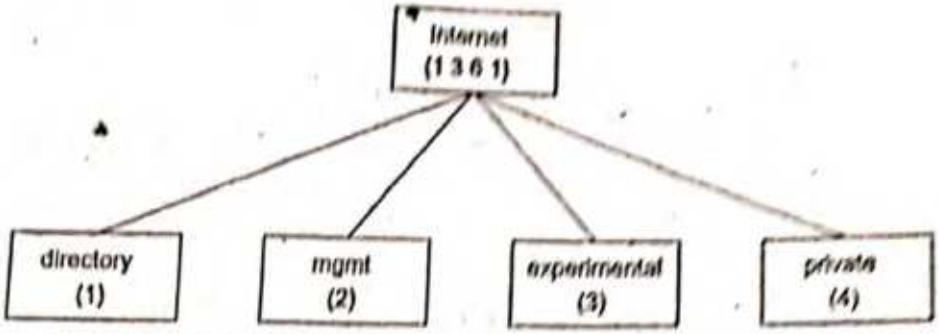


Figure 4.13 Subnodes under Internet Node in SNMPv1

node connected directly to {internet 2} is *mib-2*. As we said earlier, MIB-2 is a superset of MIB-1, and hence *mib-2* is the only node under {mgmt} as shown below:

mib-2 OBJECT IDENTIFIER ::= {mgmt 1}

The *experimental*(3) node was created to define objects under IETF experiments. For example, if IANA has approved a number 5 for an experimenter, we would use the OBJECT IDENTIFIER {experimental 5}.

The last node is *private*(4). This is a heavily used node. Commercial vendors can acquire a number under enterprises(1), which is under the private(4) node. Thus, we have

enterprises OBJECT IDENTIFIER ::= {private 1}

or

enterprises OBJECT IDENTIFIER ::= {1 3 6 1 4 1}

Figure 4.14 shows an example of four commercial vendors—Cisco, HP, 3Com, and Cabletron which are registered as nodes 9, 11, 43, and 52, respectively, under enterprises(1). Nodes under any of these nodes are entirely left to the discretion of the vendors.

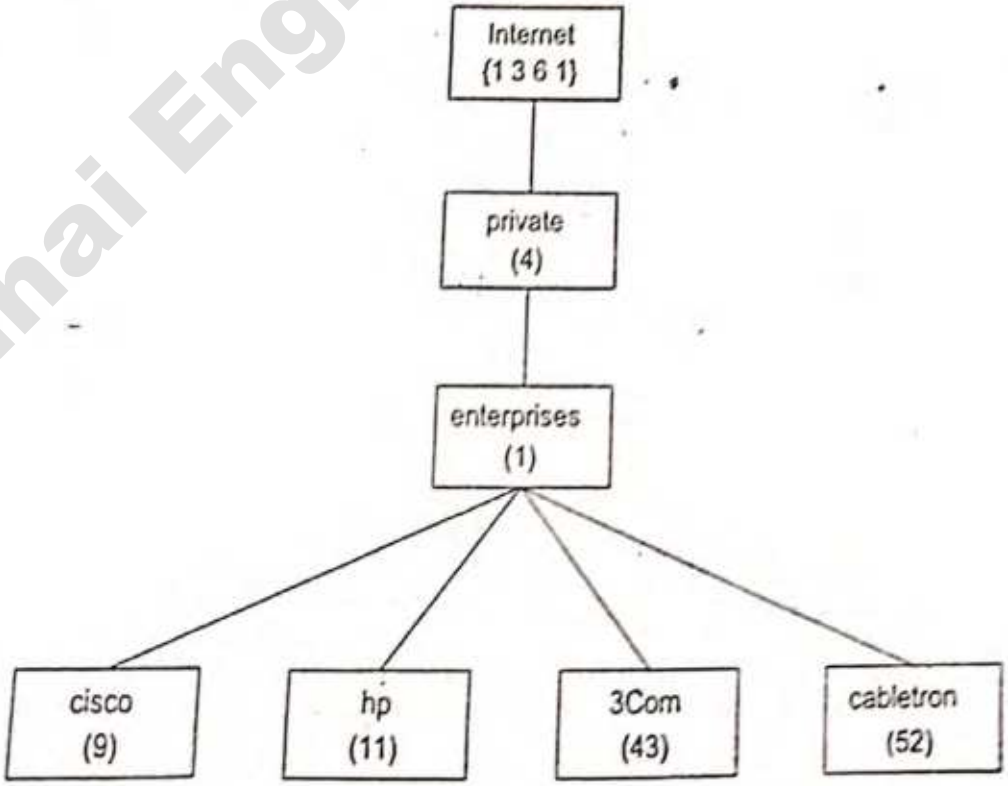


Figure 4.14 Private Subtree for Commercial Vendors

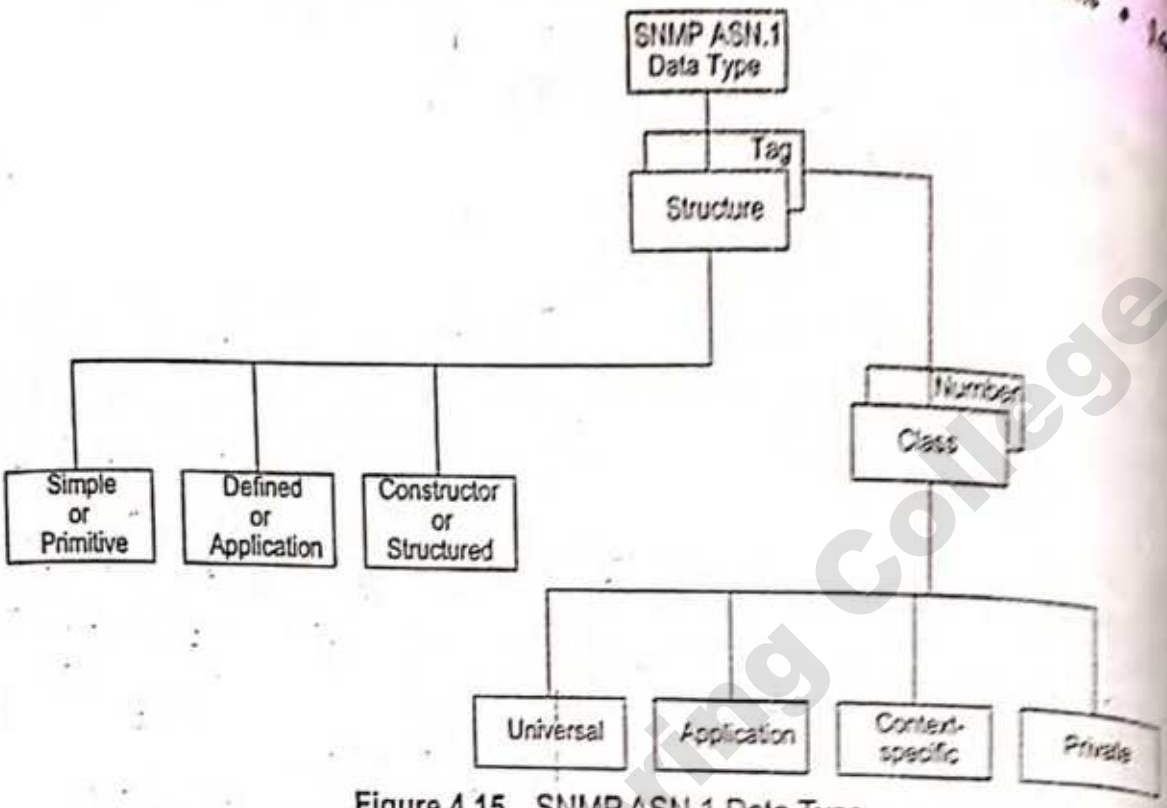


Figure 4.15 SNMP ASN.1 Data Type

Syntax. ASN.1 syntax that was introduced in Section 3.7 is used to define the structure of object types. Not all constructs of ASN.1 are used in TCP/IP-based SNMP management. Figure 4.15 shows the TCP/IP-based ASN.1 data type. It is very similar to Figure 3.15, but only has three categories under structure.

The three structural types shown in Figure 4.15 are simple, constructor, and defined types, as defined in RFC 1155. Other common terms used for these are primitive (or atomic), structured, and application types, respectively, as shown in Figure 3.9. The tagged type is not explicitly used in TCP/IP management although the IMPLICIT and EXTERNAL keywords are utilized for derived application data types.

SNMP ASN.1 data types are listed in Table 4.1. All data types except SEQUENCE and SEQUENCE OF are called base types.

Primitive or simple types are atomic in nature and are: INTEGER, OCTET STRING, OBJECT IDENTIFIER, and NULL. These are also referred to as non-aggregate types.

INTEGER has numerous variations based on the sign, length, range, and enumeration. The reader is referred to Perkins and McGinnis [1997] for a detailed presentation on the subject. When the integer value is restricted by a range, it is called a subtype, as presented in the comments column of Table 4.1, as INTEGER (n1..nN).

The data type ENUMERATED was specified in Section 3.6.2 as a special case of INTEGER data type. In SNMP management, it is specified as INTEGER data type with labeled INTEGER values. The following example of error-status in GetResponse associated with GetRequest-PDU illustrates the use of it. Each enumerated INTEGER has a name associated with it:

```
error-status INTEGER {
        noError(0)
        tooBig(1)
        genErr(5)
        authorizationError(16)
```

Table 4.1 SNMP-based ASN.1 Data Type Structure

STRUCTURE	DATA TYPE	COMMENTS
Primitive types	INTEGER	Subtype INTEGER (n1..nN)
		Special case: Enumerated INTEGER type
	OCTET STRING	8-bit bytes binary and textual data
		Subtypes can be specified by either range or fixed
	OBJECT IDENTIFIER	Object position in MIB
	NULL	Placeholder
Defined types	NetworkAddress	Not used
	IpAddress	Dotted decimal IP address
	Counter	Wrap-around, non-negative integer, monotonically increasing, max 2^32 -1
	Gauge	Capped, non-negative integer, increase or decrease
	TimeTicks	Non-negative integer in hundredths of second units
	Opaque	Application-wide arbitrary ASN.1 syntax, double-wrapped OCTET STRING
Constructor types	SEQUENCE	List maker
	SEQUENCE OF	Table maker

Any non-zero value indicates the type of error encountered by the agent in responding to a manager's message. As a convention, the value 0 is not permitted in the response message. Thus, a noError message is filled with NULL.

The OCTET STRING data type is used to specify either binary or textual information that is 8 bits long. Just as in INTEGER data type, a subtype in OCTET STRING can be specified. In fact, the subtype value can either be ranged, fixed, or a choice between them. Some examples of the subtype are:

```
OCTET STRING (SIZE 0..255)
OCTET STRING (SIZE 8)
OCTET STRING (SIZE 4 | 8)
OCTET STRING (SIZE 0..255 | 8)
```

The combination keyword OBJECT IDENTIFIER, as we discussed before, is the object position in the MIB. The fourth primitive type listed in Table 4.1 is NULL and is also a keyword. SNMPv1 keywords are listed in Table 4.2.

The second category of data types shown in Figure 4.15 and Table 4.1 consists of **defined types.** These are application-specific data types, and are also SNMP-based types. They are defined using primitive types. The primitive types used are NetworkAddress (not used in SNMP management), IpAddress, Counter, Gauge, and TimeTicks. The base type, Opaque, is used to specify octets of binary information. It is intended for adding new base types to extend SNMP SMI. Other application-wide data types can be constructed as long as they are IMPLICITly defined using these application data types.

Table 4.2 SNMPv1 Keywords

ACCESS
BEGIN
CHOICE
Counter
DEFINITIONS
DEFVAL
DESCRIPTION
END
ENTERPRISE
FROM
Gauge
IDENTIFIER
IMPORTS
INDEX
INTEGER
IpAddress
NetworkAddress
OBJECT
OBJECT-TYPE
OCTET
OF
Opaque
REFERENCE
SEQUENCE
SIZE
STATUS
STRING
SYNTAX
TRAP-TYPE
TimeTicks
VARIABLES

NetworkAddress is a choice of the address of the protocol family. For us, it is the TCP/IP-base Internet family, which uses the base type IpAddress.

IpAddress is the conventional four groups of dotted decimal notation of IPv4; for example, 190.146.252.255. The 32-bit string is designated as OCTET STRING of length 4 in network byte order.

Counter is an application-wide data type and is a non-negative integer. It can only increase in value up to a maximum of $2^{32}-1$ (4,294,967,295) and then wraps around starting from 0. The counter type is useful for defining values of data types that continually increase, such as input packets received on an interface or output packet errors on an interface.

The data type **Gauge** is also a non-negative integer, but its value can move either up or down. It pegs at its maximum value of $2^{32}-1$ (4,294,967,295). Gauge is used for data types whose value increases or decreases, such as the number of interfaces that are active in a router or hub.

TimeTicks is a non-negative integer and measures time in units of hundredths of a second. Its value indicates in hundredths of a second the number of units of time between the current instant and the time it was initialized to 0. The maximum value is $2^{32}-1$ (4,294,967,295). The system up time in Figure 4.2 is an example of this.

Opaque is an application-wide data type that supports the capability to pass arbitrary ASN.1 syntax. It is used to create more data types based on previously defined data types. This is extensively used in private vendors defining new data types in their products. When it is encoded, it is double wrapped, meaning the TLV (tag, length, and value) for the new definition is wrapped around the TLV of the previously defined type. Its size is undefined in SNMPv1, which causes some problem in its implementation. It is limited in SNMPv2.

The Opaque data type can be defined both IMPLICITly and EXPLICITly. By use of EXTERNAL type, encoding other than ASN.1 may be used in opaquely encoded data.

The third and last type of structure shown in Figure 4.15 is **constructor** or **structured type**. SEQUENCE and SEQUENCE OF are the only two constructor data types in Table 4.1 that are not base types. They are used to build lists and tables. Note that the constructs SET and SET OF, which are in ASN.1, are not included in the SNMP-based management syntax. SEQUENCE is used to build a list and SEQUENCE OF is used to build a table. We can conceptualize the list as values in a row of a table.

The syntax for list is

SEQUENCE { <type1>, <type2>,..., <typeN> }

where each type is one of ASN.1 primitive types.

The syntax for table is

SEQUENCE OF <entry>

where <entry> is a list constructor.

Illustrations of building list and table are shown in Figures 4.16(a) and (b). Figure 4.16(a) shows the object *ipAddrEntry* as an entry that is created from a list of objects. The list of objects in Figure 4.16(a) is 1 through 5 in the table. They are all basic types and each row of an object has the object name, OBJECT IDENTIFIER and *ObjectSyntax*. For example, object 1 on row 1 is the IP address defined as *ipAdEntAddr*. It has an OBJECT IDENTIFIER {ipAddrEntry 1} and syntax IpAddress. Note that there are two data types (ObjectSyntax) in the table, namely IpAddress and INTEGER. Thus, data types can be mixed in building a list. However, they are all basic data types and not constructor types.

The sixth object in the table is the object *ipAddrEntry* and is made up of the list of the first five objects. Construction for that is a SEQUENCE data type structure as shown. In Figure 4.16(a), the object *ipAdEntReasmMaxSize* has the syntax INTEGER(0..65535), which denotes that it is a subtype and the integer can take on values in the range from 0 to 65535.

Figure 4.16(b) shows the seventh object, *ipAddrTable*. It is node 20 under ip node and has a SEQUENCE OF construct. The *ipAddrTable* table is made up of instances of *ipAddrEntry* object.

Encoding. SNMPv1 has adopted BER with its TLV for encoding information to be transmitted between agent and manager processes. We covered this in Section 3.8 and illustrated a few ASN.1 data types. SNMP data types and tags are listed in Table 4.3. Encoding rules for various types follow.

OBJECT IDENTIFIER is encoded with each subidentifier value encoded as an octet and concatenated in the same order as in the object identifier. Since a subidentifier could be longer than an octet length, the

	Object	OBJECT IDENTIFIER	ObjectSyntax
1	ipAdEntAddr	{ipAddrEntry 1}	IpAddress
2	ipAdEntIfIndex	{ipAddrEntry 2}	INTEGER
3	ipAdEntNetMask	{ipAddrEntry 3}	IpAddress
4	ipAdEntBcastAddr	{ipAddrEntry 4}	INTEGER
5	ipAdEntReasmMaxSize	{ipAddrEntry 5}	INTEGER
6	ipAddrEntry	{ipAddrTable 1}	SEQUENCE

```
List:  IpAddrEntry  ::=
           SEQUENCE  {
                ipAdEntAddr              IpAddress
                ipAdEntIfIndex           INTEGER
                ipAdEntNetMask           IpAddress
                ipAdEntBcastAddr         INTEGER
                ipAdEntReasmMaxSize      INTEGER (0..65535)
                }
```

(a) Managed Object IpAddrEntry as a List

	Object Name	OBJECT IDENTIFIER	Syntax
7	ipAddrTable	{ip 20}	SEQUENCE OF

```
Table:  IpAddrTable ::=
           SEQUENCE OF IpAddrEntry
```

(b) Managed Object ipAddrTable as a Table

Figure 4.16 Example of Building a List and a Table for a Managed Object

Table 4.3 SNMP Data Types and Tags

TYPE	TAG
OBJECT IDENTIFIER	UNIVERSAL 6
SEQUENCE	UNIVERSAL 16
IpAddress	APPLICATION 0
Counter	APPLICATION 1
Gauge	APPLICATION 2
TimeTicks	APPLICATION 3
Opaque	APPLICATION 4

most significant bit (8th bit) is set to 0, if the subidentifier is only one octet long. The 8th bit is set
the value that requires more than one octet and indicates more octet(s) to follow. An exception
rule of one or more octets for each subidentifier is the specification of the first two subidentifi

example, *iso(1)* and *standard(3)* {1 3}, are coded as 43 in the first octet of the value. As an illustration, let us consider the object identifier *internet* {1 3 6 1}. The first octet of the TLV is the UNIVERSAL 6 tag, and the second octet defines the length of the value, which consists of three octets (43, 6, and 1). Thus, the encoded format is:

00000110 00000011 00101011 00000110 00000001

IP Address is encoded as straight octet strings. Counter, Gauge, and TimeTicks are coded as integers. Opaque is OCTET STRING type.

4.7.3 Managed Objects

In Chapter 3 we briefly looked at the perspective of a managed object in an SNMP management model and compared it to the OSI model. We will now specify in detail the SNMP data type format that would serve the basis for defining managed objects. We will address managed objects in the MIB in Section 4.7.4.

Structure of Managed Objects. Managed object, as we saw in Section 3.4.2, has five parameters. They are textual name, syntax, definition, access, and status as defined in RFC 1155. For example, *sysDescr* is a data type in the MIB that describes a system. Specifications for the object that describes a system are given in Figure 4.17.

As we notice in Figure 4.17, the **textual name** for an object type is mnemonic and is defined as OBJECT DESCRIPTOR. It is unique and is made up of a printable string beginning with a lowercase letter, *sysDescr*, in our example. OBJECT DESCRIPTOR defines only the object type, which is a data type. We will henceforth use the term *object type* and not *data type* when referring to a managed object. OBJECT DESCRIPTOR does not specify instances of a managed object. Thus, it describes what type of object it is and not the occurrence or instantiation of it, as we pointed out in Section 4.7.2. In Figures 4.2(a) and (b), the system description for the two hubs is 3Com LinkBuilder FMS with an appropriate software version. They both could be of the same software version and hence could be identical. Identification of each instance is left to the specific protocol that is used, and is not part of the specifications of either SMI or MIB. Thus, instances of the two hubs in Figures 4.2(a) and (b) are identified with their respective IP addresses, 172.16.46.2 and 172.16.46.3.

Associated with each OBJECT DESCRIPTOR is an OBJECT IDENTIFIER, which is the unique position it occupies in the MIB. In Figure 4.17, *sysDescr* is defined by OBJECT IDENTIFIER {system 1}.

OBJECT:	
sysDescr:	{ system 1 }
Syntax:	DisplayString (SIZE (0..255))
Definition:	"A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating system, and networking software. It is mandatory that this contain only printable ASCII characters."
Access:	read-only
Status:	mandatory

Figure 4.17 Specifications for System Description

Syntax is the ASN.1 definition of the object type. The syntax of *sysDescr* is OCTET STRING.

A definition is an accepted textual description of the object type. It is a basis for the common language or semantics to be used by all vendors. It is intended to avoid confusion in the exchange of information between the managed object and the management system, as well as between various NMSs.

Access is the specification for the privilege associated with accessing the information. It is one of read-only, read-write, write-only, or not-accessible. The first two choices are obvious and the third choice, not-accessible, is applicable, for example, in specifying a table. We access the values of the entries in the table and not the table itself, and hence it is declared not-accessible. The access for *sysDescr* is read-only. Its value is defined by the system vendor during the manufacturing process.

Status specifies whether the managed object is current or obsolete. A managed object, once defined, can only be made obsolete and not removed or deleted. If it is current, the implementation of it is specified as either mandatory or optional. Thus, the three choices for status are mandatory, optional, and obsolete. The status for *sysDescr* is mandatory.

Related objects can be grouped to form an **aggregate object type**. In this case the objects that make up the aggregate object type are called subordinate object types. The **subordinate object type** could either be simple (primitive type) or an aggregate type. However, it should eventually be made up of simple object types.

Macros for Managed Objects. In order to encode the above information on a managed object to be processed by machines, it has to be defined in a formalized manner. This is done using macros. Figure 4.18(a) shows a macro where an object type is represented in a formal way [RFC 1155]. A macro always starts with the name of the type—in this case, OBJECT-TYPE—followed by the keyword MACRO, and then the definition symbol. The right side of the macro definition always starts with BEGIN and ends with END.

The body of the macro module consists of three parts: type notation, value notation, and supporting productions. TYPE NOTATION defines the data types in the module and VALUE NOTATION defines the name of the object. Thus, in the example of Figure 4.18, the notations SYNTAX, ACCESS, and STATUS define the data types Object Syntax, Access, and Status. The notation for value specifies the Object Name. Supporting productions in Figure 4.18 define the allowed values for access and status. Access can only be one of any of the four options: read-only, read-write, write-only, or not-accessible. Allowed values for Status are mandatory, optional, or obsolete.

```
OBJECT-TYPE MACRO ::=
BEGIN
        TYPE NOTATION ::= "SYNTAX" TYPE (TYPE ObjectSyntax)
                "ACCESS" Access
                "STATUS" Status
        VALUE NOTATION ::= value (VALUE ObjectName)
        Access ::= "read-only" | "read-write" | "write-only" | "not-accessible"
        Status ::= "mandatory" | "optional" | "obsolete"
END
```

(a) An OBJECT-TYPE Macro [RFC 1155]

Figure 4.18 Scalar OBJECT-TYPE Macro and Example

```
sysDescr OBJECT-TYPE
        SYNTAX Display String (SIZE (0..255))
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
                "A textual description of the entity. This value should include the full
                name and version identification of the system's hardware type,
                software operating system, and networking software. It is
                mandatory that this contain only printable ASCII characters."
::= {system 1 }
```

(b) A Scalar or Single Instance Macro: sysDescr [RFC 1213]

Figure 4.18 (continued)

Figure 4.18(b) [RFC 1213] shows the application of the macro to a scalar, single-instance managed object, *sysDescr*, which is one of the components of the system group in the MIB, as we shall see in the next section. Its OBJECT IDENTIFICATION is {system 1}. DESCRIPTION defines the textual description of the object.

Aggregate Object. An aggregate object is a group of related objects. Figure 4.19 shows an example of an aggregate managed object, *ipAddrTable*, which we briefly considered as an example of structured data type in Figure 4.16. This is the IP address table that defines the IP address for each interface of the managed object. Objects 1 through 5 represent simple data types that make up an entry in a table. The textual name of the entry is *ipAddrEntry*. Thus, object 1 with the OBJECT DESCRIPTOR, *ipAdEntAddr*, is the first element of the entry, *ipAddrEntry*, and is given the unique OBJECT IDENTIFICATION, {ipAddrEntry 1}. This represents the IP address and has the syntax IpAddress, a keyword listed in Table 4.2. The access privilege to it is read-only and every managed object and management system is required to implement it.

Object 2 is *ipAdEntIfIndex* and is the second subordinate object type of *ipAddrEntry*. It identifies the instance of occurrence of the entry in the table. It references the values of other elements associated with the interface for that entry occurrence. Although a single element is adequate to uniquely identify the occurrence of an entry in this table, we will see later that there could be more than one element needed in other tables. The syntax of *ipAdEntIfIndex* is INTEGER, a primitive data type. Access and status are read-only and mandatory.

Objects 3, 4, and 5, *ipAdEntNetMask*, *ipAdEntBcastAddr*, and *ipAdEntReasmMaxSize*, respectively, specify the subnet mask, broadcast address information, and the size of the largest datagram. The definition for each describes what the object is.

Object 6 is the managed object, *ipAddrEntry*, which consists of the subordinate object types of 1 to 5 above. It describes the complete set of information consisting of the five fields needed for an entry in the IP interface address table. The syntax for *ipAddrEntry* is a SEQUENCE data type consisting of the five data types. Each data type is identified with its OBJECT DESCRIPTOR and syntax. Note that the access for *ipAddrEntry* is non-accessible. *ipAddrEntry* is itself a subordinate object type of the managed object, *ipAddrTable*. It is the first (and only) element of *ipAddrTable* and has the OBJECT IDENTIFICATION {*ipAddrTable* 1}.

OBJECT 1	ipAdEntAddr	{ IpAddrEntry 1 }
Syntax		IpAddress
Definition		"The IP address to which this entry's information pertains"
Access		read-only
Status		mandatory

OBJECT 2	ipAdEntIfIndex	{ ipAddrEntry 2 }
Syntax		INTEGER
Definition		"The index value which uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex. "
Access		read-only
Status		mandatory

OBJECT 3	ipAdEntNetMask	{ ipAddrEntry 3 }
Syntax		IpAddress
Definition		"The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and the host bits set to 0."
Access		read-only
Status		mandatory

OBJECT 4	ipAdEntBcastAddr	{ ipAddrEntry 4 }
Syntax		INTEGER
Definition		"The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1. This value applies to both the subnet and network broadcasts addresses used by the entity on this (logical) interface
Access		read-only
Status		mandatory

Figure 4.19 Specifications for an Aggregate Managed Object: ipAddrTable

OBJECT 5

ipAdEntR easmMaxSize		{ ipAddrEntry 5 }
Syntax	INTEGER (0..65535)	
Definition	"The size of the largest IP datagram which this entity can reassemble from incoming IP fragmented datagrams received on this interface."	
Access	read-only	
Status	mandatory	

OBJECT 6

ipA ddrEntry		{ ipAddrTable 1 }
Syntax	ipAddrEntry ::= SEQUENCE {	
ipAdEntAddr		IpAddress,
ipAdEntIfIndex		INTEGER,
IpAdEntNetMask		IpAddress,
IpAdEntBcastAddr		INTEGER,
ipAdEntReasmMaxSize		INTEGER (0..65535)
Definition	"The addressing information for one of this entitys'IP addresses."	
Access	not-accessible	
Status	mandatory	

OBJECT 7

ipAddrTable		{ ip 20 }
Syntax	SEQUENCE OF IpAddrEntry	
Definition	"The table of addressing information relevant to this entity's IP addresses."	
Access	not-accessible	
Status	m andatory	

Figure 4.19 (continued)

ipAddrTable is the OBJECT DESCRIPTOR for the IP address table, which has a unique place in the MIB tree with the OBJECT IDENTIFIER {ip 20}. We will see how the managed object *ip* group fits in the MIB tree in the next section. The syntax of *ipAddrTable* is the structure SEQUENCE OF the data type *ipAddrEntry*. Again, the access is not-accessible.

As an example of the use of the above specifications in a table, let us consider the following entry in an IP address table:

OBJECT 1	{ipAdEntAddr } = { internet "123.45.2.1".}
OBJECT 2	{ipAdEntIfIndex} = { "1" }
OBJECT 3	{ipAdEntNetMask} = { internet "255.255.255.0" }
OBJECT 4	{ipAdEntBcastAddr} = { "0" }
OBJECT 5	{ipAdEntReasmMaxSize} = { "12000" }

The value of *ipAdEntIfIndex* for this entry in the IP address table is equal to 1, and the IP address defining this interface is 123.45.2.1 using the Internet-specific protocol. The value associated with network mask is 255.255.255.0, with *ipAdEntBcastAddr* 0, and with the maximum size of the packet 12,000.

Figure 4.20 [RFC 1213] presents the macro for the IP address table, a multiple-instance presented in Figure 4.17. The text following "--" are comments and not encoded. The module starts at the highest level defining the *ipAddrTable*, then follows up with *ipAddrEntry* and finally defines the subordinate object types of *ipAddrEntry*. Note that there is an additional clause, INDEX, in the *ipAddrEntry* macro in Figure 4.20. This uniquely identifies the instantiation of the entry object type in the table. Thus,

```
-- the IP address table
-- The IP address table contains this entity's IP addressing information.

    ipAddrTable OBJECT-TYPE
      SYNTAX  SEQUENCE OF IpAddrEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
          "The table of addressing information relevant to this entity's IP addresses."
      ::= { ip 20 }

    ipAddrEntry OBJECT-TYPE
      SYNTAX  IpAddrEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
          "The addressing information for one of this entity's IP addresses."

      INDEX   { ipAdEntAddr }
      ::= { ipAddrTable 1 }

    IpAddrEntry ::=
      SEQUENCE {
        ipAdEntAddr
          IpAddress,
        ipAdEntIfIndex
          INTEGER,
        ipAdEntNetMask
          IpAddress,
        ipAdEntBcastAddr
          INTEGER,
        ipAdEntReasmMaxSize
          INTEGER (0..65535) }

    ipAdEntAddr OBJECT-TYPE
      SYNTAX  IpAddress
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
          "The IP address to which this entry's addressing information pertains."
```

Figure 4.20 Aggregate Managed Object Macro: ipAddrTable [RFC 1155]

```
        ::= { ipAddrEntry 1 }
        ipAdEntIfIndex OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
      "The index value which uniquely identifies the interface to which this entry is applicable. The
  interface identified by a particular value of this index is the same interface as identified by
  the same value of ifIndex."
        ::= { ipAddrEntry 2 }

        ipAdEntNetMask OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
      "The subnet mask associated with the IP address of this entry. The value of the mask is an
IP address with all the network bits set to 1 and all the host bits set to 0."
        ::= { ipAddrEntry 3 }

        ipAdEntBcastAddr OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
      "The value of the least-significant bit in the IP broadcast address used for sending datagrams
  on the (logical) interface associated with the IP address of this entry. For example, when the Internet
  standard all-ones broadcast address is used, the value will be 1. This value applies to both
  the subnet and network broadcasts addresses used by the entity on this (logical) interface."
        ::= { ipAddrEntry 4 }

        ipAdEntReasmMaxSize OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
      "The size of the largest IP datagram which this entity can reassemble from incoming IP frag-
mented datagrams received on this interface."
        ::= { ipAddrEntry 5 }
```

Figure 4.20 (continued)

ipAdEntAddr object uniquely identifies the instantiation. We will discuss this more in the next section on columnar objects.

We have so far presented the SMI as it was originally developed in RFC 1155. This helped us understand the two aspects of an object module: specifications and formal structure. Obviously, there is duplication in this. It was originally developed this way to eventually migrate to OSI specifications. However, with the reality that the OSI standards were not implemented and SNMP standards had been deployed extensively, the specifications and formal structure were combined into a concise definition of object macro, described in RFC 1212. It is presented in Figure 4.21.

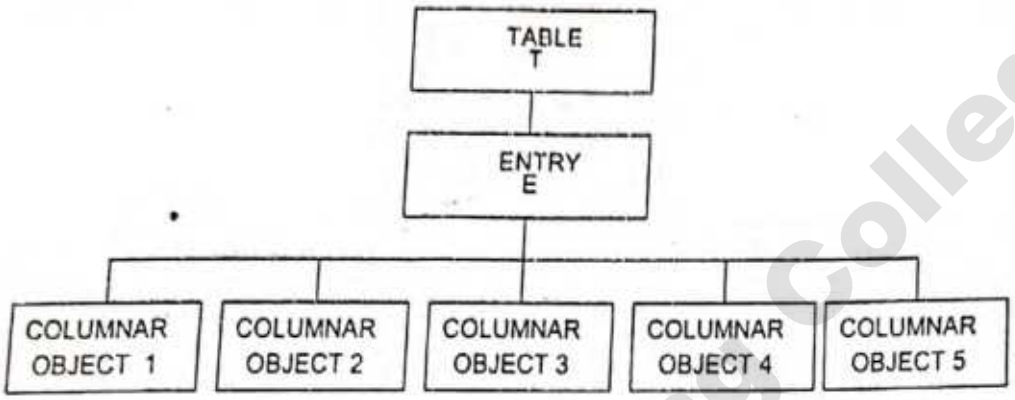
```
IMPORTS
        ObjectName FROM RFC1155-SMI
        DisplayString FROM RFC1158-MIB

OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::=
                                -- must conform to RFC 1155's ObjectSyntax
                "SYNTAX" type   (TYPE ObjectSyntax)
                "ACCESS" Access
                "STATUS" Status
                DescrPart
                ReferPart
                IndexPart
                DefValPart
    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only" | "read-write" | "write-only" | "not-accessible"
    Status ::= "mandatory" | "optional" | "obsolete" | "deprecated"
    DescrPart ::= "DESCRIPTION" value (description DisplayString) | empty
    ReferPart ::= "REFERENCE" value (reference DisplayString) | empty
    IndexPart ::= "INDEX" "{" IndexTypes "}" | empty
    IndexTypes ::= IndexType | IndexTypes "," IndexType
    IndexType ::=
                --if indexobject, use SYNTAX
                --value of the correspondent
                --OBJECT-TYPE invocation
            value (indexobject ObjectName)
                --otherwise use named SMI type
                -- must conform to IndexSyntax below
                | (type IndexType)
    DefValPart ::=
            "DEFVAL" "{" value (defvalue ObjectSyntax) "}" | empty
END

IndexSyntax ::=
    CHOICE {
            number  INTEGER (0..MAX),
            string          OCTET STRING,
            object          OBJECT IDENTIFIER,
            address         NetworkAddress,
            ipAddress       IpAddress
    }
```

Figure 4.21 OBJECT-TYPE Macro: Concise Definition [RFC 1212]

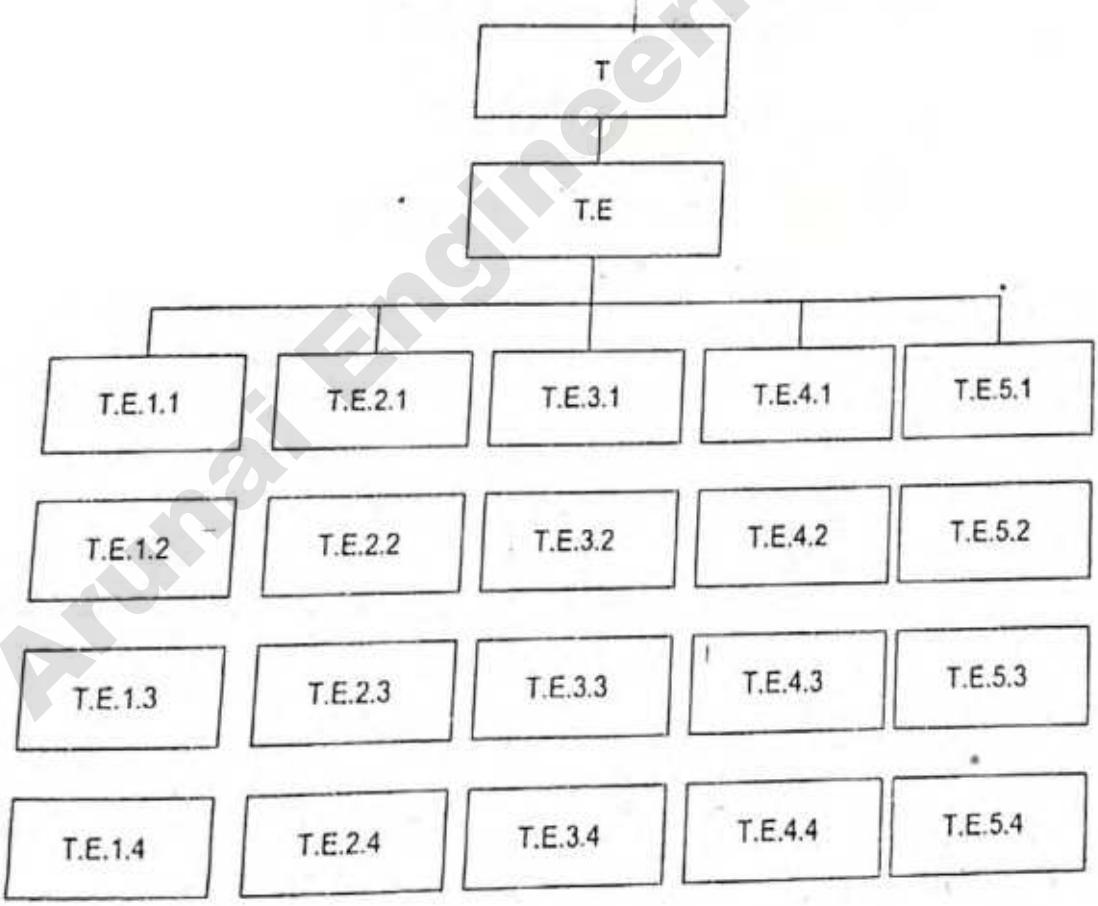
Note that there is the definition of imports from other modules. Also, there are additional cl ReferPart, IndexPart, and DefVal, and their associated value definitions. The REFERENCE cla a textual reference to the document from which the object is being mapped. The INDEX clause columnar object identifier, which as we said, will be discussed in the next section under colu objects. DEFVAL is the default value to the object, if applicable.

Aggregate Object as Columnar Object. The aggregate object that was discussed above has been formally defined as columnar objects in RFC 1212. SNMP operations apply exclusively to scalar operations. This means that a single scalar value is retrieved or edited on a managed object with any one operation. However, managed objects do have multiple instances within a system and need to be represented formally. An aggregate object type comprises one or more subtypes; and each subtype could have multiple instances, with a value associated with each instance.

It is convenient to conceptually define a tabular structure for objects that have multiple values, such as the IP address table. Such tables can have any number of rows including none, with each row containing one or more scalar objects. This is shown in Figure 4.22(a). Table T contains subordinate object Entry E



(a) Multiple-Instance Managed Object



(b) Example of a 5-Columnar Object with 4 Instances (Rows)

Figure 4.22 Numbering Convention of a Managed Object Table

that is a row in the table. Since the table is a SEQUENCE OF construction with entry E as components, there are multiple entries in the table; i.e., there are multiple rows in the table. Entry E is a SEQUENCE construct consisting of subordinate objects, columnar objects 1 through 5, in Figure 4.22(a).

Figure 4.22(b) shows a five-columnar object with four instances, i.e., four rows. It is important to note the convention used in denoting each object in the rows. The columnar objects in each row are denoted by the concatenation of the object identifier of the table, the entry, and then the object, and lastly by the row number. Note that the last two numbers are not like what we would normally think of as a row and column sequence in a matrix representation. It is more like column and row designation. Thus, the third occurrence (third row) of the fourth columnar object (fourth column) is T.E.4.3. The value for the row number is the value of the index of the table. For example, *ipAdEntAddr*, which is the IP address, is the index for the IP address table example shown in Figure 4.20. Hence, the value of *ipAdEntAddr* will determine the row of the table.

Let us apply this conceptual table to the IP address table example we have been following. This is shown in Figure 4.23. Figure 4.23(a) presents the detail of the columnar object, *ipAdEntBcastAddr*, which is the fourth columnar object under *ipAddrEntry*, which is a subordinate object of *ipAddrTable*. The OBJECT IDENTIFIER of the *ipAddrTable* in the MIB is 1.3.6.1.2.1.4.20. The *ipAddrEntry* is node 1 under it and *ipAdEntBcastAddr* is the fourth node under *ipAddrEntry*. Thus, the columnar object identifier of *ipAdEntBcastAddr* is {1.3.6.1.2.1.4.20.1.4}.

Figure 4.23(b) shows the tabular presentation of an IP address table. The table shows four rows and six columns. Each of the four rows in the IP address table indicates a set of values associated with each instance of *ifAddrEntry* in the table.

The first column in Figure 4.23(b) is the row number, which is added to the other five columns (column 2 through 6) that represent the five columnar objects of the IP address table. We have added the first column of the row number for easy explanation only; it is not part of the managed objects. The first columnar object *ipAdEntAddr* is in bold letters to indicate that it is the index for the table. As each row in an aggregate object table is uniquely identified by the INDEX clause of the OBJECT-TYPE macro, each row in our example is uniquely identified by indexing the value of *ipAdEntAddr*. The second row is the columnar object *ipAdEntIfIndex*. Note that *ipAdEntIfIndex*, which is the same as the *ifNumber* of the Interfaces group, is not an index, but just an object associated with each row of the table. The last three columns in Figure 4.23(b) represent the columnar objects *ipAdEntNetMask*, *ipAdEntBcastAddr*, and *ipAdEntReasmMaxSize*.

Figure 4.23(c) shows the representation of the object identifier associated with each instance. There are four instances illustrated in the figure. The first column is the columnar object identifier, the second column is the row number shown in Figure 4.23(b), and the last column is the object identifier for the instance of the columnar object. Let us first look at the first row of Figure 4.23(c). We want to represent the object identifier associated with the columnar object *idAdEntAddr* for the specific occurrence presented in the second row of Figure 4.23(b). The object identifier *ipAdEntAddr* in the first row of Figure 4.23(c) is its columnar object identifier 1.3.6.1.2.1.4.20.1.1. It is suffixed with the value of the table index field *ipAdEntAddr* 123.45.3.4. The resultant object identifier 1.3.6.1.2.1.4.20.1.1.123.45.3.4 is shown in the first row of the last column of Figure 4.23(c).

The second entry in Figure 4.23(c) illustrates the object identifier 1.3.6.1.2.1.4.20.1.2.165.8.9.25 for the columnar object *ipAdEntIfIndex* for the instance indicated in the third row of Figure 4.23(b). The third and fourth entries in Figure 4.23(c) illustrate the object identifier values of *ipAdEntBcastAddr* and *ipAdEntReasmMaxSize* for rows 1 and 4 of Figure 4.23(b), respectively.

ipAddrTable (1.3.6.1.2.1.4.20)
ipAddrEntry (1)
ipAdEntAddr (1)
ipAdEntIfIndex (2)
ipAdEntNetMask (3)
ipAdEntBcastAddr (4)
ipAdEntReasmMaxSize (5)

Columnar object ID of ipAdEntBcastAddr is (1.3.6.1.2.1.4.20.1.4):

iso org dod internet mgmt mib ip ipAddrTable ipAddrEntry ipAdEntBcastAddr
1 3 6 1 2 1 4 20 1 4

(a) Columnar Objects under ipAddrEntry

Row	ipAdEntAddr	ipAdEntIfIndex	ipAdEntNetMask	ipAdEntBcast Addr	ipAdEntReasmMax Size
1	123.45.2.1	1	255.255.255.0	0	12000
2	123.45.3.4	3	255.255.0.0	1	12000
3	165.8.9.25	2	255.255.255.0	0	10000
4	9.96.8.138	4	255.255.255.0	0	15000

(b) Object Instances of ipAddrTable (1.3.6.1.2.1.4.20)

Columnar Object	Row # in (b)	Object Identifier
ipAdEntAddr 1.3.6.1.2.1.4.20.1.1	2	{1.3.6.1.2.1.4.20.1.1.123.45.3.4}
ipAdEntIfIndex 1.3.6.1.2.1.4.20.1.2	3	{1.3.6.1.2.1.4.20.1.2.165.8.9.25}
ipAdEntBcastAddr 1.3.6.1.2.1.4.20.1.4	1	{1.3.6.1.2.1.4.20.1.4.123.45.2.1}
ipAdEntReasmMaxSize 1.3.6.1.2.1.4.20.1.5	4	{1.3.6.1.2.1.4.20.1.5.9.96.8.138}

(c) Object ID for Specific Instances

Figure 4.23 Multiple-Instance Managed Object: ipAddrTable

The formalized definitions of SMI as presented in STD 16/RFC 1155 are shown in Figure 4.24. In addition to the definition of the object type macro, it also specifies the exports of names and object types, as well as the Internet MIB, which is addressed in the next section.

```
RFC1155-SMI DEFINITIONS ::= BEGIN

    EXPORTS -- EVERYTHING
        internet, directory, mgmt, experimental, private, enterprises,
        OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
        ApplicationSyntax, NetworkAddress, IpAddress, Counter, Gauge,
        TimeTicks, Opaque;

    -- the path to the root

    internet        OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }

    directory       OBJECT IDENTIFIER ::= { internet 1 }
    mgmt            OBJECT IDENTIFIER ::= { internet 2 }
    experimental    OBJECT IDENTIFIER ::= { internet 3 }
    private         OBJECT IDENTIFIER ::= { internet 4 }

    enterprises     OBJECT IDENTIFIER ::= { private 1 }

    -- definition of object types

    OBJECT-TYPE MACRO ::=
BEGIN
      TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                        "ACCESS" Access
                        "STATUS" Status
      VALUE NOTATION ::= value (VALUE ObjectName)

      Access ::= "read-only" | "read-write" | "write-only" | "not-accessible"
      Status ::= "mandatory" | "optional" | "obsolete"
END

    -- names of objects in the MIB

    ObjectName ::=
      OBJECT IDENTIFIER

    -- syntax of objects in the MIB

    ObjectSyntax ::=
      CHOICE {
        simple
          SimpleSyntax,

    -- Note that simple SEQUENCEs are not directly mentioned here to keep things
    simple (i.e., prevent misuse). However, application-wide types, which are IMPLIC-
    ITly encoded simple SEQUENCEs, may appear in the following CHOICE.

        application-wide
          ApplicationSyntax
      }
```

Figure 4.24 SMI Definitions [RFC 1155]

```
SimpleSyntax ::=
  CHOICE {
    number
      INTEGER,
    string
      OCTET STRING,
    object
      OBJECT IDENTIFIER,
    empty
      NULL
  }

ApplicationSyntax ::=
  CHOICE {
    address
      NetworkAddress,
    counter
      Counter,
    gauge
      Gauge,
    ticks
      TimeTicks,
    arbitrary
      Opaque

-- Other application-wide types, as they are defined, will be added here.
  }

-- application-wide types

NetworkAddress ::=
  CHOICE {
    internet
      IpAddress
  }
IpAddress ::=
  [APPLICATION 0]          -- in network-byte order
    IMPLICIT OCTET STRING (SIZE (4))
Counter ::=
  [APPLICATION 1]
    IMPLICIT INTEGER (0..4294967295)
Gauge ::=
  [APPLICATION 2]
    IMPLICIT INTEGER (0..4294967295)
TimeTicks ::=
  [APPLICATION 3]
    IMPLICIT INTEGER (0..4294967295)
Opaque ::=
  [APPLICATION 4]          -- arbitrary ASN.1 value,
    IMPLICIT OCTET STRING  -- "double-wrapped"
END
```

Figure 4.24 (continued)

4.7.4 Management of Information Base

As stated in Section 4.7.1, MIB-II specified in RFC 1213 is the current standard, STD 17. It is a superset of MIB-I or simply MIB, as it was then addressed in RFC 1156. We will present here MIB-II information. Both MIB-I and MIB-II can be implemented in SNMPv1. MIB is organized such that implementation can be done on an as-needed basis. The entire MIB does not have to be implemented in either the manager or the agent process.

Let us remember that MIB is a virtual information store (base). Managed objects are accessed via this virtual information base. Objects in the MIB are defined using ASN.1. In the previous section, we discussed the SMI, which defines the mechanism for describing these objects. The definition consists of three components: *name* (OBJECT DESCRIPTOR), *syntax* (ASN.1), and *encoding* (BER).

Objects defined in MIB-II have the OBJECT IDENTIFIER prefix:

```
mib-2        OBJECT IDENTIFIER ::= {mgmt 1}
```

MIB-II has an additional attribute to the status of a managed object. The new term is "deprecated." This term mandates the implementation of the object in the current version of MIB-II, but is most likely to be removed in future versions. For example, *atTable* is deprecated in MIB-II.

Object Groups. Objects that are related are grouped into object groups. Notice that this grouping is different from the grouping of object types to construct an aggregate object type. Object groups facilitate logical assignment of object identifiers. One of the criteria for choosing objects to be included in standards is that it is essential for either fault or configuration management. Thus, if a group is implemented in a system by a vendor, all the components are implemented, i.e., status is mandatory for all its components. For example, if the External Gateway Protocol (EGP) is implemented in a system, then all EGP group objects are mandatory to be present.

The MIB module structure consists of the module name, imports from other modules, and definitions of the current module. The basic ASN.1 structure is shown in Figure 4.25.

There are 11 groups defined in MIB-II. The tree structure is shown in Figure 4.26, and Table 4.4 presents the name, object identification (OID), and a brief description of each group. It can be observed that these groups are nodes under the MIB object *mib-2* whose OBJECT IDENTIFIER is 1.3.6.1.2.1.

The System group contains objects describing system administration. The Interfaces group defines interfaces of the network component and network parameters associated with each of those interfaces. The Address translation group is a cross-reference table between the IP address and the physical address. IP, ICMP, TCP, UDP, and EGP groups are the grouping of objects associated with the respective protocol of the system. The group, CMOT, is a placeholder for future use of the OSI protocol, CMIP over TCP/IP. The Transmission group was created as a placeholder for network transmission-related parameters and was a placeholder in RFC 1213. Numerous transmission systems and objects have been developed under this group since then. SNMP group is the communication protocol group associated with SNMP

```
<module name > DEFINITIONS ::= BEGIN
        <imports>
        <definitions>
END
```

Figure 4.25 MIB Module Structure

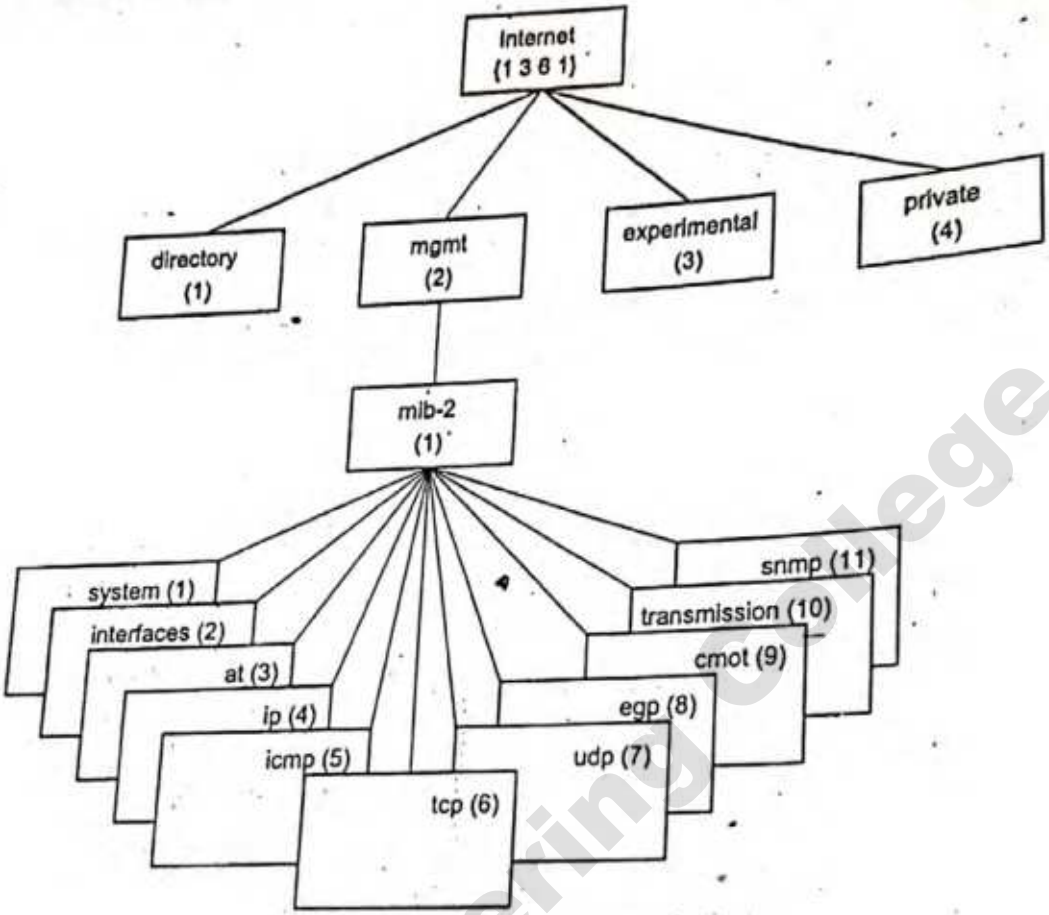


Figure 4.26 Internet MIB-II Group

Table 4.4 MIB-II Groups

GROUP	OID	DESCRIPTION (BRIEF)
system	mib-2 1	System description and administrative information
interfaces	mib-2 2	Interfaces of the entity and associated information
at	mib-2 3	Address translation between IP and physical address
ip	mib-2 4	Information on IP protocol
icmp	mib-2 5	Information on IGMP protocol
tcp	mib-2 6	Information on TCP protocol
udp	mib-2 7	Information on UDP protocol
egp	mib-2 8	Information on EGP protocol
cmot	mib-2 9	Placeholder for OSI protocol
transmission	mib-2 10	Placeholder for transmission information
snmp	mib-2 11	Information on SNMP protocol

management. We will now learn more about some of these groups. It should be noted that there are
groups defined under the Internet node, which we will address in Chapter 5.

The following sections describe details of each group except for CMOT, transmission, and SNM
The CMOT group is a placeholder and is not yet defined. The Transmission group is based on the
mission media underlying each interface of the system; the corresponding portion of the Transmis
group is mandatory for that system. The SNMP group will be addressed in Chapter 5 as part of
communication model.

Although there are many more groups in MIB-II, details on only the generic groups directly rela
to physical properties of basic network elements (System and Interfaces) and the managed obje
associated with Internet protocols (IP, TCP, and UDP) are presented here. They are intended to familiar
the reader quickly with how to read and interpret RFCs specifying MIBs. It is strongly recommend
that you refer to the RFC for detailed specifications on each group and understand the structure of ea
MIB group.

Some examples associated with managed objects in the group are presented along with a descripti
of the group in order to appreciate the significance of each MIB. In Chapter 9 we will learn to use t
SNMP command using SNMP tools and retrieve the values associated with managed objects.

System Group. The System group is the basic group in the Internet standard MIB. Its elements a
probably the most accessed managed objects. After an NMS discovers all the components in a netwo
or newly added components in the network, it has to obtain information on the system it discovere
such as system name, object ID, etc. The NMS will initiate the get-request command on the objects
this group for this purpose. Data on the systems shown in Figure 4.2 were obtained by the NMS usi
this group. The group also has administrative information, such as contact person and physical locati
that helps a network manager.

Implementation of the System group is mandatory for all systems in both the agent and the manag
It consists of seven entities, which are presented in Figure 4.27 and Table 4.5. The vendor of the equi
ment programs the system description (*sysDescr*) and OBJECT IDENTIFIER (*sysObjID*) during man
facturing. System up time is filled in hundredths of a second dynamically during operation. Netwo
management systems usually convert this into a readable format of days, hours, and minutes in the
presentation, as shown in Figure 4.2. Although system services (*sysServices*) object is mandatory to
implemented, most NMSs do not show the information automatically.

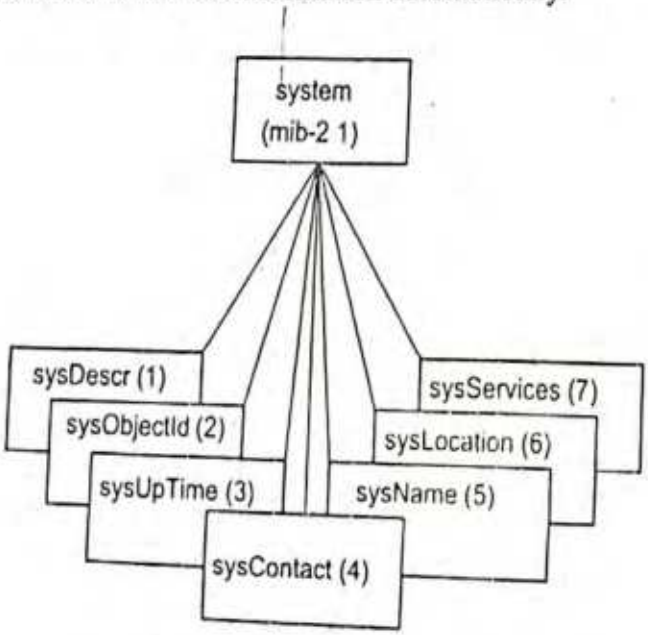


Figure 4.27 System Group

Table 4.5 System Group

ENTITY	OID	DESCRIPTION (BRIEF)
sysDescr	system 1	Textual description
sysObjectID	system 2	OBJECT IDENTIFIER of the entity
sysUpTime	system 3	Time (in hundredths of a second since last reset)
sysContact	system 4	Contact person for the node
sysName	system 5	Administrative name of the system
sysLocation	system 6	Physical location of the node
sysServices	system 7	Value designating the layer services provided by the entity

Interfaces Group. The Interfaces group contains managed objects associated with the interfaces of a system. If there is more than one interface in the system, the group describes the parameters associated with each interface. For example, if an Ethernet bridge has several network interface cards, the group would cover information associated with each interface. However, the Interfaces MIB contains only generic parameters. In the Ethernet example, there is more information associated with the Ethernet LAN, which is addressed in the MIB specifications of the particular medium, as in Definitions of Managed Objects for the Ethernet-like Interface types [RFC 2358]. An NMS would combine information obtained from various groups in presenting comprehensive data to the user.

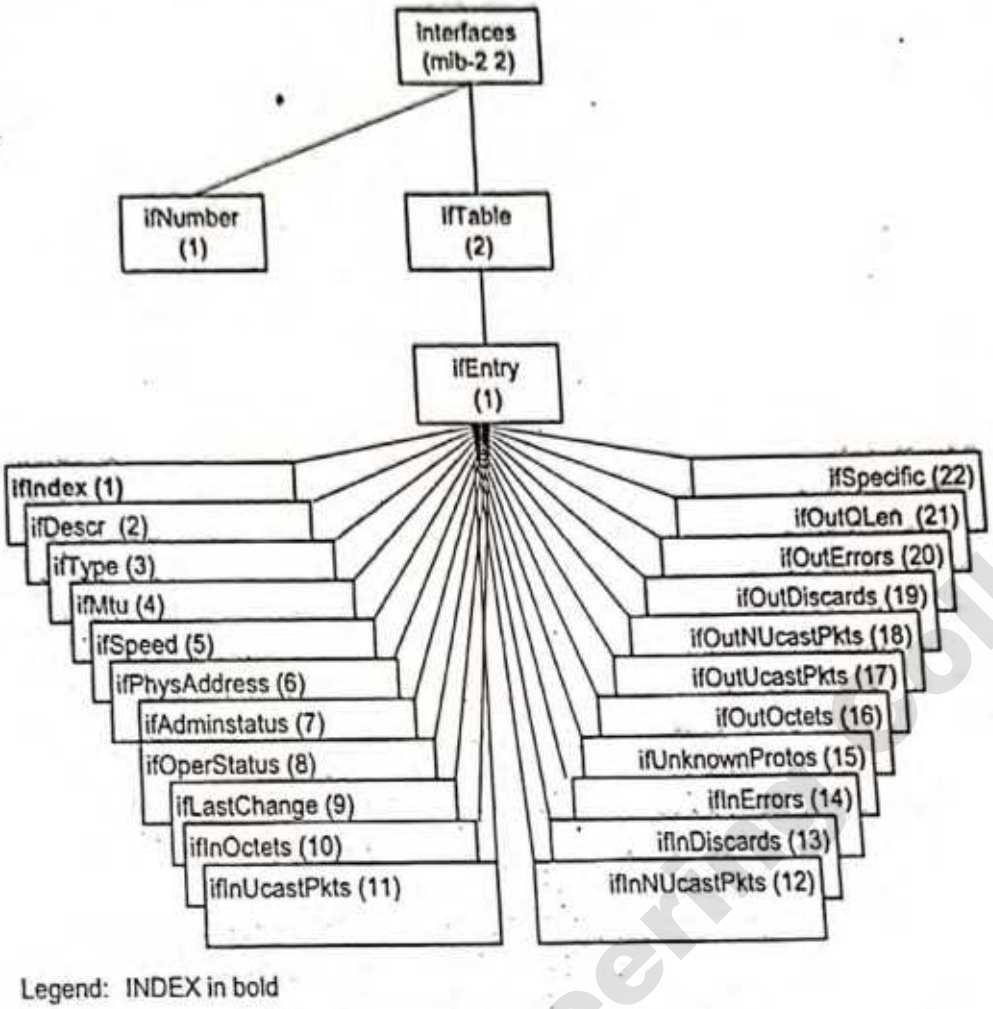
The Interfaces group specifies the number of interfaces in a network component and managed objects associated with each interface. Implementation of Interfaces group is mandatory for all systems. It consists of two nodes as shown in Figure 4.28 and Table 4.6. The number of interfaces of the entity is defined by *ifNumber*, and the information related to each interface is defined in the Interfaces table, *ifTable*.

Each interface in the Interfaces table can be visualized as being attached to either a subnetwork or a system. The term **subnetwork** is not to be confused with the term **subnet**, which refers to an addressing partitioning scheme in the Internet suite of protocols. The index for the table is just one entity, specified by *ifIndex*, as shown below in the definition of the *ifEntry* module under *ifTable*.

```
IfEntry  OBJECT-TYPE
         SYNTAX        ifEntry
         ACCESS        not-accessible
         STATUS        mandatory
         DESCRIPTION
                  "An interface entry containing objects at the subnetwork layer and below for a
                  particular interface."
         INDEX {ifIndex}
         ::= {ifTable 1}
```

The index is also shown in bold letters in the figure and the table.

The entity *ifType* describes the type of data link layer directly below the network layer. It is defined as an enumerated integer. Examples of these are: ethernet-csmacd(7), iso88025-tokenRing(9). See RFC 1213 for the specified type of standard interfaces.



Legend: INDEX in bold

Figure 4.28 Interfaces Group

The administrative and operational status that is indicated by object identifiers 7 and 8 should agree with each other when the system interface is functioning as administered.

Object identifiers 11–15 refer to the measurements (with counter syntax) on inbound traffic and object identifiers 16–21 to measurements on outbound traffic.

An example of use of Interfaces MIB would be to measure the incoming and the outgoing traffic rate on a given interface of an Ethernet hub. We can specify a port on an Ethernet network interface card by the value of *ifIndex* and query (*get-request*) the number of input unicast packets (*ifInUcastPkts*) and the number of output unicast packets (*ifOutUcastPkts*) every second. Remember that we get the reading of two counters, which are incremented with every packet coming in or going out of the port from the management agent associated with the port. We would then take the difference in the consecutive counter reading to derive the packet rate of traffic with time.

Interface Sublayers. One of the strengths of an IP network layer protocol is that it is designed to run over any network interface. IP considers any and all protocols it runs over as a single "network interface" layer. The Interfaces group defines a generic set of managed objects such that any network interface can be managed in an interface-independent manner through these managed objects. The Interfaces group provides the means for additional managed objects specific to particular types of network interface (e.g., a specific medium such as Ethernet or Time Division Multiplex (TDM) channels) to be

Table 4.6 Interfaces Group

ENTITY	OID	DESCRIPTION (BRIEF)
ifNumber	interfaces 1	Total number of network interfaces in the system
ifTable	interfaces 2	List of entries describing information on each interface of the system
ifEntry	ifTable 1	An interface entry containing objects at the subnetwork layer for a particular interface
ifIndex	ifEntry 1	A unique integer value for each interface
ifDescr	ifEntry 2	Textual data on product name and version
ifType	ifEntry 3	Type of interface layer below the network layer defined as an enumerated integer
ifMtu	ifEntry 4	Largest size of the datagram for the interface
ifSpeed	ifEntry 5	Current or nominal data rate for the interface in bps
ifPhysAddress	ifEntry 6	Interface's address at the protocol layer immediately below the network layer
ifAdminStatus	ifEntry 7	Desired status of the interface: up, down, or testing
ifOperStatus	ifEntry 8	Current operational status of the interface
ifLastchange	ifEntry 9	Value of sysUpTime at the current operational status
ifInOctets	ifEntry 10	Total number of input octets received
ifInUcastPkts	ifEntry 11	Number of subnetwork unicast packets delivered to a higher-layer protocol
ifInNUcastPkts	ifEntry 12	Number of non-unicast packets delivered to a higher-layer protocol
ifInDiscards	ifEntry 13	Number of inbound packets discarded irrespective of error status
ifInErrors	ifEntry 14	Number of inbound packets with errors
ifInUnknownProtos	ifEntry 15	Number of unsupported protocol packets discarded
ifOutOctets	ifEntry 16	Number of octets transmitted out of the interface
ifOutUcastPkts	ifEntry 17	Total number of unicast packets that higher-level layer requested to be transmitted
ifOutNUcastPkts	ifEntry 18	Total number of non-unicast packets that higher-level layer requested to be transmitted
ifOutDiscrds	ifEntry 19	Number of outbound packets discarded irrespective of error status
ifOutErrors	ifEntry 20	Number of outbound packets that could not be transmitted because of errors
ifOutQLen	ifEntry 21	Length of the output queue in packets
ifSpecific	ifEntry 22	Reference to MIB definitions specific to the particular media used to realize the interface

defined as extensions to the Interfaces group for media-specific management. Since the standardizatio
of MIB-II, many such media-specific MIB modules have been defined. Concurrently, the Interface
group has evolved to accommodate the additional managed objects that need to be specified in a da
link layer (DLL)—Layer 2.

DLL can be visualized, in general, as comprising several sublayers. These can either be horizontall
stacked or vertically sliced (or "stacked"), as shown in Figures 4.29(a) and (b), respectively. An exampl
of the former is an interface with PPP running over a High data rate Digital Subscriber Line (HDL
link, which uses an RS232-like connector. An example of the latter is a cable access link with a dow
stream channel and several upstream channels.

Since the simplistic model of a single conceptual row in the *ifTable* in the Interfaces group, an add
itional MIB group, *ifMIB* (mib-2 31) was created. This is not shown in Figure 4.26, which is the origin
MIB-II Group. It is shown in Figure 4.30. The first subnode of *ifMIB* is *ifMIBObjects*. There are oth
subnodes under *ifMIB* and the reader is referred to [RFC 2863] for details. Under the subnode *ifMIBO*
jects *(ifMIB 1)*, there are three tables *ifXTable (ifMIBObjects 1)*, *ifStackTable (ifMIBObjects 2)*, and *if*
cvAddressTable (ifMIBObjects 4). Including the *ifTable (interfaces 2)*, there are four generic Interfac
group tables under the two MIBs, *interfaces* and *ifMIB*, which we should be concerned with in definin
managed objects in the DLL layer. In addition to this, there are device-specific interface MIBs, suc
as Ethernet-like managed objects (*transmission 7*) that we would discuss under each subject as we de
with them. It is worth noting that specifications for *ifMIB* have gone through a series of documenta
tion—RFC 1229, RFC 1573, RFC 2233, and RFC 2863—each obsolescing the previous version.

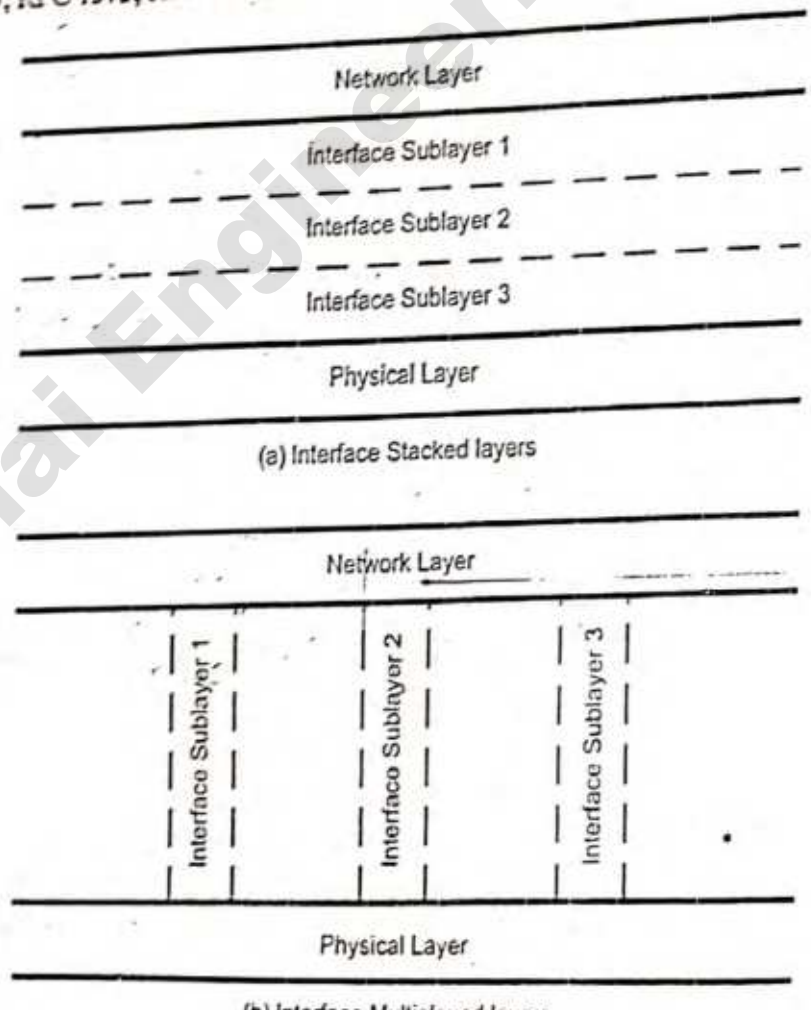


Figure 4.29 Interface Sublayers

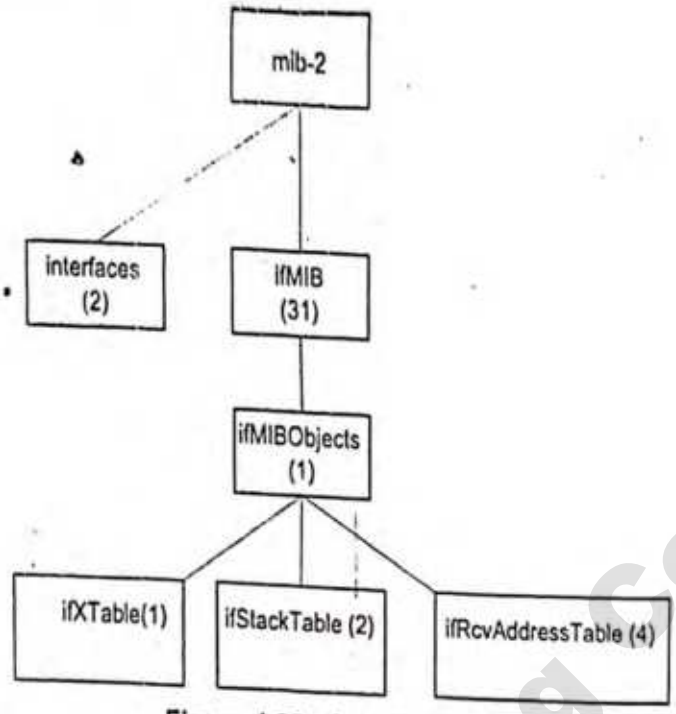


Figure 4.30 Interfaces Groups

ifXTable contains objects that have been added to the Interface MIB group as a result of the Interface evolution effort, or replacements for objects of the original (MIB-II) *ifTable* that were deprecated because the semantics of the said objects have significantly changed. It is an augmentation of *ifTable*. How two tables are augmented in SMI to appear as a single table is described in Chapter 6 under SNMP2.

ifStackTable contains objects that define relationships among sublayers of an interface. Each sublayer is *defined* as an *ifType* and is represented by a conceptual row in the *ifTable*. Because of the addition of such conceptual rows, the value of *ifIndex* is no longer constrained. In other words, it can be stated that the index of a conceptual row no longer has to be less than or equal to the value of *ifIndex*. The upper layer in the *ifStackTable*, *ifStackHigherLayer*, is the sublayer above the sublayer under consideration and carries the value of the *ifIndex* of that sublayer. If there is no interface sublayer above, i.e., it interfaces directly with the network layer, then the *ifIndex* value is zero. Similarly, *ifStackLowerLayer* is the lower interface sublayer, it has a corresponding *ifIndex* value of that row. If it interfaces directly with the physical medium, its value is zero.

ifRcvAddressTable contains objects that are used to define media-level addresses, which this *interface* will receive, such as a port ID. This table is a generic table.

Address Translation Group. The Address Translation group consists of a table that converts NetworkAddress to a physical or subnetwork address for all interfaces of the system. For example, in Ethernet the translation table is ARP cache. Since in MIB-II each protocol group contains its own translation table, this is not needed and hence its status is deprecated. It is mandatory to be implemented to be backward compatible with MIB-I.

IP Group. The Internet is based on IP protocol as the networking protocol. This group has information on various parameters of the protocol. It also has a table that replaces the Address Translation table. Routers in the network periodically execute the routing algorithm and update its routing table, which are defined as managed objects in this group. We will discuss the contents of this group in detail now.

The IP group defines all the parameters needed for the node to handle a network layer IP protocol either as a host or as a router; implementation is mandatory. Figure 4.31 and Table 4.7 present the tree structure and details of the entities, respectively. The group contains three tables, IP address table, IP routing table, and IP Address Translation table.

We can use the IP MIB to acquire any information associated with the IP layer. For example, to learn the value of the managed object, *ipForwarding* will indicate whether the node is acting as just a router or a gateway between two autonomous networks. We can measure IP datagrams received that are in error, such as those with wrong addresses (*ipInAddrErrors*).

The three tables belonging to the IP group are shown in Figure 4.32 (IP Address Table), Figure 4.33 (IP Routing Table), and Figure 4.34 (IP Address Translation Table). Table 4.8 shows the entity table for the IP address table. The index for the table, *ipAdEntAddr*, is shown in bold letters.

In Figure 4.23(b), we illustrated an example of four instantiations (rows) associated with the IP address table. The IP address table MIB shown in Figure 4.32 and Table 4.8 is used to retrieve data from the router. It could be retrieved using *get-request* or *get-next-request* commands.

The IP routing table is shown in Figure 4.33 and Table 4.9. It contains an entry for each route presently known to the entity. Multiple routes, up to five, to a single destination can appear in the table, but access to such multiple entries is dependent on the table-access mechanism defined by the network management protocol. Routes are indicated by the entities, *ipRouteMetricN*, where *N* is any integer from 1

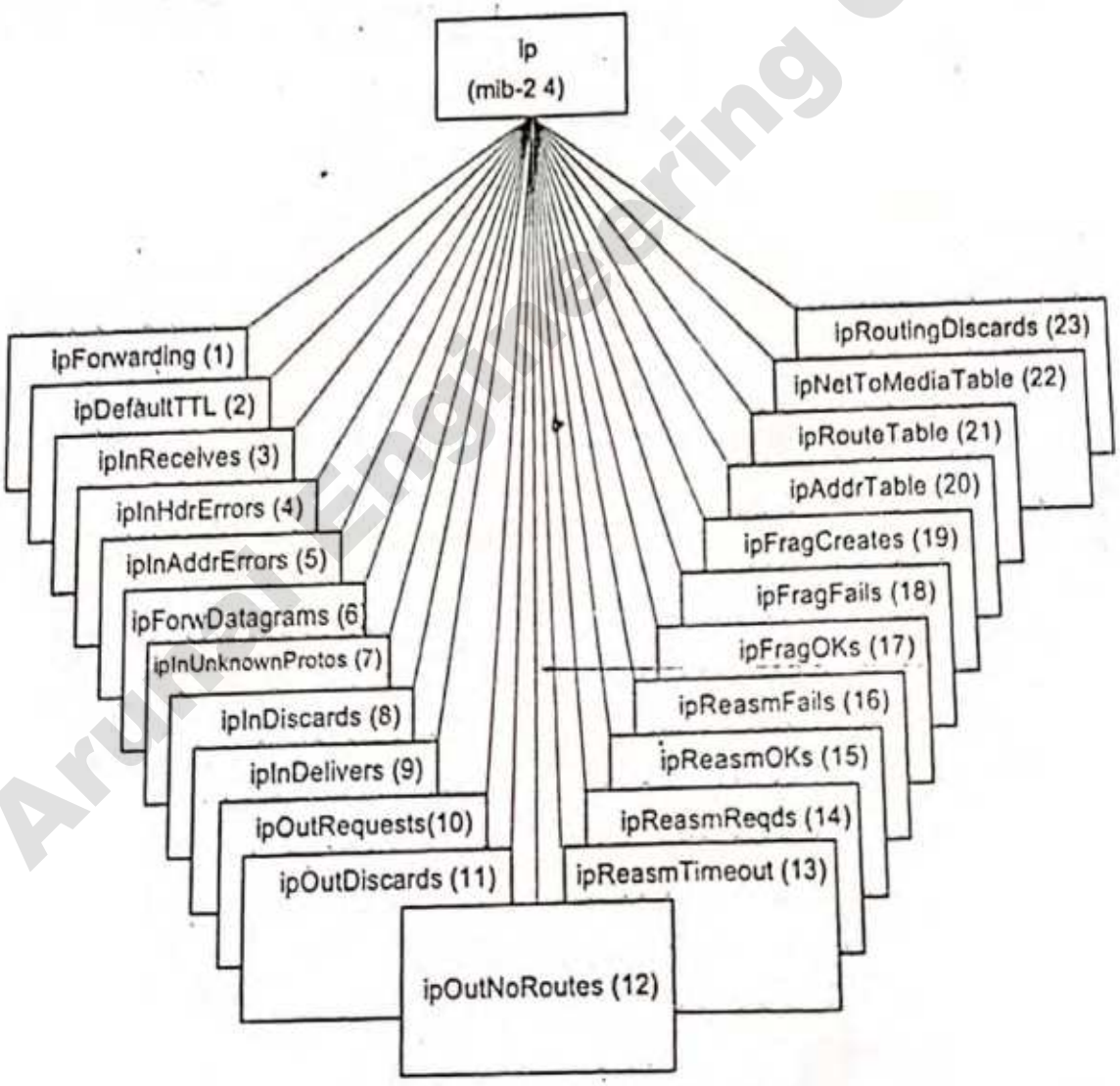
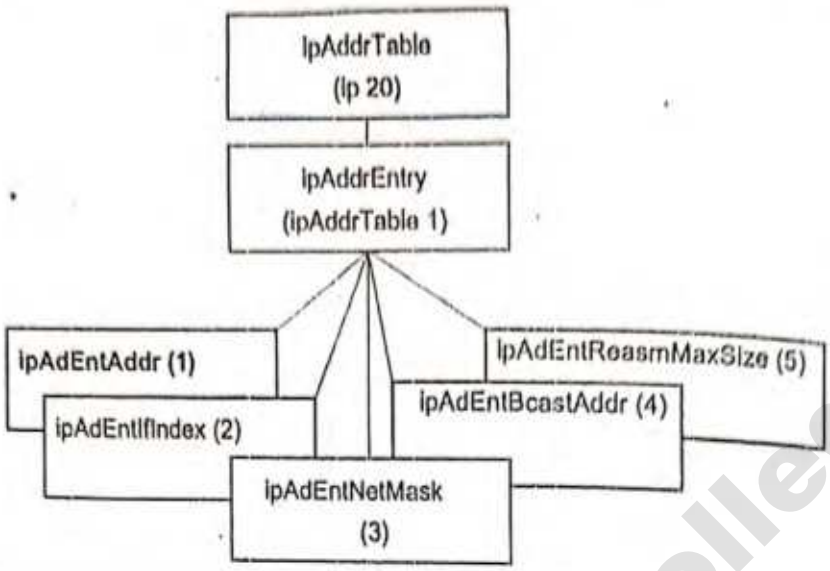


Figure 4.31 IP Group

Table 4.7 IP Group

ENTITY	OID	DESCRIPTION (BRIEF)
ipForwarding	Ip 1	Node acting as a gateway or not
ipDefaultTTL	Ip 2	Time-to-Live field of the IP header
ipInReceives	Ip 3	Total number of input datagrams received from interfaces, including those in error
ipInHdrErrors	Ip 4	Number of datagrams discarded due to header errors
IpInAddrErrors	Ip 5	Number of datagrams discarded due to address errors
ipForwDatagrams	ip 6	Number of input datagrams attempted to forward to the destination; successfully forwarded datagrams for source routing
ipInUnknownProtos	ip 7	Number of locally addressed datagrams received successfully but discarded due to unsupported protocol
ipInDiscards	ip 8	Number of input datagrams discarded with no problems (e.g. back of buffer space)
ipInDelivers	ip 9	Total number of input datagrams successfully delivered to IP user protocols
ipOutRequests	ip 10	Total number of IP datagrams that local IP user protocols supplied to IP
ipOutDiscards	ip 11	Number of no-error IP datagrams discarded with no problems (e.g. lack of buffer space)
ipOutNoRoutes	ip 12	Number of IP datagrams discarded because no route could be found to transmit them to their destination
ipReasmTimeOut	ip 13	Maximum number of seconds that received fragments are held while they are awaiting reassembly
ipReasmReqds	ip 14	Number of IP datagrams received needing reassembly
ipReasmOKs	ip 15	Number of successfully reassembled datagrams
ipReasmFails	ip 16	Number of failures detected by the IP reassembly algorithm (not discarded fragments)
ipFragOKs	ip 17	Number of successfully fragmented datagrams
ipFragFails	ip 18	Number of IP datagrams not fragmented due to "Don't Fragment Flag" set
ipFragCreates	ip 19	Number of datagram fragments generated as a result of fragmentation
ipAdddrTable	ip 20	Table of IP addresses
ipRouteTable	ip 21	IP routing table containing an entry
ipNetToMediaTable	ip 22	IP Address Translation table mapping IP addresses to physical addresses
ipRoutingDiscards	ip 23	Number of routing entries discarded even though they were valid



Legend: INDEX in bold

Figure 4.32 IP Address Table

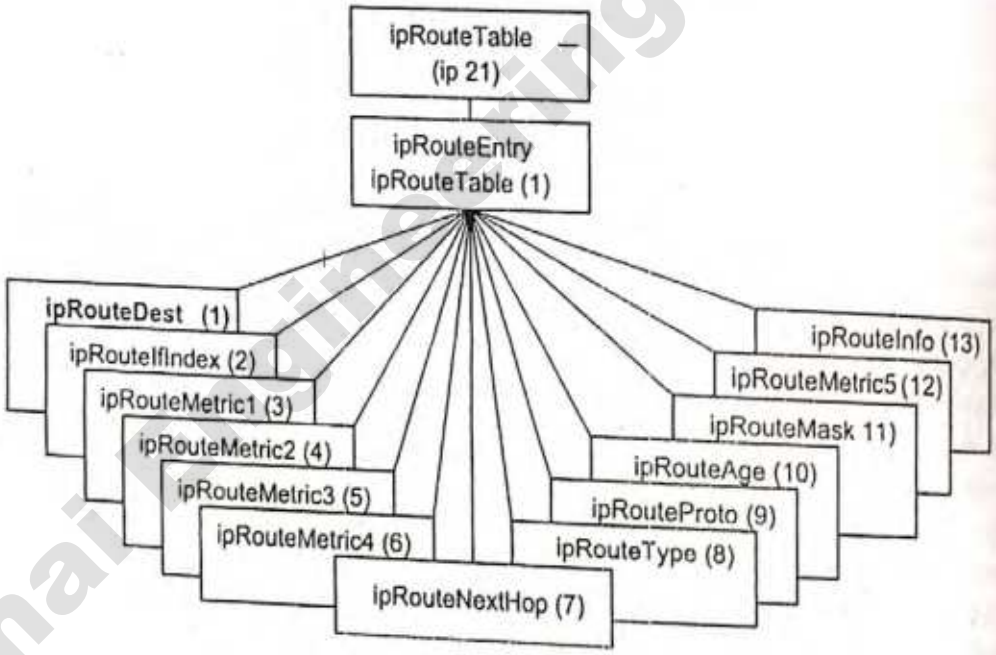


Figure 4.33 IP Routing Table

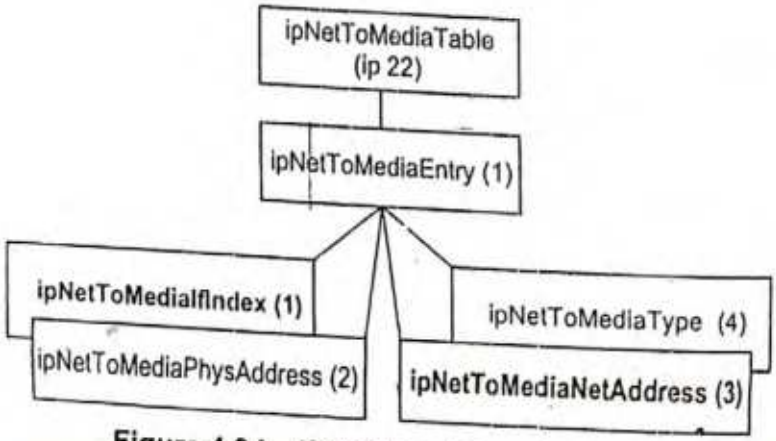


Figure 4.34 IP Address Translation Table

Table 4.8 IP Address Table

ENTITY	OID	DESCRIPTION (BRIEF)
ipAddrTable	ip 20	Table of IP addresses
ipAddrEntry *	IpAddrTable 1	One of the entries in the IP address table
ipAdEntAddr	IpAddrEntry 1	The IP address to which this entry's addressing information pertains
ipAdEntIfIndex	IpAddrEntry 2	Index value of the entry, same as ifIndex
ipAdEntNetMask	IpAddrEntry 3	Subnet mask for the IP address of the entry
ipAdEntBcastAddr	IpAddrEntry 4	Broadcast address indicator bit
ipAdEntReasmMaxSize	IpAddrEntry 5	Largest IP datagram that can be reassembled on this interface

Table 4.9 IP Routing Table

ENTITY	OID	DESCRIPTION (BRIEF)
ipRouteTable	ip 21	IP routing table
ipRouteEntry	ipRouteTable 1	Route to a particular destination
ipRouteDest	ipRouteEntry 1	Destination IP address of this route
ipRouteIfIndex	ipRouteEntry 2	Index of interface, same as ifIndex
ipRouteMetric1	ipRouteEntry 3	Primary routing metric for this route
ipRouteMetric2	ipRouteEntry 4	An alternative routing metric for this route
ipRouteMetric3	ipRouteEntry 5	An alternative routing metric for this route
ipRouteMetric4	ipRouteEntry 6	An alternative routing metric for this route
ipRouteNextHop	ipRouteEntry 7	IP address of the next hop
ipRouteType	ipRouteEntry 8	Type of route
ipRouteProto	ipRouteEntry 9	Routing mechanism by which this route was learned
ipRouteAge	ipRouteEntry 10	Number of seconds since routing was last updated
ipRouteMask	ipRouteEntry 11	Mask to be logically ANDed with the destination address before comparing with the ipRouteDest field
ipRouteMetric5	ipRouteEntry 12	An alternative metric for this route
ipRouteInfo	ipRouteEntry 13	Reference to MIB definition specific to the routing protocol

to 5. An entry 0.0.0.0 in *ipRouteDest* is considered a default route. The index for the table is *ipRoute Dest*. As in the IP address table, the *ipRouteIfIndex* has the same value as the *ifIndex* of the Interfaces table.

Figure 4.34 and Table 4.10 show the IP Address Translation table. It contains cross-references between IP addresses and physical addresses, such as MAC address of Ethernet interface cards. In some

Table 4.10 IP Address Translation Table

ENTITY	OID	DESCRIPTION.(BRIEF)
ipNetToMediaTable	Ip 22	Table mapping IP addresses to physical addresses
ipNetToMediaEntry	IpNetToMediaTable 1	IP address to physical address for the particular interface
IpNetToMediaIfIndex	IpNetToMediaEntry 1	Interfaces on which this entry's equivalence is effective; same as ifIndex
ipNetToMediaPhysAddress	IpNetToMediaEntry 2	Media-dependent physical address
IpNetToMediaNetAddress	IpNetToMediaEntry 3	IP address
ipNetToMediaType	IpNetToMediaEntry 4	Type of mapping; validates with IpNetToMediaType object

situations, such as DDN-X.25 where this relationship is algorithmic, this table is not needed and hence has zero entries. Indices for this table consist of two entities, *ipNetToMediaIfIndex* and *ipNetToMediaNetAddress*. Again, the *IpNetToMediaIfIndex* has the same value as *ifIndex* in the Interfaces group.

Baker [RFC 1354] has proposed an improved implementation of the IP routing table, called the IP Forwarding Table shown as an MIB tree in Figure 4.35 and the associated table in Table 4.11. The routing table that was originally proposed in RFC 1213 is inconsistent with SNMP protocol in that no specific policy was defined to choose the path among multiple choices in the IP route table. RFC 1354 has fixed this deficiency. Besides, it has added next hop autonomous system number, useful to the administrators of regional networks.

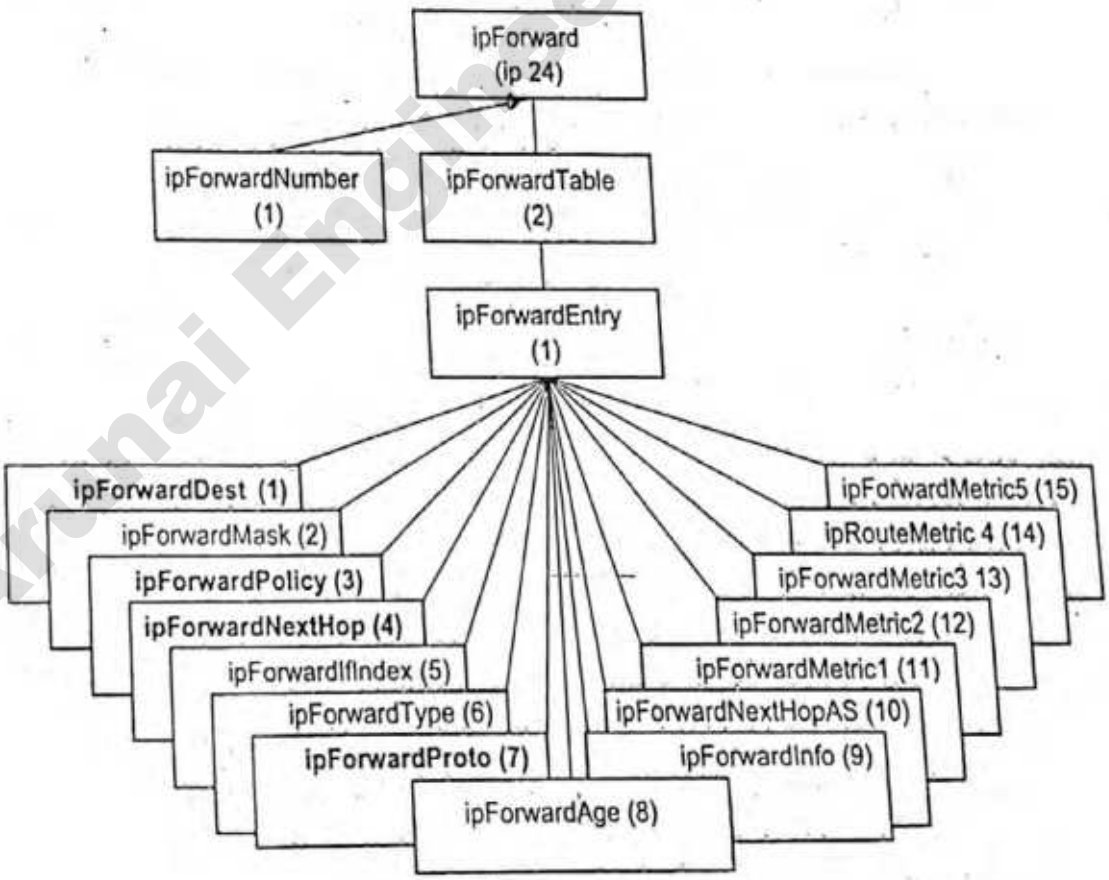


Figure 4.35 IP Forwarding Table

Table 4.11 IP Forwarding Table

ENTITY	OID	DESCRIPTION (BRIEF)
ipForward	ip 24	Contains information on IP forwarding table; deprecates IP routing table
ipForwardNumber	ipForward 1	Number of entries in the IP forward table
ipForwardTable	ipForward 2	Routing table of this entity
IpForwardEntry	IpForwardTable 1	A particular route to a particular destination under a particular policy
ipForwardDest	IpForwardEntry 1	Destination IP route of this address
ipForwardMask	IpForwardEntry 2	Mask to be logically ANDed with the destination address before comparing with the ipRouteDest field
ipForwardPolicy	IpForwardEntry 3	Set of conditions that selects one multipath route
ipForwardNextHop	IpForwardEntry 4	Address of the next system
ipForwardIfIndex	IpForwardEntry 5	ifIndex value of the interface
ipForwardType	IpForwardEntry 6	Type of route: remote, local, invalid, or otherwise; enumerated integer syntax
ipForwardProto	IpForwardEntry 7	Routing mechanism by which this route was learned
ipForwardAge	IpForwardEntry 8	Number of seconds since routing was last updated
ipForwardInfo	IpForwardEntry 9	Reference to MIB definition specific to the routing protocol
ipForwardNextHopAS	IpForwardEntry 10	Autonomous system number of Next Hop
ipForwardMetric1	IpForwardEntry 11	Primary routing metric for this route
ipForwardMetric2	IpForwardEntry 12	An alternative routing metric for this route
ipForwardMetric3	IpForwardEntry 13	An alternative routing metric for this route
ipForwardMetric4	IpForwardEntry 14	An alternative routing metric for this route
ipForwardMetric5	IpForwardEntry 15	An alternative routing metric for this route

The entity *ipForwardPolicy* defines the general set of conditions that would cause the selection of one multipath route over others. Selections of path can be done by the protocol. If it is not done by the protocol, it is then specified by the IP Type-of-Service (TOS) Field, which is a part of the IP type of service field. See Baker [RFC 1354] for more details.

ICMP Group. We used the ICMP to do some of the networking exercises in Chapter 1. It is part of the TCP/IP suite of protocols. All parameters associated with ICMP protocol are covered in this group.

As mentioned in Section 4.2, ICMP is a precursor of SNMP and a part of the TCP/IP suite. It is included in MIB-I and MIB-II and implementation is mandatory. The ICMP group contains statistics on ICMP control messages of ICMP and is presented in Figure 4.36 and Table 4.12. The syntax of all entities is read-only counter. For example, statistics on the number of *ping* requests (icmp echo request) sent might be obtained from the counter reading of *icmpOutEchoes*.

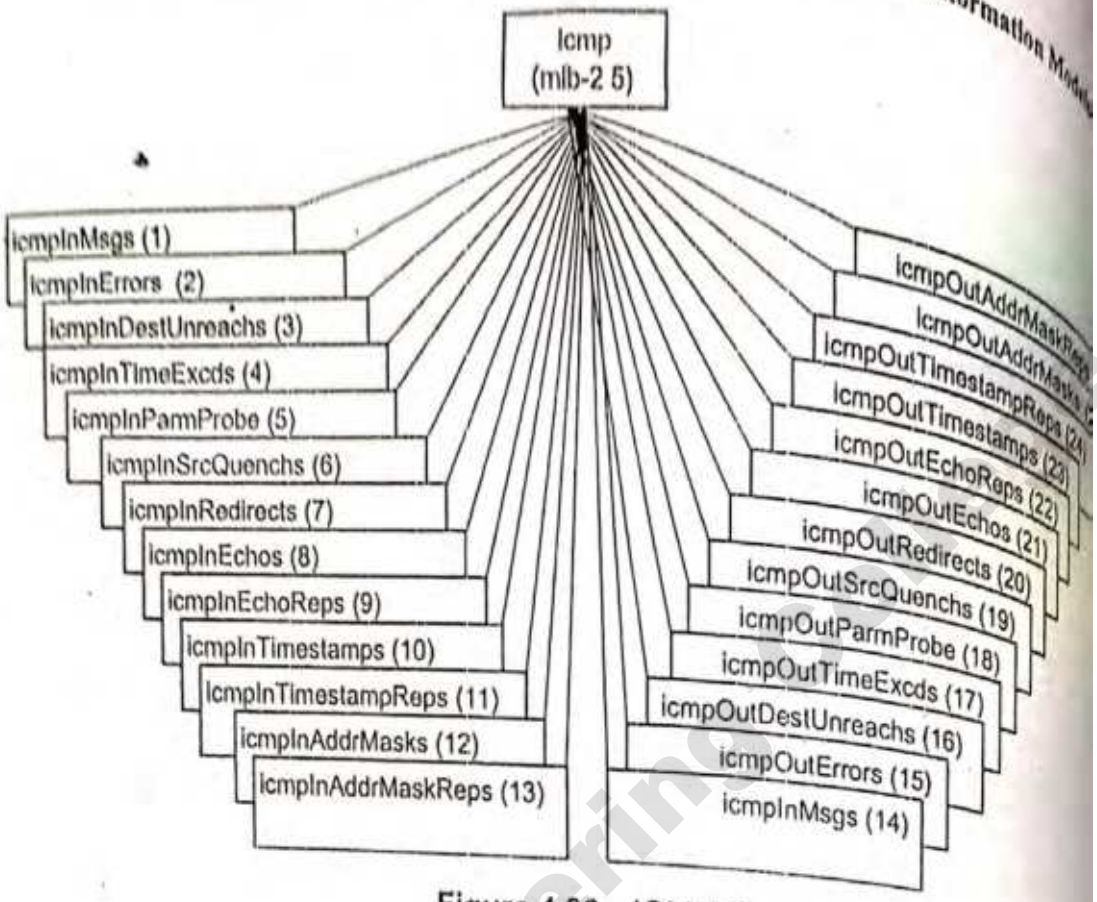


Figure 4.36 ICMP Group

Table 4.12 ICMP Group

ENTITY	OID	DESCRIPTION (BRIEF)
icmpInMsgs	icmp 1	Total number of ICMP messages received by the entity including icmpInErrors
icmpInErrors	icmp 2	Number of messages received by the entity with ICMP-specific errors
icmpInDestUnreachs	icmp 3	Number of ICMP Destination Unreachable messages received
icmpInTimeExcds	icmp 4	Number of ICMP Time Exceeded messages received
icmpInParmProbs	icmp 5	Number of ICMP Parameter Problem messages received
icmpInSrcQuenches	icmp 6	Number of ICMP Source Quench messages received
icmpInRedirects	icmp 7	Number of ICMP Redirect messages received
icmpInEchos	icmp 8	Number of ICMP Echo (request) messages received
icmpInEchoReps	icmp 9	Number of ICMP Echo Reply messages received
icmpInTimestamps	icmp 10	Number of ICMP Timestamp (request) messages received
icmpInTimestampReps	icmp 11	Number of ICMP Timestamp Reply messages received
icmpInAddrMasks	icmp 12	Number of ICMP Address Mask Request messages received

Table 4.12 (continued)

ENTITY	OID	DESCRIPTION (BRIEF)
icmpInAddrMaskReps	icmp 13	Number of ICMP Address Mask Reply messages received
icmpOutMsgs	icmp 14	Total number of ICMP messages attempted to be sent by this entity
icmpOutErrors	icmp 15	Number of good ICMP messages not sent, does not include the ones with errors
icmpOutDestUnreachs	icmp 16	Number of ICMP Destination Unreachable messages sent
icmpOutTimeExcds	icmp 17	Number of ICMP Time Exceeded messages sent
icmpOutParmProbs	icmp 18	Number of ICMP Parameter Problem messages sent
icmpOutSrcQuenchs	icmp 19	Number of ICMP Source Quench messages sent
icmpOutRedirects	icmp 20	Number of ICMP Redirect messages sent
icmpOutEchos	icmp 21	Number of ICMP Echo (request) messages sent
icmpOutEchoReps	icmp 22	Number of ICMP Echo Reply messages sent
icmpOutTimestamps	icmp 23	Number of ICMP Timestamp (request) messages sent
icmpOutTimestampReps	icmp 24	Number of ICMP Timestamp Reply messages sent
icmpOutAddrMasks	icmp 25	Number of ICMP Address Mask Request messages sent
icmpOutAddrMaskReps	icmp 26	Number of ICMP Address Mask Reply messages sent

TCP Group. The transport layer of the Internet defines Transmission Control Protocol (TCP) for a connection-oriented circuit and User Datagram Protocol (UDP) for a connectionless circuit. We will describe the TCP group in this section and UDP in the next subsection.

The TCP group contains entities that are associated with the connection-oriented TCP. They are present only as long as the particular connection persists. It is mandatory to implement this group. The entities are shown in Figure 4.37 and Table 4.13. It contains one table, the TCP connection table, which is presented in Figure 4.38 and Table 4.14. The table entry has four indices to uniquely define it in the table. They are: *tcpConnLocalAddress, tcpConnLocalPort, tcpConnRemAddress*, and *tcpConnRemPort* and are identified in boldface. One may obtain all TCP active sessions from this table with addresses and ports of local and remote entities.

UDP Group. The UDP group contains information associated with the connectionless transport protocol. Its implementation is mandatory. Figure 4.39 and Table 4.15 present the UDP group tree structure and entities, respectively. The group contains a UDP listener table, shown as part of Figure 4.39 and Table 4.15. The table contains information about the entity's UDP end-points on which a local application is currently accepting datagrams. Indices for the table entry are *udpLocalAddress* and *udpLocalPort*, and are indicated in bold letters.

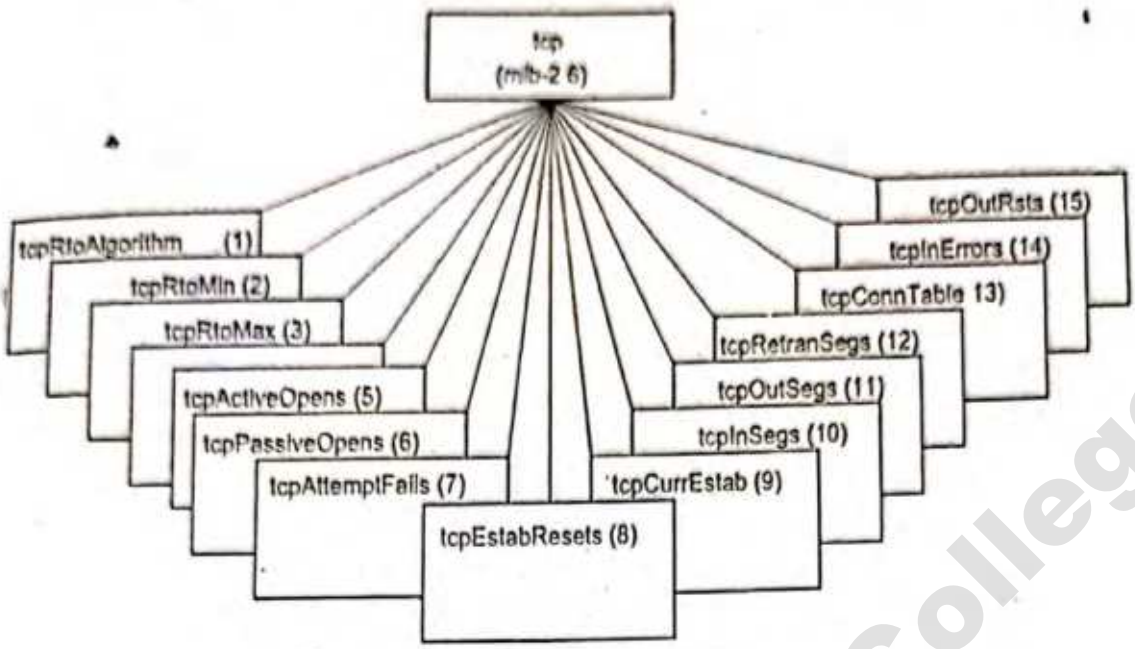


Figure 4.37 TCP Group

Table 4.13 TCP Group

ENTITY	OID	DESCRIPTION (BRIEF)
tcpRtoAlgorithm	tcp 1	Timeout algorithm for retransmission of octets
tcpRtoMin	tcp 2	Minimum value for timeout in milliseconds for retransmission
tcpRtoMax	tcp 3	Maximum value for timeout in milliseconds retransmission
tcpMaxConn	tcp 4	Maximum number of TCP connections
tcpActiveOpens	tcp 5	Number of active connections made CLOSED to SYN-SENT state
tcpPassiveOpens	tcp 6	Number of passive connections made LISTEN to SYN-RCVD state
tcpAttemptFails	tcp 7	Number of failed attempts to make connection
tcpEstabResets	tcp 8	Number of resets done to either CLOSED or LISTEN state
tcpCurrEstab	tcp 9	Number of connections for which the current state is either ESTABLISHED or CLOSED-WAIT
tcpInSegs	tcp 10	Total number of segments received including with errors
tcpOutSegs	tcp 11	Total number of segments sent excluding retransmission
tcpRetransSegs	tcp 12	Total number of segments retransmitted
tcpConnTable	tcp 13	TCO connection table
tcpInErrs	tcp 14	Total number of segments received in error
tcpOutRsts	tcp 15	Number of segment sent containing RST flag

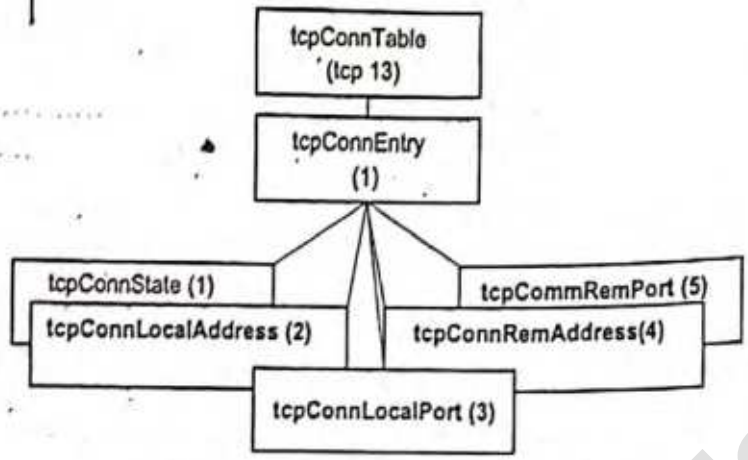


Figure 4.38 TCP Connection Table

Table 4.14 TCP Connection Table

ENTITY	OID	DESCRIPTION (BRIEF)
tcpConnTable	tcp 13	TCO connection table
tcpconnEntry	TcpConnTable 1	Information about a particular TCP connection
tcpConnState	TcpConnEntry 1	State of the TCP connection
tcpConnLocalAddress	TcpConnEntry 2	Local IP address
tcpConnLocalPort	TcpConnEntry 3	Local port number
tcpConnRemAddress	TcpConnEntry 4	Remote IP address
tcpConnRemPort	TcpConnEntry 5	Remote port number

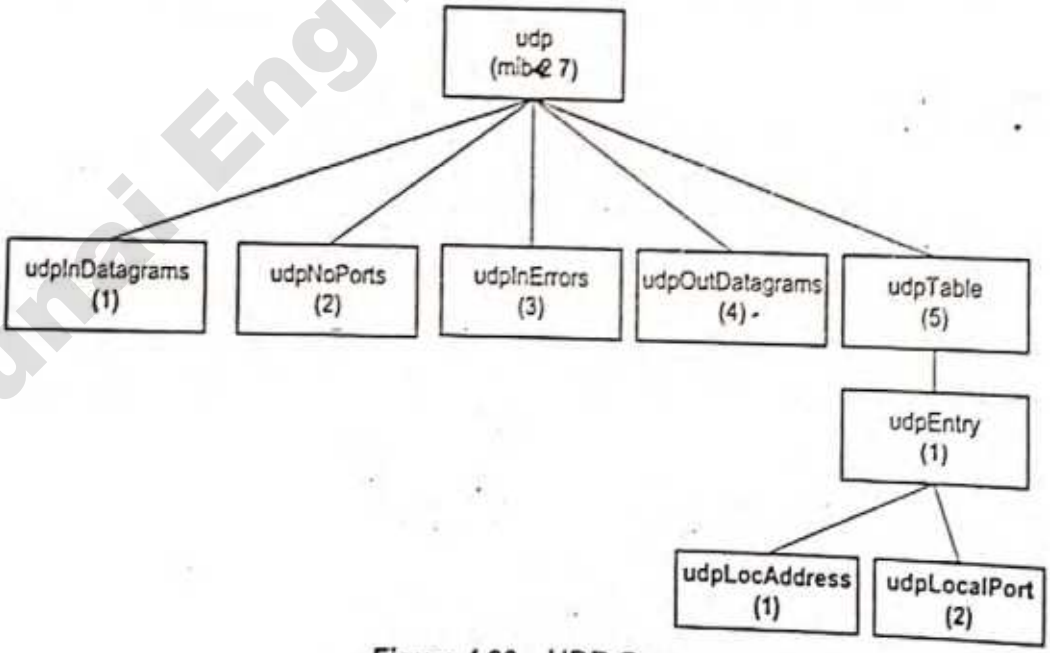


Figure 4.39 UDP Group

Table 4.15 UDP Group

ENTITY	OID	DESCRIPTION (BRIEF)
udpInDatagrams	udp 1	Total number of datagrams delivered to the users
udpNoPorts	udp 2	Total number of received datagrams for which there is no application
udpInErrors	udp 3	Number of received datagrams with errors
udpOutDatagrams	udp 4	Total number of datagrams sent
udpTable*	udp 5	UDP Listener table
udpEntry	udpTable 1	Information about a particular connection or UDP listener
udpLocalAddress	udpEntry 1	Local IP address
udpLocalPort	udpEntry 2	Local UDP port

Summary

We have learned the basic functions of SNMP management in this chapter. Advanced functio covered in the next chapter. The subject matter included in this chapter has been approved as a sta by IETF and implemented by most vendors.

We briefly learned the historical development of SNMP standards and documents. They grew out of practical necessity than the need for setting standards. The Internet Engineering Task Force standards organization and RFC, STD, and FYI are IETF documents on standards development.

SNMP management is organized as two-tier management, in which a manager process and an process communicate with each other. The agent process resides in the network element. The ma process is built in network management stations. The agent process does not perform any ana which is done in the manager. The two-tier structure can be extended to three-tier by sandwich proxy agent, or RMON, between the manager and the agent.

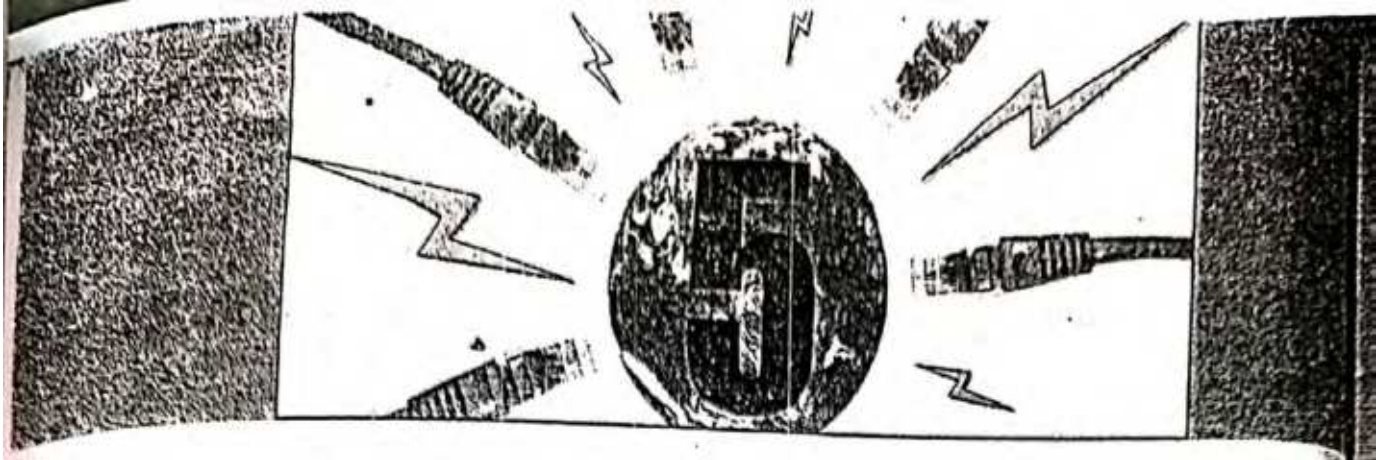
All management operations are done using five messages in SNMPv1, which is the current stan They are get-request, get-next, set-request, get-response, and trap. The first three are sent fro manager to the agent, and the last two are sent by the agent to the manager.

Messages are exchanged according to specifications defined in the Structure of Manage Information (SMI). It is composed of name, syntax, and encoding rules. The name is a unique for the managed object and an associated unique object identifier. The syntax uses Abstract S Notation 1 (ASN.1) language. Encoding is done using basic encoding rules (BER).

Objects or entities can be composed of other scalar objects. Multiple instances of a managed ob such as the IP address table, are handled by defining tables and columnar objects in the table. Man objects are organized in a virtual database, called the Management Information Base (MIB). It is a da from the management database that contains values for managed objects. Managed objects are gro in the MIB according to their function. MIB-II, which is a superset of MIB-I, consists of 11 groups. Si groups have since been added to the MIB, although they have not been approved as a standard.

Exercises

1. Refer to Figure 4.3 to answer the following questions:
   (a) What are the classes of networks shown in Figure 4.3(a)?
   (b) Explain the function of a network mask.



SNMPv1 Network Management: Communication and Functional Models

OBJECTIVES

- *Communication model: Administrative and messages*
- *Administrative structure*
  - *Community-based model*
  - *Access policy*
  - *MIB view*
- *Message PDU*
- *SNMP protocol specifications*
- *SNMP operations*
- *SNMP MIB*
- *SNMP functional model*

We have covered the organization and information models of SNMPv1 in the previous chapter. In this chapter we will address the SNMPv1 communication and functional models. Although SNMPv1 does not formally define the functional model, applications are built in the community-based access policy of the SNMP administrative model.

5.1 SNMP COMMUNICATION MODEL

The SNMPv1 communication model defines specifications of four aspects of SNMP communication: architecture, administrative model that defines data access policy, SNMP protocol, and SNMP MIB. Security in SNMP is managed by defining community, and only members belonging to the same community can communicate with each other. A manager can belong to multiple communities and can thus manage multiple domains. SNMP protocol specifications and messages are presented. SNMP entities are grouped into an SNMP MIB module.

5.1.1 SNMP Architecture

The SNMP architectural model consists of a collection of network management stations and network elements or objects. Network elements have management agents built in them. if they are managed

elements. The SNMP communications protocol is used to communicate information between network management stations and management agents in the elements.

There are three goals of the architecture in the original specifications of SNMP [RFC 1157]. First, it should minimize the number and complexity of management functions realized by the management agent. Secondly, it should be flexible for future expansion (addition of new aspects of operation and management). Lastly, the architecture should be independent of architecture and mechanisms of particular hosts and gateways.

Only non-aggregate objects are communicated using SNMP. The aggregate objects are communicated as instances of the object. This has been enhanced in SNMPv2, as we shall see in the next chapter. Consistent with the rest of SNMP standards, ASN.1 transfer syntax and BER encoding scheme are used for data transfer SNMP.

SNMP monitors the network with the five messages shown in Figure 4.9; and we discussed them in Section 4.6. They comprise three basic messages: set, get, and trap. Information about the network is primarily obtained by the management stations polling the agents. The get-request and get-next-request messages are generated by the manager to retrieve data from network elements using associated management agents. The set-request is used to initialize and edit network element parameters. The get-response-request is the response from the agent to get and set messages from the manager. The number of unsolicited messages in the form of traps is limited to make the architecture simple and to minimize traffic.

There are three types of traps—generic-trap, specific-trap, and time-stamp, which are application specific. The generic-trap type consists of *coldStart*, *warmStart*, *linkDown*, *linkUp*, *authenticationFailure*, *egpNeighborLoss*, and *enterpriseSpecific*. The specific-trap is a specific code and is generated even when an *enterpriseSpecific* trap is not present. An example of this would be to gather statistics whenever a particular event occurs, such as use by a particular group. The time-stamp trap is the time elapsed between the last initialization or re-initialization of the element and the generation of the trap.

SNMP messages are exchanged using a connectionless UDP transport protocol in order to be consistent with simplicity of the model, as well as to reduce traffic. However, the mechanisms of SNMP are suitable for a variety of protocols.

5.1.2 Administrative Model

Although the topic of administrative models should normally be discussed as part of security and privacy under the functional model, at this point it helps to understand the administrative relationship among entities that participate in the communication protocol in SNMP. Hence, we will discuss it now.

In RFC 1157 the entities residing in management stations and network elements are called SNMP application entities. Peer processes, which implement SNMP, and thus support SNMP application entities, are termed protocol entities. We will soon discuss protocol entities in detail. First, let us look at the application entities.

We will refer to the *application entity* residing in the management station as the SNMP manager, and the application entity in the element as the SNMP agent. The pairing of the two entities is called an SNMP community. The SNMP community name, called the *community*, is specified by a string of octets. Multiple pairs can belong to the same community. Figure 5.1 shows multiple SNMP managers communicating with a single SNMP agent. While an SNMP manager is monitoring traffic on an element, another manager may be configuring some administrative information on it. A third manager can be monitoring it to perform some statistical study. We also have the analogous situation where a manager communicates with multiple agents.

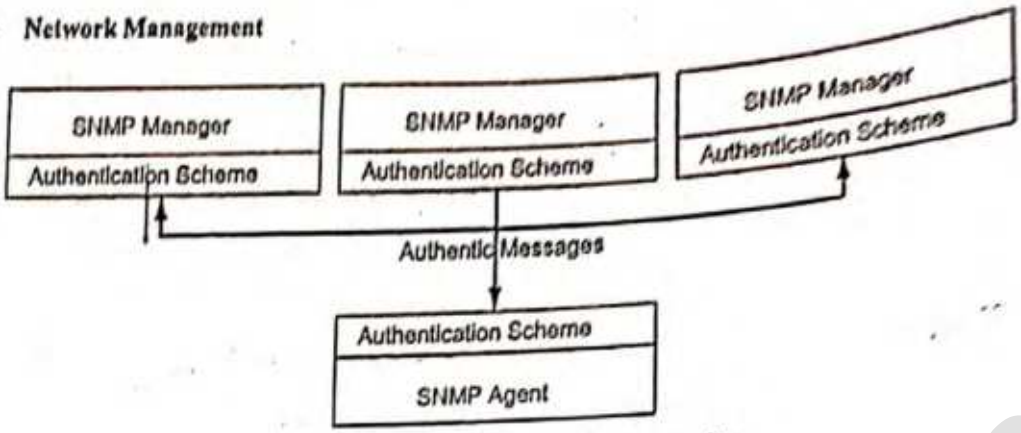


Figure 5.1 SNMP Community

With one-to-many, many-to-one, and many-to-many communication links between managers and agents, a basic authentication scheme and an access policy have been specified in SNMP. Figure 5.1 shows the *authentication scheme*, which is a filter module in the manager and the agent. The simplest form of authentication is the common community name between the two application entities. Encryption would be a higher level of authentication in which case both the source and the receiver know the common encryption and decryption algorithms.

The SNMP authorization is implemented as part of managed object MIB specifications. We discussed MIB specifications for managed objects in Chapter 4, and will discuss MIB specifications for SNMP protocol in Section 5.1.4. A network element comprises many managed objects—both standard and private. However, a management agent may be permitted to view only a subset of the network element's managed objects. This is called the community *MIB view*. In Figure 5.2 the SNMP agent has a MIB view of objects 2, 3, and 4, although there may be other objects associated with a network element. In addition to the MIB view, each community name is also assigned an *SNMP access mode*, either READ-ONLY or READ-WRITE, as shown in Figure 5.2. A pairing of SNMP MIB views with an SNMP access code is called a *community profile*.

A community profile in combination with the access mode of a managed object determines the operation that can be performed on the object by an agent. For example, in Figure 5.2, an SNMP agent with READ-WRITE SNMP access mode can perform all operations—get, set, and trap—on objects 2, 3, and 4. On the other hand, if the SNMP agent has READ-ONLY access mode privilege, it can only perform get and trap operations on objects 2, 3, and 4. Object 1 has a "not-accessible" access mode and hence no operation can be performed on it.

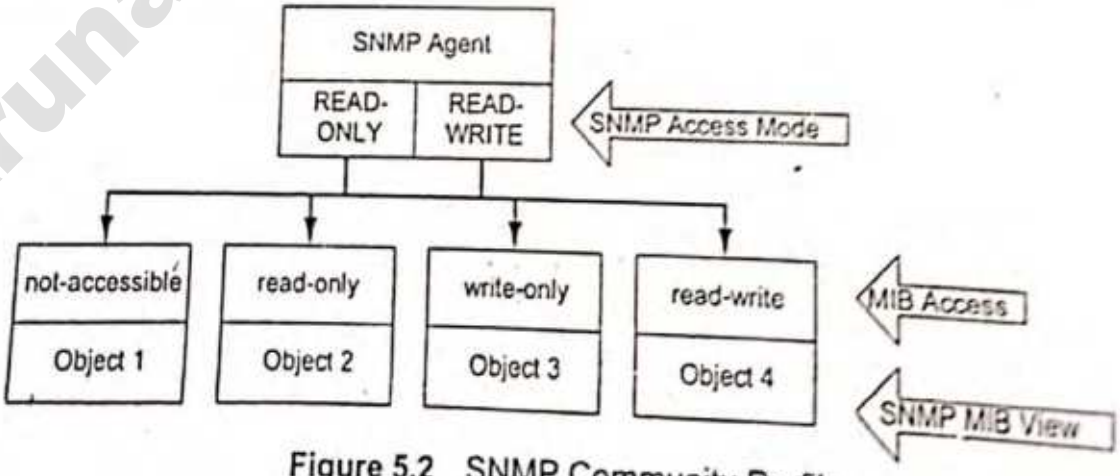


Figure 5.2 SNMP Community Profile

There are four access modes shown in Figure 5.2. They are not-accessible, read-only, write-only, and read-write. The tables are examples of no-access mode. One can only access scalar objects associated with the entities under the table. Most objects available for the public community are read-only, such as the interface statistics and the IP table in a router. These are the get and trap operations. If the access mode is defined as read-write, that operand is available for all three operations of get, set, and trap. An example of read-write access is *sysContact* in the system group. The write-only access mode is used to set the operand value of a get MIB object by the network manager, for example *sysDescr* in the system group. This is done in network management systems as implementation-specific.

We can now define an SNMP access policy in SNMP management. A pairing of an SNMP community with an SNMP community profile is defined as an SNMP access policy. This defines the administrative model of SNMP management. Figure 5.3 shows an example of three network management systems in three network operation centers (NOC) having different access to different community domains. Agents 1 and 2 belong to Community 1. However, they do have two different community profiles, community profiles 1 and 2. Manager 1, which is part of Community 1, can communicate with both Agents 1 and 2. However, it cannot communicate with Agents 3 and 4 belonging to Community 2. Manager 2 has access to them as it also belongs to Community 2. Agent 3 has community profile 3 and Agent 4 has community profile 4. Manager 3 has access to both Community 1 and 2 and hence can communicate with all the agents. We can picture an enterprise network management fitting this scenario. If a corporation has two operations in two cities, Manager 1 in NOC 1 and Manager 2 in NOC 2 are responsible for managing their respective domains. A top view of the overall operations can be viewed and managed by NOC 3 in the headquarters operation.

Figure 5.3 SNMP Access Policy

A practical application of the SNMP access policy can be envisioned in an enterprise management system of a corporation with headquarters in New York and domains or network sites in New York and San Francisco. Let Manager 1 and Community 1 be associated with San Francisco, and Manager 2 and Community 2 with New York. Let Manager 3 be the overall network management system, the Manager of Managers (MoM). Manager 1 manages Agents 1 and 2 associated with network elements in San Francisco. Manager 2 manages the New York network domain. Manager 1 does not have the view of New York and Manager 2 cannot perform operations on network elements belonging to the San Francisco domain. Manager 3 has both community names defined in its profile and hence has the view of the total enterprise network in New York and San Francisco.

The SNMP access policy has far-reaching consequences beyond that of servicing a TCP/IP-based Internet SNMP community. It can be extended to managing non-SNMP community using the SNMP proxy access policy. The SNMP agent associated with the proxy policy is called a proxy agent or commercially, a proxy server. The proxy agent monitors a non-SNMP community with non-SNMP agents and then converts objects and data to SNMP-compatible objects and data to feed to an SNMP manager.

Figure 5.4 shows an illustration of SNMP and non-SNMP communities being managed by an SNMP manager. A practical example of this would be a network of LAN and WAN. LAN could be a TCP/IP network with SNMP agents. WAN could be an X.25 network, which is not an Internet model, but can be managed by a proxy agent and integrated into the overall management system.

5.1.3 SNMP Protocol Specifications

Peer processes, which implement SNMP, and thus support SNMP application entities, are termed *protocol entities*. Communication among protocol entities is accomplished using messages encapsulated in a UDP datagram. An SNMP message consists of a version identifier, an SNMP community name, and a protocol data unit (PDU). Figure 5.5 shows the encapsulated SNMP message. The version and community names are added to the data PDU and along with the application header is passed on to the transport layer as SNMP PDU. The UDP header is added at the transport layer, which then forms the transport PDU for the network layer. The addition of the IP header to the Transport PDU forms the Network PDU for the data link layer (DLC). The network or DLC header is added before the frame is transmitted on to the physical medium.

An SNMP protocol entity is received on port 161 on the host except for trap, which is received on port 162. The maximum length of the protocol in SNMPv1 is 484 bytes (1,472 bytes now in practice). It

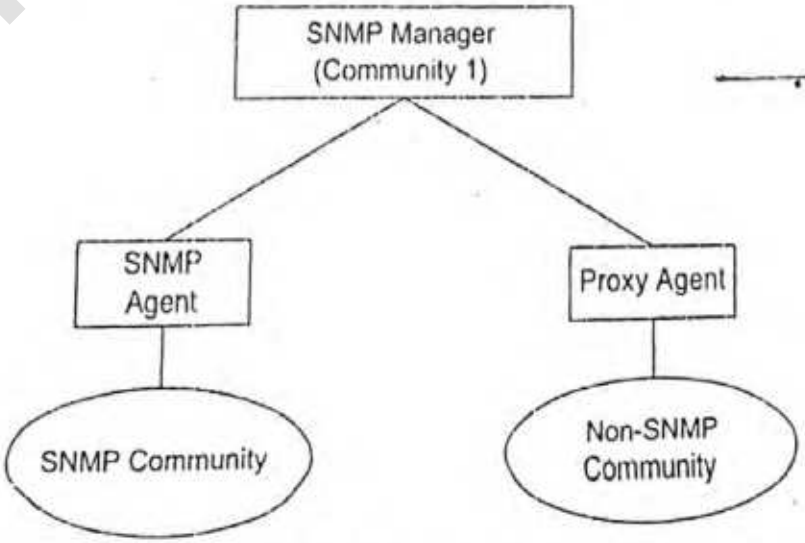


Figure 5.4 SNMP Proxy Access Policy

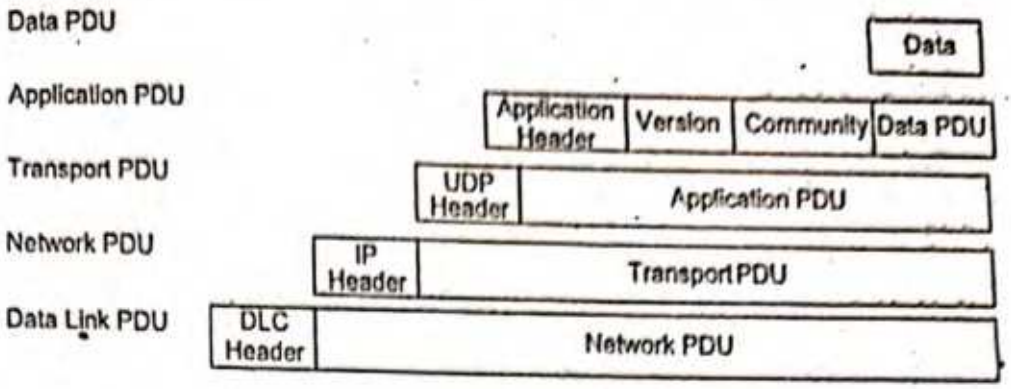


Figure 5.5 Encapsulated SNMP Message

is mandatory that all five PDUs be supported in all implementations: GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU, SetRequest-PDU, and Trap-PDU. One of these five data PDUs is the data PDU that we start with at the top in Figure 5.5. RFC 1157-SNMP Macro definition is given in Figure 5.6.

```
RFC1157 SNMP DEFINITIONS ::= BEGIN

IMPORTS

        ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
            FROM RFC1155 -SMI

--top-level message

        Message ::=
                SEQUENCE {
                        version         -- version
                         INTEGER {          -1 for this RFC
                                version-1(0)
                        },
                        community    -- community name
                         OCTET STRING,
                        data            -- e.g., PDUs if trivial
                         ANY            -- authentication is being used
                        }

-- protocol data units

        PDUs ::=
                CHOICE {
                        get-request
                        get-next-request    GetRequest-PDU,
                        get-response        GetNextRequest-PDU,
                        set-request         GetResponse-PDU,
                        trap                SetRequest-PDU,
                        }                   Trap-PDU

-- the individual PDUs and commonly used data types will be defined later

END
```

Figure 5.6 RFC 1157-SNMP Macro

```
-- request/response information

RequestId ::=
        INTEGER

ErrorStatus ::=
        INTEGER {
                noError(0)
                tooBig(1)
                noSuchName(2)
                bad value(3)
                readOnly(4)
                genErr(5)
        }

ErrorIndex ::=
        INTEGER

-- variable bindings

VarBind ::=
        SEQUENCE {
                name        ObjectName
                value       ObjectSyntax
        }

VarBindList ::=
        SEQUENCE OF
                VarBind
```

Figure 5.7 Get and Set Type PDU ASN.1 Construct [RFC 1157]

Basic operations of the protocol entity involve the following steps as a guide to implementation [RFC 1157]. The protocol entity that generates the message constructs the appropriate data PDU as an ASN.1 object. It then passes the ASN.1 object along with a community name and the transport addresses of itself and the destination (e.g., 123.234.245.156:161) to the authentication scheme. The authentication scheme returns another ASN.1 object (possibly encrypted). The protocol entity now constructs the message to be transmitted with the version number, community name, and the new ASN.1 object, then serializes it using the BER rules, and transmits it.

The reverse process goes on at the receiver. The message is discarded if an error is encountered in any of the steps. A trap may be generated in case of authentication failure. On successful receipt of the message, a return message is generated, if the original message is a get-request.

A managed object is a scalar variable and is simply called a variable. Associated with the variable is its value. The pairing of the variable and value is called *variable binding* or *VarBind*. The data PDU in the message contains a VarBind pair. For efficiency sake, a list of VarBind pairs can be sent in a message. The ASN.1 construct for get and set type of messages is shown in Figure 5.7 and a conceptual presentation in Figure 5.8. The *VarBindList* contains *n* instances of VarBind (pairs).

The PDU type for the five messages are application data types, which are defined in RFC 1157 as:

get-request [0]
get-next-request [1]

PDU Type	RequestID	Error Status	Error Index	VarBind 1 Name	VarBind 1 Value	...	VarBind n Name	VarBind n Value

Figure 5.8 Get and Set Type PDUs

set-request	[2]
get-response	[3]
trap	[4]

In Figure 5.8 *RequestID* is used to track a message with the expected response or for loss of a message (remember UDP is unreliable). Loss-of-message detection is implementation specific, such as time out if no response is received for a request within a given time. A non-zero *ErrorStatus* is used to indicate that an error occurred. The convention is not to use 0 if no error is detected. *ErrorIndex* is used to provide additional information on the error status. The value is filled with NULL in those cases where it is not applicable, such as in get-request data PDU. Otherwise, it is filled with the *varBind* number where the error occurred; for example, 1 if the error occurred in the first *varBind*, 5 if the fifth *varBind* had an error and so on.

Figure 5.9 shows the structure for a trap PDU, which contains *n VarBinds*, i.e., *n* managed objects. The enterprise [RFC 1155] and agent-address pertain to the system generating the trap. The generic-trap consists of seven types as listed in Table 5.1. The integer in parenthesis associated with each name indicates the enumerated INTEGER. The specific-trap is a trap that is not covered by the *enterpriseSpecific* trap. Time-stamp indicates the elapsed time since last re-initialization.

PDU Type	Enterprise	Agent Address	Generic-Trap Type	Specific-Trap Type	Time-Stamp	VarBind 1 Name	VarBind 1 Value	...	VarBind n Name	VarBind n Value

Figure 5.9 Trap PDU

Table 5.1 Generic Traps

GENERIC-TRAP TYPE	DESCRIPTION (BRIEF)
coldStart(0)	Sending protocol entity is reinitializing itself; agent configuration or protocol entity implementation may be altered
warmStart(1)	Sending protocol entity is reinitializing itself; agent configuration or protocol entity implementation not altered
linkDown(2)	Failure of one of the communication links
linkUp(3)	One of the links has come up
authenticationFailure(4)	Authentication failure
egpNeighborLoss(5)	Loss of EGP neighbor
enterpriseSpecific(6)	Enterprise-specific trap

5.7.4 SNMP Operations

SNMP operations comprise get and set messages from the manager to the agent, and get and trap messages from the agent to the manager. We will now look at these operations in detail in this section.

GetRequest PDU Operation. Figure 5.10 shows a sequence of operations in retrieving the values of objects in a System group. It starts with the get-request operation using a GetRequest PDU from a manager process to an agent process and the get-response from the agent with a GetResponse PDU. The message from the manager starts from the left side and ends at the agent process on the right side of the figure. The message from the agent process starts on the right side of the figure and ends at the manager process on the left side of the figure. The sequence of directed messages moves with time as we move down the figure. Messages depicted represent the values of the seven objects in the System group.

The manager process starts the sequence in Figure 5.10 with a GetRequest PDU for the object *sysDescr*. The agent process returns a GetResponse PDU with a value "SunOS." The manager then sends a request for *sysObjectID* and receives the value "E:hp." The exchange of messages goes on until the value of 72 for the last object in the group *sysServices* is received.

GetNextRequest PDU Operation. A get-next-request operation is very similar to get-request, except that the requested record is the next one to the OBJECT IDENTIFIER specified in the request. Figure 5.11 shows the operations associated with retrieving data for the System group by the manager process using the get-next-request. The first message is a GetRequest PDU for *sysDescr* with the response returning the value "SunOS." The manager process then issues a GetNextRequest PDU with the OBJECT IDENTIFIER *sysDescr*. The agent processes the name of the next OBJECT IDENTIFIER *sysObjectID* and its value "E:hp." The sequence terminates when the manager issues a get-next-request for the object identifier next to *sysServices*, and the agent process returns the error message "noSuchName."

The System group example we just looked at is a simple case where all the objects are single-valued scalar objects. Let us now consider a more complex scenario of a MIB that contains both scalar and aggregate objects. A generalized case of a conceptual MIB comprising three scalar objects and a table is

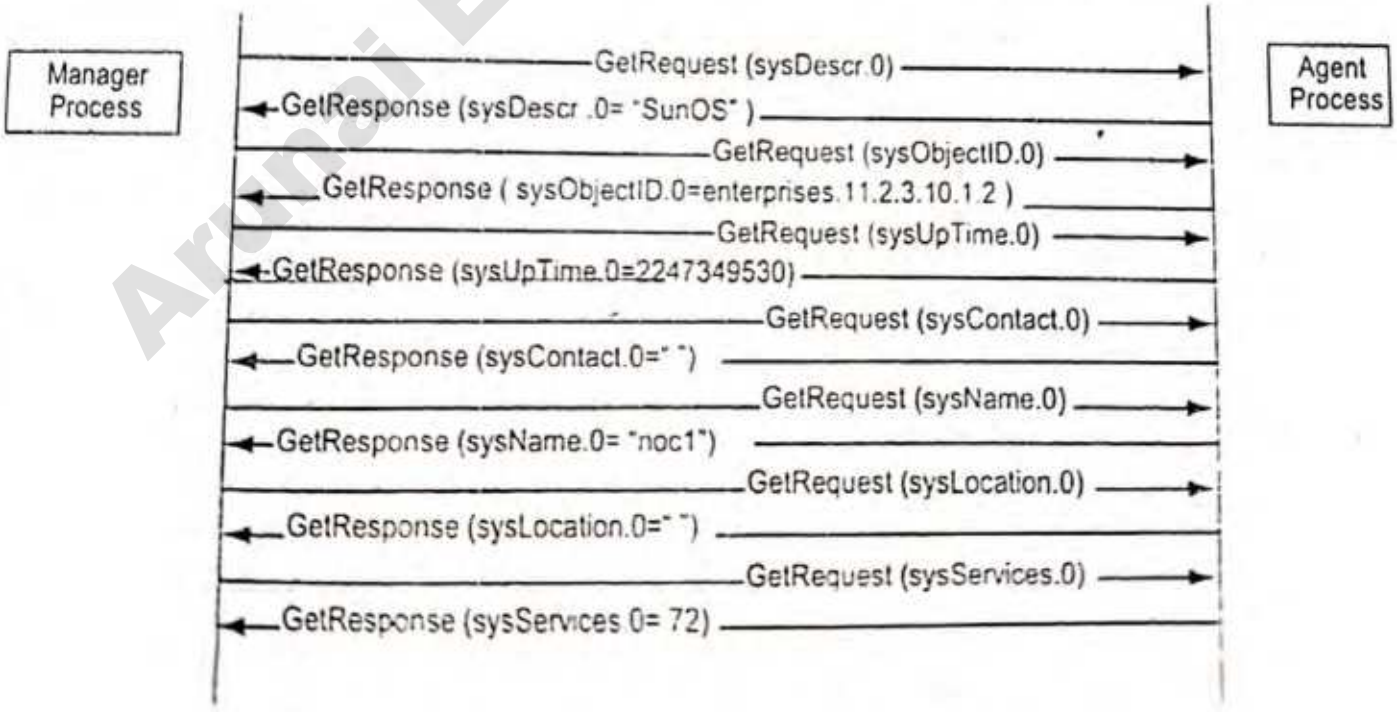


Figure 5.10 Get-Request Operation for System Group

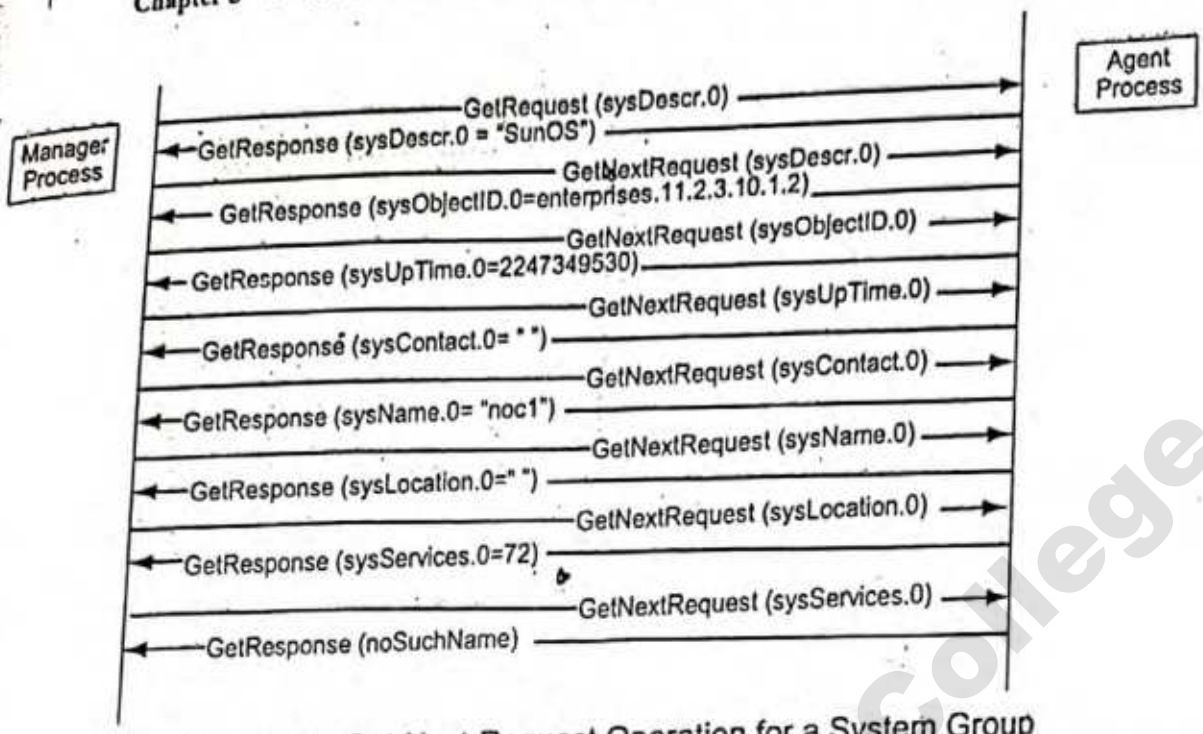


Figure 5.11 Get-Next-Request Operation for a System Group

shown in Figure 5.12. The first two objects A and B are single-valued scalar objects. They are followed by an aggregate object represented by the table T with an entry E and two rows of three columnar objects, T.E.1.1. through T.E.3.2. The MIB group ends with a scalar object Z.

Figure 5.13 shows the use of nine get-request messages to retrieve the nine objects. The left side of the figure shows the sequential operation for getting the MIB shown on the right side of the figure. The MIB shown is the same as in Figure 5.12, now drawn to follow the sequence of operations. We observe

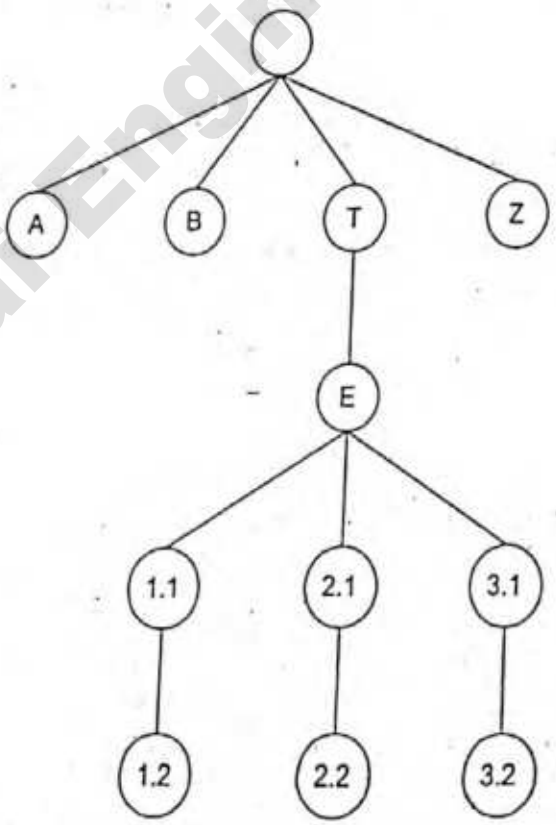


Figure 5.12 MIB for Operation Examples in Figures 5.13 and 5.15

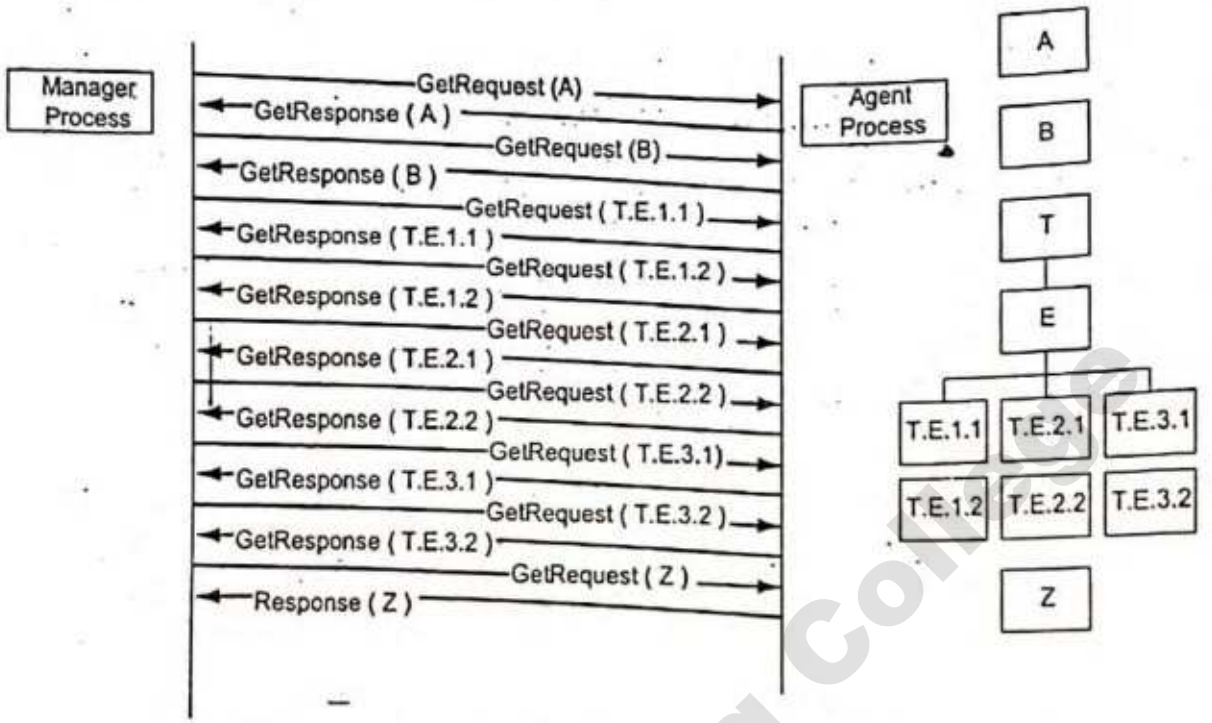


Figure 5.13 Get-Request Operation for a MIB in Figure 5.12

a few hidden assumptions in retrieving the data using the get-request operations. First, we need to know all the elements in the MIB including the number of columns and rows in the table. Second, we traversed the MIB from top to bottom, which is really from right to left in the MIB tree structure. Third, we retrieved the data in the table by traversing all the instances of a columnar object. The number of instances or rows in a table could be dynamic and is not always known to the management process. Thus, if the manager had issued a request for the object T.E.1.3 after acquiring T.E.1.2, it would have received an error message from the agent process. This is when get-next-request is very useful. However, we need to have a convention on the definition of what the next object in a MIB tree is, especially on the table representing an aggregate object. In SNMP, objects are retrieved using lexicographic convention. We will first explain what this convention is before using the get-next-request operation to retrieve the same MIB group data.

The increasing order of entity used in SNMP operations is in lexicographic order. Let us understand lexicographic order by considering a simple set of integers shown in Table 5.2. The left side is a sequence of numerically increasing integer numbers, and the right side is lexicographically increasing order for this sequence. We notice that in the lexicographic order, we start with the lowest integer in the leftmost character, which in our case is 1. Before increasing the order in the first position, we select the lowest integer in the second position from the left, which is 11. There are two numbers (1118 and 115) that start with 11. We anchor at 11 for the first two positions and then move on to select the lowest digit in the third position, which is 111. We then move to the fourth position and obtain 1118 as the second number. Now, return to the third position and retrieve 115 as the third number. Having exhausted 1s (ones) in positions two to four, select 2 for the second position, and retrieve 126 as the next number. We continue this process until we reach 9.

We will now apply the lexicographic sequence to ordering object identifiers in a MIB. Instead of each character being treated as a literal, we treat each node position as a literal and follow the same rules. An example illustrating this is given in Table 5.3. The MIB associated with this example is shown in

Table 5.2 Lexicographic-Order Number Example

NUMERICAL ORDER	LEXICOGRAPHIC ORDER
1	1
2	1118
3	115
9	126
15	15
22	2
34	22
115	250
126	2509
250	3
321	321
1118	34
2509	9

Figure 5.14. It can be noticed that the lexicographically increasing order of node traces the traversal of the tree starting from the leftmost node 1. We traverse down the path all the way to the leftmost leaf 1.1.5, keeping to the right whenever a fork is encountered. We then move up the tree and take a right on the first fork. This leads us to the leaf node 1.1.18. Thus, the rule at a forked node is to always keep to the right while traversing down and while going up. Thus, we are always keeping to the right if you imagine ourselves walking along the tree path and looking in the forward direction. We turn around when we reach a leaf.

Returning to get-next-request operation, the get-response message contains the value of the next lexicographic object value in each VarBind. If the request VarBind contains a scalar, non-tabular object,

Table 5.3 MIB Example for Lexicographic Ordering

1
1.1
1.1.5
1.1.18
1.2
1.2.6
2
2.2
2.10
2.10.9
3
3.4
3.21
9

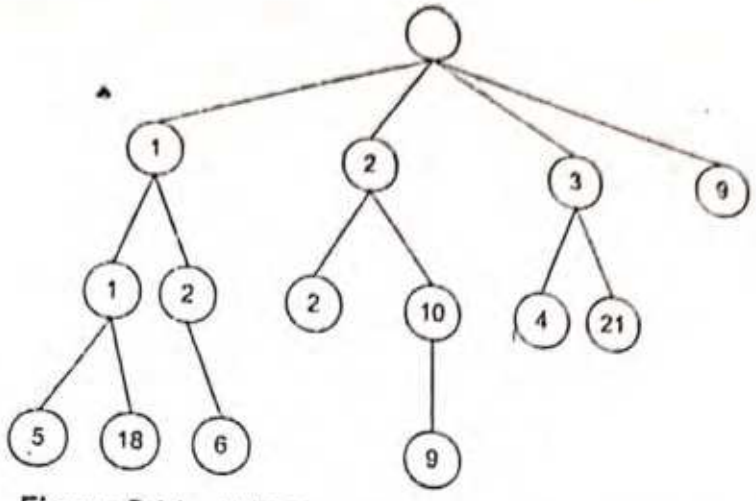


Figure 5.14 MIB Example for Lexicographic Ordering

the response contains the next scalar, non-tabular value, or the first columnar object value of a table, if it is the next lexicographic entity. Figure 5.15 shows the principle of operation of the functioning of get-next-request and response. We use the same MIB view that we had in Figure 5.12 using get-request operation. The manager process starts the operation with a get-request message for object A and receives the response with the value of A filled in. Subsequent requests from the manager are get-next-request type with the object ID of the just received ones. Responses received are the next object ID with its value. Operations continue until Z is received. The subsequent request receives a response with an error message "noSuchName."

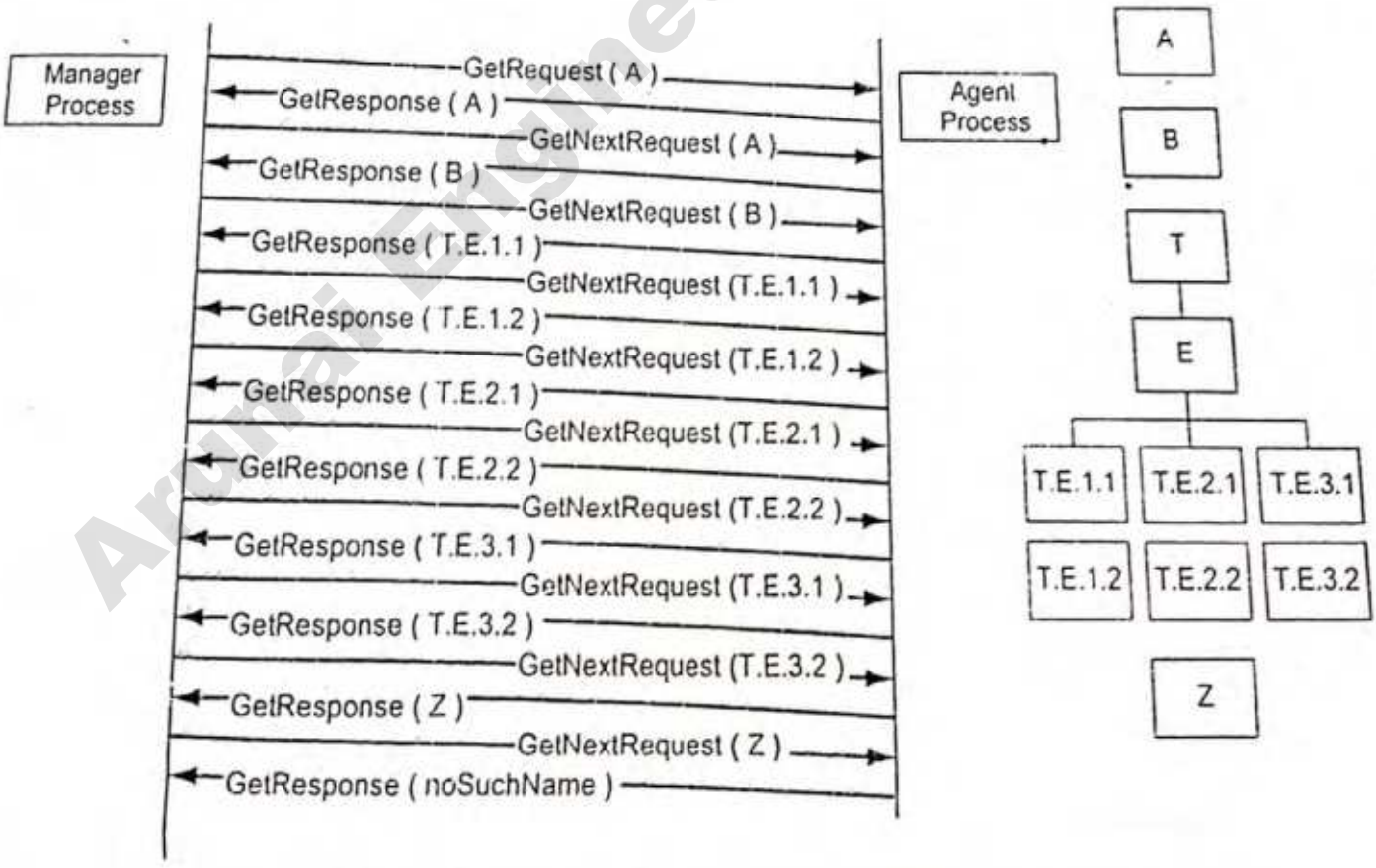


Figure 5.15 Get-Next-Request Operation for a MIB in Figure 5.12

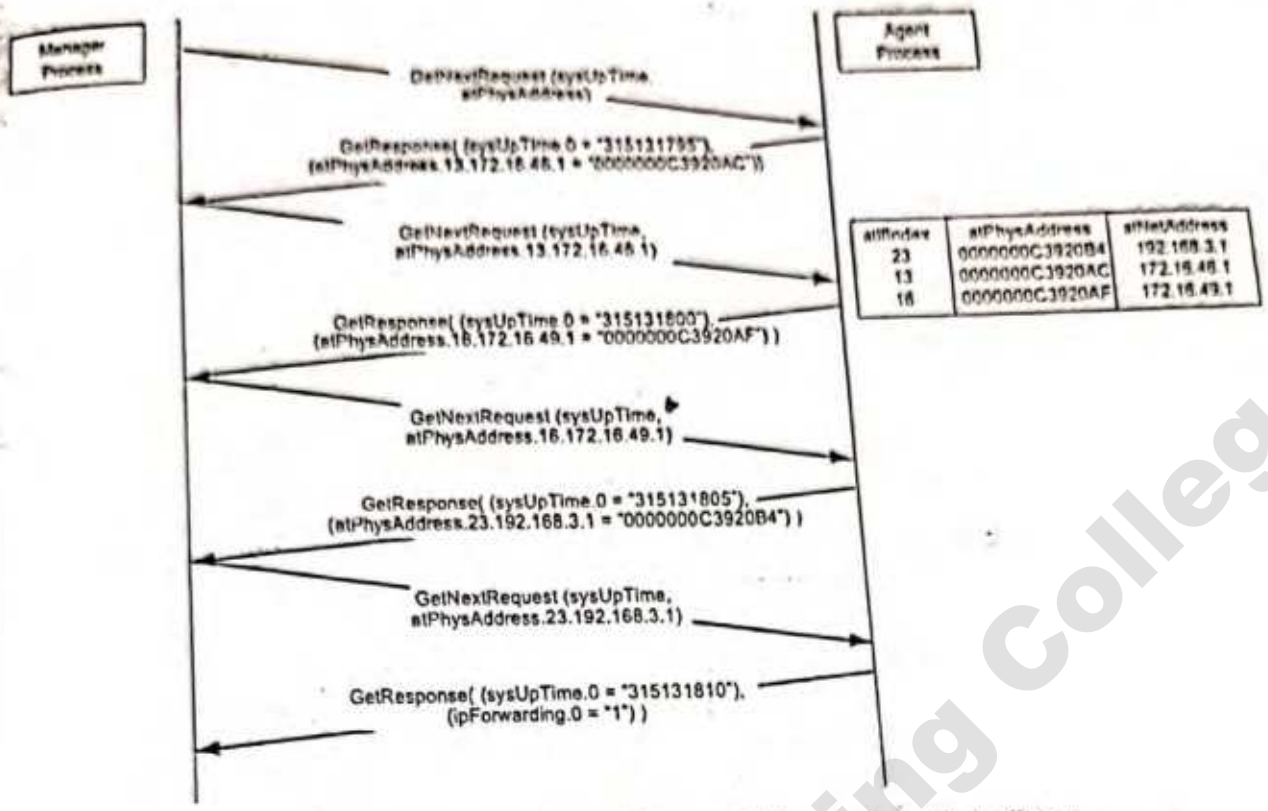


Figure 5.16 GetNextRequest Example with Indices

There are several advantages in using get-next-request. First, we do not need to know the object identifier of the next entity. Knowing the current OBJECT IDENTIFIER, we can retrieve the next one. Next, in the case of an aggregate object, the number of rows is dynamically changing. Thus, we do not know how many rows exist in the table. The get-next-request resolves this problem.

There is also another advantage of the get-next-request. We can use this to build a MIB tree by repeating the request from any node to any node. This is called MIB walk, and is used by a MIB browser in NMS implementation.

Figure 5.16 shows a faster method to retrieve an aggregate object. It shows an Address Translation table with a matrix of three columnar objects, *atIfIndex*, *atPhysAddress*, and *atNetAddress*. The objects *atIfIndex* and *atNetAddress* are the indices that uniquely identify a row. There are three rows in the table. If we use the get-next-request operation shown in Figure 5.15, it would take us ten message exchanges. The *VarBindList* comprises two VarBind name–value pairs, *sysUpTime* and *atPhysAddress*, suffixed with the values of *atIfIndex* and *atNetAddress*. Instead of issuing ten get-next-requests with a single VarBind in the message, the manager generates four GetNextRequest PDUs with a list of two VarBind fields. Although the Address Translation table is relatively stable, in general, a table is dynamic, and hence the time-stamp is requested by including *sysUpTime*.

In this method, the manager has to know the columnar objects of the table. The first query message retrieves the indices automatically. For the Address Translation table, the *atIfIndex* and *atNetAddress* are indices. This is shown in the request and response message OIDs. The first get-next-request message does not contain any operand value. The next three contain the value returned by the response. The fourth and last get-next-request brings the object, *ipForwarding*, which is the first element in the

IP group, which is the next group in Internet MIB. This is because all table entries in the Address Table have been retrieved. It is up to the manager process to recognize this and terminate the process. If the table contained more columns, the *VarBindList* could be expanded and values for all the objects in the next row obtained with each request.

There are more details to this PDU operation and the reader is referred to the references Perkins and McGinnis [1997], RFC [1905], and Stallings [1998].

SNMP PDU Format Examples. We will now look at the PDU for the System group example shown in Figure 5.10 using a sniffer tool. Sniffer is a management tool that can capture packets going across a transmission medium. We have used this tool to "sniff" some SNMP messages to display how messages actually look. We are presenting a series of messages that query a system for its system group data (Figure 5.17). This corresponds to the data shown in Figure 5.10. We then set the missing values for a couple of entities in the group (Figures 5.18 and 5.19) and finally reexamine them (Figure 5.20).

Figure 5.17(a) shows a GetRequest message for the system group values going from the manager noc3.gatech.btc.gatech.edu (noc3, for short), to the agent, noc1.btc.gatech.edu (noc1, for short). The first line shows that it was sent at 13:55:47 from port 164 of noc1 to snmp port of noc3. The tool that was used has actually translated the conventional port number 161 to snmp. The community name is public and the GetRequest message is 111 bytes in length. The SNMP version number is not filled

```
13:55:47.445936 noc3.btc.gatech.edu.164 > noc1.btc.gatech.edu.snmp:
Community = public
GetRequest(111)
Request ID = 1
system.sysDescr.0
system.sysObjectID.0
system.sysUpTime.0
system.sysContact.0
system.sysName.0
system.sysLocation.0
system.sysServices.0
```

(a) Get-Request Message from Manager-to-Agent (Before)

```
13:55:47.455936 noc1.btc.gatech.edu.snmp > noc3.btc.gatech.edu.164:
Community = public
GetResponse(172)
Request ID = 1
system.sysDescr.0 =    "SunOS noc1 5.5.1 Generic_103640-08 sun4u"
system.sysObjectID.0 = E:hp.2.3.10.1.2
system.sysUpTime.0 = 247349530
system.sysContact.0 =    ""
system.sysName.0 =    "noc1 "
system.sysLocation.0 =    ""
system.sysServices.0 = 72
```

(b) Get-Response Message from Agent-to-Manager (Before)

Figure 5.17 Sniffer Data of Get Messages (Incomplete Data in Agent)

```
13:56:24.894369 noc3.btc.gatech.edu.164 > noc1.btc.gatech.edu.snmp:
Community = netman
SetRequest(41)
Request ID = 2
system.sysContact.0 = "Brandon Rhodes"

13:56:24.894369 noc1.btc.gatech.edu.snmp > noc3.btc.gatech.edu.164:
Community = netman
GetResponse(41)
Request ID = 2
system.sysContact.0 = "Brandon Rhodes"
```

Figure 5.18 Sniffer Data of Set-Request and Response for System Contact

```
13:56:27.874245 noc3.btc.gatech.edu.164 > noc1.btc.gatech.edu.snmp:
Community = netman
SetRequest(37)
Request ID = 3
system.sysLocation.0 = "BTC NM Lab"

13:56:27.884244 noc1.btc.gatech.edu.snmp > noc3.btc.gatech.edu.164:
Community = netman
GetResponse(37)
Request ID = 3
system.sysLocation.0 = "BTC NM Lab"
```

Figure 5.19 Sniffer Data of Set-Request and Response for System Location

```
14:03:36.788270 noc3.btc.gatech.edu.164 > noc1.btc.gatech.edu.snmp:
Community = public
GetRequest(111)
Request ID = 4
system.sysDescr.0
system.sysObjectID.0
system.s ysUpTime.0
system.sysContact.0
system.sysName.0
system.sysLocation.0
system.sysServices.0
```

(a) Get-Request Message from Manager-to-Agent (After)

Figure 5.20 Sniffer Data of Get Messages (Complete Data in Agent)

```
14:03:36.798269 noc1.btc.gatech.edu.snmp > noc3.btc.gat ech.edu.164:
Community = public
GetResponse(166)
Request ID = 4
system.sysDescr.0 =      "SunOS noc1 5.5.1 Generic_103640 -08 sun4u"
system.sysObjectID.0 = E:hp.2.3.10.1.2
system.sysUpTime.0 = 247396453
system.sysContact.0 = "Brandon Rhodes"
system.sysName.0 = "noc1"
system.sysLocation.0 =  "BTC NM Lab"
system.sysServices.0 = 72
```

(b) Get-Response Message from Agent-to-Manager (After)

Figure 5.20 (continued)

The seven object IDs from *system.sysDescr*.0 to *system.sysServices*.0 all end with zero to indicate that they are single-valued scalar objects. The agent, noc1, sends a GetResponse message of 172 bytes with values filled in for all seven objects. The GetResponse message is shown in Figure 5.17(b). Notice that the values for *sysContact* and *sysLocation* in GetResponse are blank as they have not been entered in the agent. In addition, the request number identified in the GetResponse PDU is the same as the one in the GetRequest PDU.

Figure 5.18 shows the use of the SetRequest message to write the *sysContact* name in noc1 whose value is "Brandon Rhodes." Notice that the community name is changed to netman. The community of netman has the access privilege to write in noc1, and the object, *system.sysContact*, has the read-write access for the netman community. The agent, noc1, makes the change and sends a GetResponse message back to noc3. Figure 5.19 shows a similar set of messages for setting the entity *sysLocation* with the value "BTC NM Lab."

Figures 5.20(a) and (b) are a repetition of Figure 5.14 of the GetRequest and GetResponse messages. We now see the completed version of the system group data.

5.1.5 SNMP MIB Group

Figure 5.21 shows the MIB tree for the SNMP group, and Table 5.4 gives the description of the entities. Note that OID 7 and OID 23 are not used. The number of transactions in the description column in the table indicates ins and outs of the SNMP protocol entity. All entities except *snmpEnableAuthenTraps* have the syntax, Counter. The implementation of the SNMP group is mandatory—obviously!

5.2 FUNCTIONAL MODEL

There are no formal specifications of functions in SNMPv1 management. Application functions are limited, in general, to network management in SNMP and not to the services provided by the network.

There are five areas of functions (configuration, fault, performance, security, and accounting) addressed by the OSI mode. Some configuration functions, as well as security and privacy-related issues, were addressed as part of the SNMP protocol entity specifications in the previous section. For

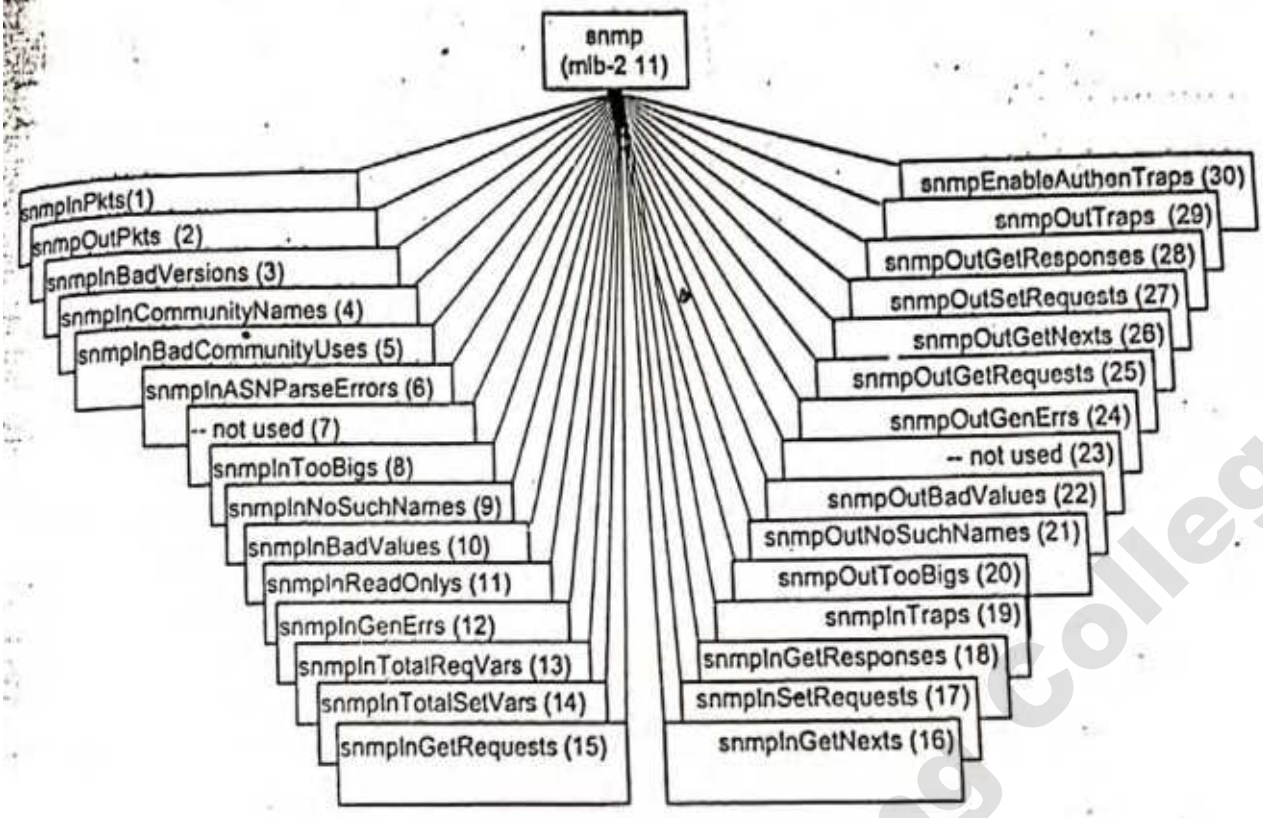


Figure 5.21 SNMP Group

example, the override function of traps is one of the objects in the SNMP group, which has the access privilege of read and write and hence can be set remotely. Security functions are built in as part of the implementation of the protocol entity. Community specifications and authentication scheme partially address these requirements.

The write access to managed objects is limited to implementation in most cases. Thus, configuration management in general is addressed by the specific network management system or by the use of console or telnet to set configurable parameters. We saw the use of the configuration management function in the examples shown in Figures 5.18 and 5.19.

Fault management is addressed by error counters built into the agents. They can be read by the SNMP manager and processed. Traps are useful to monitor network elements and interfaces going up and down.

Performance counters are part of the SNMP agent MIB. It is the function of the SNMP manager to do performance analysis. For example, counter readings can be taken at two instances of time and the data rate calculated. The intermediate manager/agent, such as RMON, can perform such statistical functions, as we will see in the next chapter.

The administrative model in protocol entity specifications addresses security function in basic SNMP.

The accounting function is not addressed by the SNMP model.

Table 5.4 SNMP Group

ENTITY	OID	DESCRIPTION (BRIEF)
snmpInPkts	snmp (1)	Total number of messages delivered from transport service
snmpOutPkts	snmp (2)	Total number of messages delivered to transport service
snmpInBadVersions	snmp (3)	Total number of messages from transport service that are of unsupported version
snmpInBadCommunityNames	snmp (4)	Total number of messages from transport service that are of unknown community name
snmpInBadCommunityUses	snmp (5)	Total number of messages from transport service, not allowed operation by the sending community
snmpInASNParseErrs	snmp (6)	Total number of ASN.1 and BER errors
	snmp (7)	Not used
snmpInTooBigs	snmp (8)	Total number of messages from transport service that have 'tooBig' errors
snmpInNoSuchNames	snmp (9)	Total number of messages from transport service that have 'noSuchName' errors
snmpInBadValues	snmp (10)	Total number of messages from transport service that have 'badValue' errors
snmpInReadOnlys	snmp (11)	Total number of messages from transport service that have 'readOnly' errors
snmpInGenErrs	snmp (12)	Total number of messages from transport service that have 'genErr' errors
snmpInTotalReqVars	snmp (13)	Total number of successful Get-Request and Get-Next messages received
snmpInTotalSetVars	snmp (14)	Total number of objects successfully altered by Set-Request messages received
snmpInGetRequests	snmp (15)	Total number of Get-Request PDUs accepted and processed
snmpInGetNexts	snmp (16)	Total number of Get-Next PDUs accepted and processed
snmpInSetRequests	snmp (17)	Total number of Set-Request PDUs accepted and processed
snmpInGetResponses	snmp (18)	Total number of Get-Response PDUs accepted and processed
snmpInTraps	snmp (19)	Total number of Trap PDUs accepted and processed
snmpOutTooBigs	snmp (20)	Total number of SNMP PDUs generated for which error-status is 'tooBig'
snmpOutNoSuchNames	snmp (21)	Total number of SNMP PDUs generated for which error-status is 'noSuchName'
snmpOutBadValues	snmp (22)	Total number of SNMP PDUs generated for which error-status is 'badValue'
	snmp (23)	Not used
snmpOutGenErrs	snmp (24)	Total number of SNMP PDUs generated for which error-status is 'genErr'
snmpOutGetRequests	snmp (25)	Total number of SNMP Get-Request PDUs generated
snmpOutGetNexts	snmp (26)	Total number of SNMP Get-Next PDUs generated
snmpOutSetRequests	snmp (27)	Total number of SNMP Set-Request PDUs generated
snmpOutGetResponses	snmp (28)	Total number of SNMP Get-Response PDUs generated
snmpOutTraps	snmp (29)	Total number of SNMP Trap PDUs generated
snmpEnableAuthenTraps	snmp (30)	Override option to generate authentication failure traps

Summary

All management operations are done using five messages in SNMPv1. They are get-request, get-next-request, set-request, get-response, and trap. The first three are sent from the manager to the agent and the last two are sent by the agent to the manager.

The SNMP communication model deals with the administrative structure and the five SNMP message PDUs. The administrative model defines the community within which messages can be exchanged. It also defines the access policy as to who has access privilege to what data. The five protocol entities are defined in ASN.1 format and macros. We learned SNMP operations by tracing messages exchanged between manager and agent processes. We then looked inside PDU formats for various messages to learn the data formats.

There is no formal specification for the functional model in SNMP management. However, management functions are accomplished by built-in schemes and managed objects. The administrative model in SNMP and the operations using managed objects are employed to accomplish various functions.

Exercises

1. Three managed hubs with interface id 11–13 (fourth decimal position value) in subnetwork 200.100.100.1 are being monitored by a network management system (NMS) for mean time between failures using the *SysUpTime* in *system* (*internet. mgmt.mib-2.system*) group. The NMS periodically issues the command get-request *object-instance community OBJECT IDENTIFIER* Fill the operands in the three set of requests that the NMS sends out. Use "public" for the *community* variable.
2. You are assigned the task of writing specifications for configuring SNMP managers and agents for a corporate network to implement the access policy. The policy defines a community profile for all managed network components where a public group (community name *public*) can only look at the System group, a privileged group (community name *privileged*) that can look at all the MIB objects, and an exclusive group (community name *exclusive*) that can do a read-write on all allowed components. Present a figure (similar, but not identical, to the flow chart shown in Figure 5.2) showing the paths from the SNMP managers to managed objects of a network component.
3. Fill in the data in the trap PDU format shown in Figure 5.9 for a message sent by the hub shown in Figure 4.2(a) one second after it is reset following a failure. Treat the trap as generic and leave the specific trap field blank. The only *varBind* that the trap sends is *sysUpTime*. (Refer to RFC 1157 and RFC 1215.)
4. An SNMP manager sends a request message to an SNMP agent requesting *sysUpTime* at 8:00 A.M. Fill in the data for the fields of an SNMP PDU shown in Figure 5.5. Please use "SNMP" for the application header, enumerated INTEGER 0 for version-1, and "public" for community name.
5. In Exercise 4, if the SNMP manager sent the request at 8:00 A.M. and the SNMP agent was reset at midnight after a failure, fill in the fields for the SNMP PDU on the response received.
6. An SNMP manager sends a request for the values of the *sysUpTime* in the System group and *ifType* in the Interfaces group for *ifNumber* value of 3. Write the PDUs with the fields filled in for
   (a) the get-request PDU, and
   (b) the get-response PDU with noSuchName error message for *ifType*

SNMP Management: SNMPv2

OBJECTIVES

- *Community-based security*
- *SNMPv2 enhancements*
  - *Additional messages*
  - *Formalization of SMI*
- *Get-bulk request and information-request*
- *SNMP MIB modifications*
- *Incompatibility with SNMPv1*
- *Proxy server*
- *Bilingual manager*

SNMPv1, which was originally called SNMP, was developed as an interim management protocol with OSI as the ultimate network management protocol. A placeholder, CMOT (CMIP over TCP/IP), was created in the Internet Management Information Base (MIB) for migrating from SNMP to CMIP. But the "best-laid plans..." never came about. SNMP caught on in the industry. Major vendors had incorporated SNMP modules in their network systems and components. SNMP now needed further enhancements.

Version 2 of Simple Network Management Protocol, SNMPv2, was developed when it became obvious that OSI network management standards were not going to be implemented in the near future. The working group that was commissioned by the IETF to define SNMPv2 released it in 1996. It is also a community-based administrative framework similar to SNMPv1 defined in STD 15 [RFC 1157], STD 16 [RFC 1155, 1212], and STD 17 [RFC 1213]. Although the original version was known as SNMP, it is now referred to as SNMPv1 to distinguish it from SNMPv2.

6.1 MAJOR CHANGES IN SNMPv2

Several significant changes were introduced in SNMPv2. One of the most significant changes was to improve the security function that SNMPv1 lacked. Unfortunately, after significant effort, due to lack of consensus, this was dropped from the final specifications, and SNMPv2 was released with the rest of the

changes. The security function continued to be implemented on an administrative framework based on the community name and the same administrative framework as in SNMPv1 was adopted for SNMPv2. SNMPv2 Working Group has presented a summary of the community-based Administrative Framework for the SNMPv2 framework, and referred to it as SNMPv2C in RFC 1901. RFC 1902 through RFC 1907 present the details on the framework. There are significant differences between the two versions of SNMP, and unfortunately version 2 is not backward compatible with version 1. RFC 1908 presents implementation schemes for the coexistence of the two versions.

The basic components of network management in SNMPv2 are the same as version 1. They are the agent and the manager, both performing the same functions. The manager-to-manager communication, shown in Figure 4.8, is formalized in version 2 by adding an additional message. Thus, the organizational model in version 2 remains essentially the same. In spite of the lack of security enhancements, major improvements to the architecture have been made in SNMPv2. We will list some of the highlights that would motivate the reader's interest in SNMPv2.

Bulk Data Transfer Message: Two significant messages were added. The first is the ability to request and receive bulk data using the get-bulk message. This speeds up the get-next-request process and is especially useful to retrieve data from tables.

Manager-to-Manager Message: The second additional message deals with interoperability between two network management systems. This extends the communication of management messages between management systems and thus makes network management systems interoperable.

Structure of Management Information (SMI): In SNMPv1, SMI is defined as STD 16, which is described in RFCs 1155 and 1212, along with RFC 1215, which describes traps. They have been consolidated and rewritten in RFCs 1902–1904 for SMI in SNMPv2. RFC 1902 deals with SMIv2, RFC 1903 with textual conventions, and RFC 1904 with conformances.

SMIv2 is divided into three parts: module definitions, object definitions, and trap definitions. An ASN.1 macro, MODULE-IDENTITY, is used to define an information module. It concisely conveys the semantics of the information module. The OBJECT-TYPE macro defines the syntax and semantics of a managed object. The trap is also termed notification and is defined by a NOTIFICATION-TYPE macro.

Textual Conventions are designed to help define new data types. They are also intended to make the semantics consistent and clear to the human reader. Although new data types could have been created using new ASN.1 class and tag, the decision was made to use the existing defined class types and apply restrictions to them.

Conformance Statements help the customer objectively compare features of various products. It also keeps vendors honest in claiming their product as being compatible with a given SNMP version. Compliance defines a minimum set of capabilities. Additional capabilities may be offered as options in the product by vendors.

Table Enhancements: Using a newly defined columnar object with a Syntax clause, *RowStatus*, conceptual rows could be added to or deleted from an aggregate object table. Further, a table can be expanded by augmenting another table to it, which is helpful in adding additional columnar objects to an existing aggregate object.

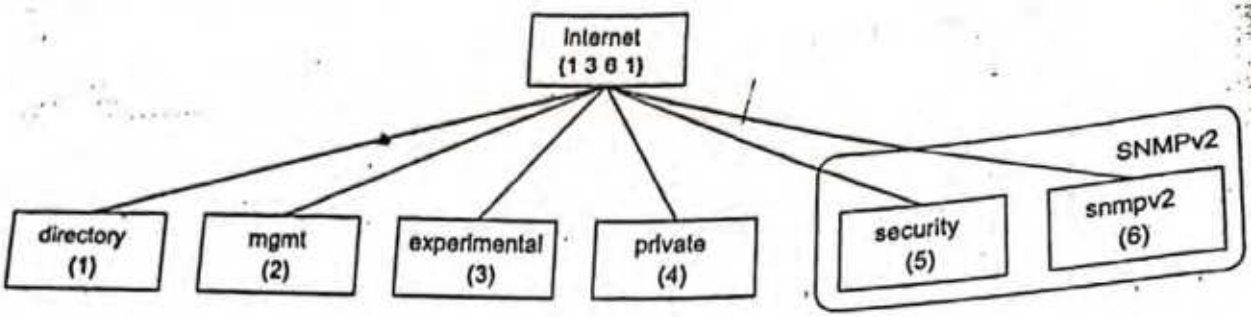


Figure 6.1 SNMPv2 Internet Group

MIB Enhancements: In SNMPv2, the Internet node in the MIB has two new subgroups: security and snmpV2, as shown in Figure 6.1. There are significant changes to System and SNMP groups of version 1. There are changes to the System group made under mib-2 node in the MIB. The SNMP entities in version 2 are a hybrid, with some entities from the SNMP group, and the rest from the groups under the newly created snmpV2 node.

Transport Mappings: There are several changes to the communication model in SNMPv2. Although use of UDP is the preferred transport protocol mechanism for SNMP management, other transport protocols could be used with SNMPv2. The mappings needed to define other protocols on to UDP are the subject of RFC 1906.

6.2 SNMPv2 SYSTEM ARCHITECTURE

SNMPv2 system architecture looks essentially the same as that of version 1, as shown in Figure 4.9. However, there are two significant enhancements in SNMPv2 architecture, which are shown in Figure 6.2. First, there are seven messages instead of five as in Figure 4.9. Second, two manager applications can communicate with each other at the peer level. Another message, report message, is missing from Figure 6.2. This is because even though it has been defined as a message, SNMPv2 Working Group did not specify its details. It is left for the implementers to generate the specifications. It is not currently being used and is hence omitted from the figure.

The messages *get-request*, *get-next request*, and *set-request* are the same as in version 1 and are generated by the manager application. The message, response, is also the same as get-response in version 1, and is now generated by both agent and manager applications. It is generated by the agent application in response to a get or set message from the manager application. It is also generated by the manager application in response to an *inform-request* message from another manager application.

An *inform-request* message is generated by a manager application and is transmitted to another manager application. As mentioned above, the receiving manager application responds with a response message. This set of communication messages is a powerful enhancement in SNMPv2, since it makes two network management systems interoperable.

The message *get-bulk-request* is generated by manager application. It is used to transfer large amounts of data from the agent to the manager, especially if it includes retrieval of table data. The retrieval is fast and efficient. The receiving entity generates and fills data for each entry in the request and transmits all the data as a response message back to the originator of the request.

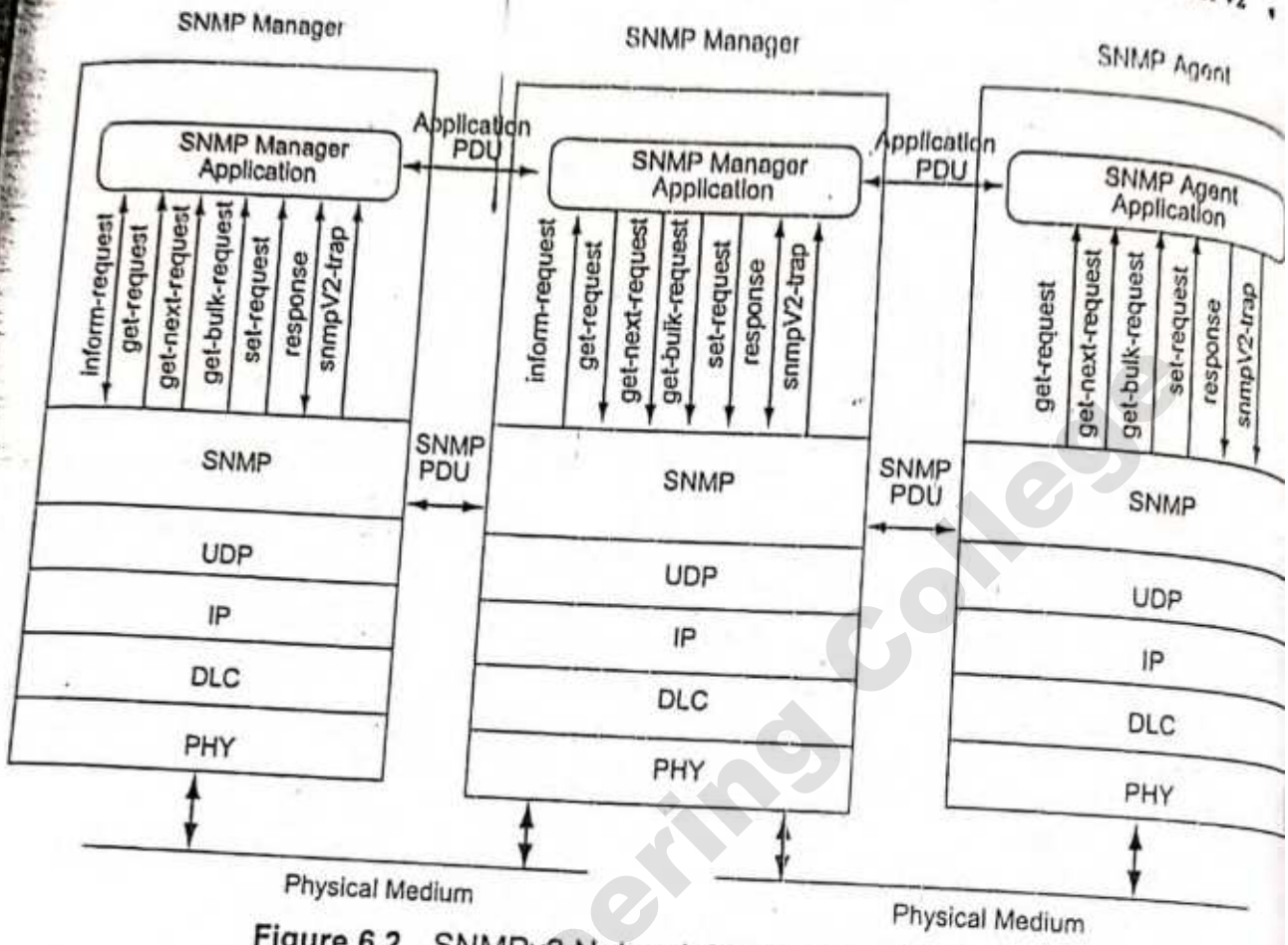


Figure 6.2 SNMPv2 Network Management Architecture

An *SNMPv2-trap event*, known as trap in version 1, is generated and transmitted by an agent proces when an exceptional situation occurs. The destination to which it is sent is implementation-dependen The PDU structure has been modified to be consistent with other PDUs.

Another enhancement in SNMPv2 over version 1 is the mapping of the SNMP layer over multipl transport domains. An example of this is shown in Figure 6.3, in which an SNMPv2 agent riding over connectionless OSI transport layer protocol, Connectionless-Mode Network Service (CLNS), commu nicates with an SNMPv2 manager over the UDP transport layer. RFC 1906, which describes transpor mappings, addresses a few well-known *transport layer mappings*; others can be added using a simila structure.

Details on the MIB relating to SNMPv2 are covered in Section 6.4 and communication protocol asp ects of messages in Section 6.5. Although not a standard, RFC 1283 specifies SNMP over Connection Oriented Transport Service (COTS), a connection-oriented OSI transport protocol. However, SNMP is not specified over connection-oriented Internet protocol, TCP.

6.3 SNMPv2 STRUCTURE OF MANAGEMENT INFORMATION

There are several changes to SMI in version 2, as well as enhancements to SMIv2 over that of SMIv1. As stated earlier, SMIv2 [RFC 1902] is divided into three parts: module definitions, object definitions, and notification definitions.

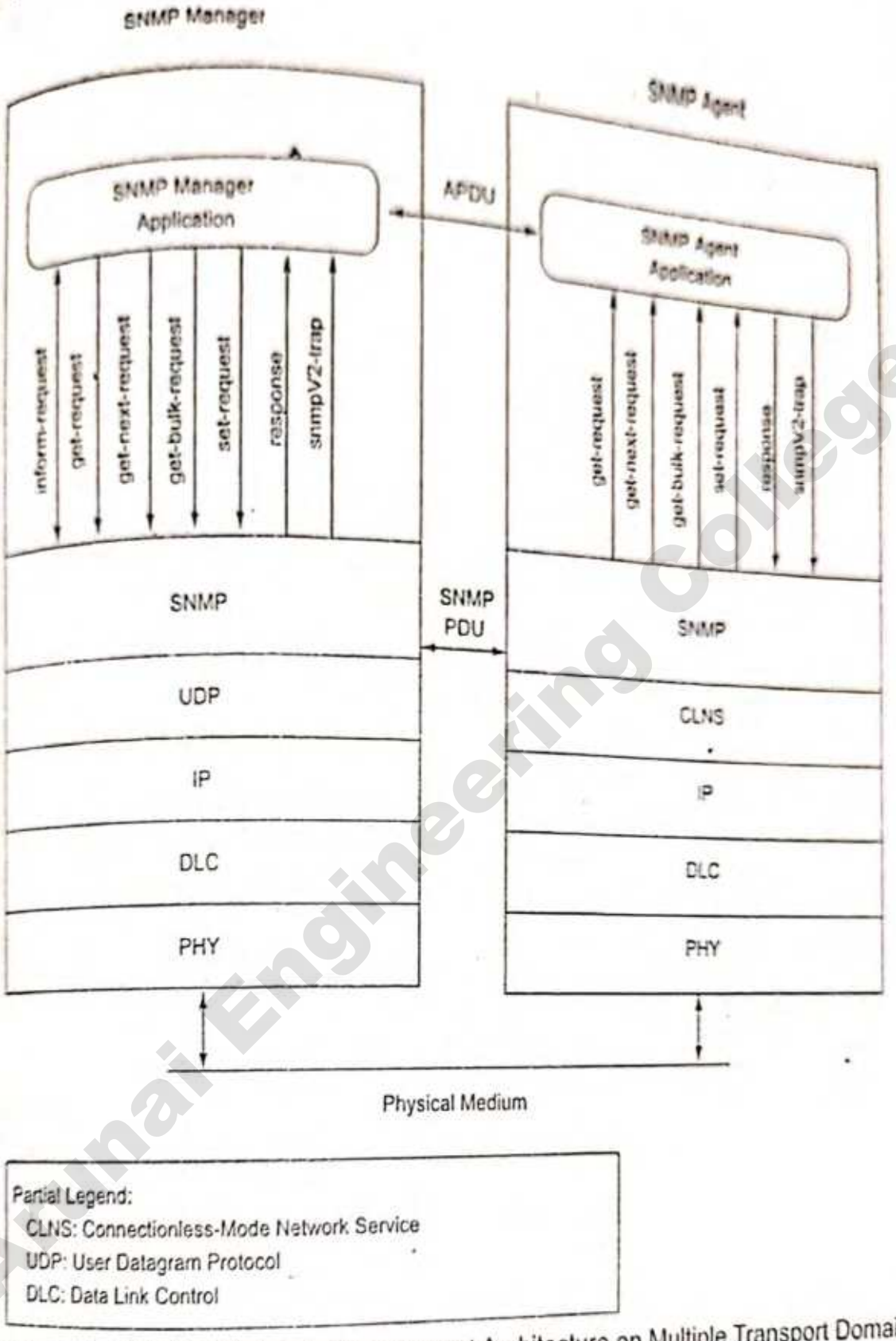


Figure 6.3 SNMPv2 Network Management Architecture on Multiple Transport Domains

We introduced the concept of a module in Section 3.6.1, which is a group of assignments that are related to each other. Module definitions describe the semantics of an information module and are formally defined by an ASN.1 macro, MODULE-IDENTITY.

Object definitions are used to describe managed objects. The OBJECT-TYPE macro that we discussed in Section 4.7.3 is used to define a managed object. OBJECT-TYPE conveys both syntax and semantics of the managed object.

Notification in SMIv2 is equivalent to trap in SMIv1. In SMIv1, trap is formally specified by an ASN.1 macro, TRAP-TYPE. In SMIv2, notification is specified by an ASN.1 macro, NOTIFICATION-TYPE, and conveys both its syntax and semantics.

In addition to the above three parts, there is an additional part defined in SMIv2, which formalizes the assignment of OBJECT IDENTIFIER. Even though we have two assignments in SMIv1, namely, object name and trap, they are not formally structured. In SMIv2, an ASN.1 macro, OBJECT-IDENTITY is introduced for the assignment of object name and notification to OBJECT IDENTIFIER, as shown in Figure 6.4.

6.3.1 SMI Definitions for SNMPv2

Figure 6.4 shows a skeleton of the SMIv2 and the reader is referred to RFC 1902 for a complete set of definitions. We have taken the liberty of presenting the definitions with some additional comments (marked by *) and structural indentations to bring out clearly the BEGIN and END of macros.

Definitions begin with the high-level nodes under the Internet MIB. Two additional nodes, security and SNMPv2, are introduced. The security node is just a placeholder and is reserved for the future. The *snmpV2* node has three subnodes: *snmpDomains*, *snmpProxys*, and *snmpModules*. The MIB tree showing all these nodes defined in SMIv2 is presented in Figure 6.5.

6.3.2 Information Modules

RFC 1902 defines *information module* as an ASN.1 module defining information relating to network management. SMI describes how to use a subset of ASN.1 to define an information module.

There are three kinds of information modules that are defined in SNMPv2. They are MIB modules, compliance statements for MIB modules, and capability statements for agent implementations. This classification scheme does not impose rigid taxonomy in the definition of managed objects. Figure 6.6 shows an example where *conformance information* and *compliance statements* are part of the SNMP group of SNMPv2 MIB. As we shall see later, the SNMP group in SNMPv2 contains some of the objects of version 1 and some new objects and object groups (to be defined later). It also has information on conformance requirements. In the example shown, the mandatory groups in implementing SNMPv2 are *snmpGroup*, *snmpSetGroup*, *systemGroup*, and *snmpBasicNotificationsGroup*. Thus, if a network component vendor claims that its management agent is SNMPv2 compliant, these groups as they are defined in SNMPv2 should be implemented.

MIB specifications contain only compliant statements in them. The *agent-capability statements* are part of implementation in the agent by the vendor. It might be included as part of an "enterprise-specific" module.

The information on SMIv2 has been split into three parts in the documentation. MIB modules for SMIv2 are covered in RFC 1902. The textual conventions to be used to describe MIB modules have been formalized in RFC 1903. The conformance information, which encompasses both compliance and agent capabilities, is covered in RFC 1904.

6.3.3 SNMP Keywords

Keywords used in the specifications of SMIv2 are a subset of ASN.1. But it is a different subset from that of SMIv1. Table 6.1 shows the comparison of keywords used in the two versions. We will address

```
SNMPv2-SMI DEFINITIONS ::=
BEGIN

-- the path to the root
        org             OBJECT IDENTIFIER ::= {iso 3}
                        ...
        private         OBJECT IDENTIFIER ::= {internet 4}
        enterprises     OBJECT IDENTIFIER ::= {private 1}
        security        OBJECT IDENTIFIER ::= {internet 5}
        snmpV2          OBJECT IDENTIFIER ::= {internet 6}
       -- transport domains
        snmpDomains         OBJECT IDENTIFIER ::= {snmpV2 1}
              -- transport proxies
        snmpProxys          OBJECT IDENTIFIER ::= {snmpV2 2}
        --module identities
        snmpModules         OBJECT IDENTIFIER ::= {snmpV2 3}
       -- definitions for information modules
        MODULE-IDENTITY MACRO
        BEGIN
              <clauses> ::= <values>
        END
       -- definitions for OBJECT IDENTIFIER assignments*
        OBJECT-IDENTITY MACRO ::=
        BEGIN
                    <clauses> ::= <values>
        END

              --names of objects
              objectName ::=        OBJECT IDENTIFIER
              notificationName ::= OBJECT IDENTIFIER
       -- syntax of objects
                    <objectSyntax Productions>
                    <dataType Productions>
        -- definition of objects
         OBJECT-TYPE MACRO ::=
         BEGIN
              <clauses> ::= <values>
         END
        -- definition for notification
         NOTIFICATION-TYPE MACRO ::=
         BEGIN
              <clauses> ::= <values>
         END

       -- definition of administration identifiers
            zeroDotZero ::= { 0 0 } -- a value for null identifiers
END
```

Figure 6.4 Definitions of SMI for SNMPv2 (Skeleton)

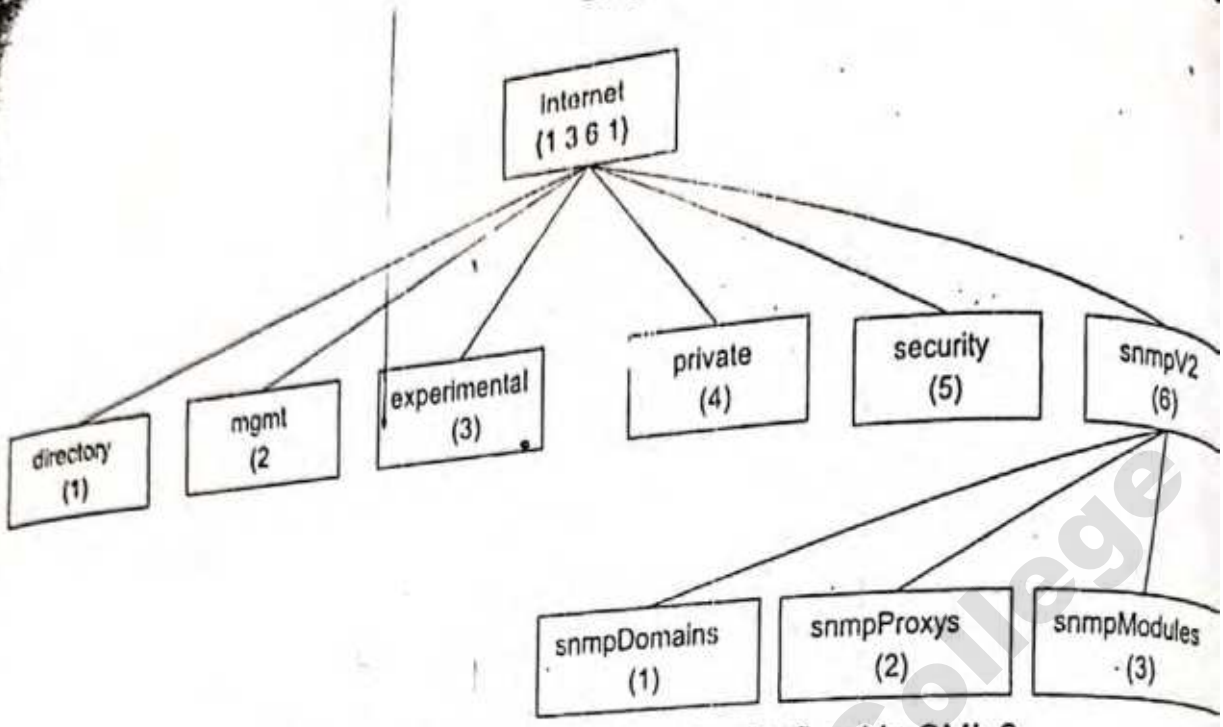


Figure 6.5 SNMPv2 Internet Nodes Defined in SMIv2

the new keywords for specific applications as we discuss them. It is worth noting here that some of general keywords have been replaced with limited keywords. Thus, Counter is replaced by Counter Gauge by Gauge32, and INTEGER by Integer32. The NetworkAddress is deleted from use and o IpAddress is used.

It is also to be noted that reference in IMPORTS clause or in clauses of SNMPv2 macros to an inf mational module is not through "descriptor" as it was in version 1. It is referenced through specifyi its module name, an enhancement in SNMPv2.

It should be observed that the expansion of the ASN.1 module macro occurs during the implement tion phase of a product, and not at run-time.

6.3.4 Module Definitions

The MODULE-IDENTITY macro is added to SMIv2 specifying an informational module. It provid administrative information regarding the informational module as well as revision history. SMIv2 MOD ULE-IDENTITY macro is presented in Figure 6.7.

Figure 6.8 shows an example of a MODULE-IDENTITY macro (a real-world example of a non existent module) for a network component vendor, InfoTech Services, Inc. (isi), which is updating the private-enterprises-isi MIB module {private.enterprises.isi}.

The last updated clause is mandatory and contains the date and time in UTC time format [RFC 1902] "Z" refers to Greenwich Mean Time. The Text clause uses the NVT ASCII character set [RFC 854] which is a printable set. All clauses, except the Revision clause, must be present in the macro.

6.3.5 Object Definitions

The OBJECT-IDENTITY macro has been added in SMIv2 and is used to define information about an OBJECT-IDENTIFIER. It is presented in Figure 6.9. The STATUS clause has one of three values current, deprecated, or obsolete. The value *mandatory* in SMIv1 is replaced with the value *current* in SMIv2. The value *optional* is not used in SMIv2. The new value, *deprecated*, has been added to defin

```
SNMPv2-MIB DEFINITIONS ::=
BEGIN
    ...          ...          ...          ...
snmpMIB     MODULE IDENTITY ::= {snmpModules 1}
    ...          ...          ...          ...
snmpMIBObjects    OBJECT IDENTIFIER ::= {snmpMIB 1}
-- the SNMP group
snmp              OBJECT IDENTIFIER ::= {mib-2 11}
snmpInPkts             OBJECT-TYPE ::= {snmp 1}
snmpOutPkts            OBJECT-TYPE ::= { snmp 2}
    ...          ...          ...          ...
snmpSet           OBJECT IDENTIFIER ::= {snmpmibObjects 6}
snmpSetSerialNo        OBJECT-TYPE ::= { snmpSet 1}
-- conformance information
snmpMIBConformance
OBJECT IDENTIFIER ::= {snmpMIB 2}
snmpMIBCompliances
                  OBJECT IDENTIFIER ::= {snmpMIBConformance 1}
snmpMIBGroups     OBJECT IDENTIFIER ::= {snmpMIBConformance 2}
-- compliance statements
snmpBasicCompliance MODULE-COMPLIANCE
      STATUS      current
      DESCRIPTION
                  "The compliance statement for SNMPv2 entities which
                  implement the SNMPv2 MIB."
      MODULE      -- this module
            MANDATORY-GROUPS {snmpGroup, snmpSetGroup,
                              systemGroup,
                              snmpBasicNotificationsGroup}
            GROUP       snmpCommunityGroup
            DESCRIPTION
                  "This group is mandatory for SNMPv2 entities which support
                  community-based authentication."
      ::= {snmpMIBCompliances 2 }
-- units of conformance
snmpGroup   OBJECT-GROUP    ::= {snmpMIBGroups 8}
snmpCommunityGroup     OBJECT-GROUP ::= {snmpMIBGroups 9}
snmpObsoleteGroup OBJECT-GROUP ::= {snmpMIBGroups 10}
    ...          ...          ...          ...
END
```

Figure 6.6 Example of the SNMP Group including Conformance and Compliance in SNMPv2 MIB

Table 6.1 SNMP Keywords

KEYWORD	SNMPV1	SNMPV2
ACCESS	Y	Y
AGENT-CAPABILITIES	N	Y
AUGMENTS	N	Y
BEGIN	Y	Y
BITS	N	Y
CONTACT-INFO	N	Y
CREATION-REQUIRES	N	Y
Counter	Y	N
Counter32	N	Y
Counter64	N	Y
DEFINITIONS	Y	Y
DEFVAL	Y	Y
DESCRIPTION	Y	Y
DISPLAY-HINT	N	Y
END	Y	Y
ENTERPRISE	Y	N
FROM	Y	Y
GROUP	N	Y
Gauge	Y	N
Gauge32	N	Y
IDENTIFIER	Y	Y
IMPLIED	N	Y
IMPORTS	Y	Y
INCLUDES	N	Y
INDEX	Y	Y
INTEGER	Y	Y
Integer32	N	Y
IpAddress	Y	Y
LAST-UPDATED	N	Y
MANDATORY-GROUPS	N	Y
MAX-ACCESS	N	Y
MIN-ACCESS	N	Y
MODULE	N	Y
MODULE-COMPLIANCE	N	Y
MODULE-IDENTITY	N	Y
NOTIFICATION-GROUP	N	Y

Table 6.1 (continued)

KEYWORD	SNMPV1	SNMPV2
NOTIFICATION-TYPE	N	Y
NetworkAddress	Y	N
OBJECT	Y	Y
OBJECT-GROUP	N	Y
OBJECT-IDENTITY	N	Y
OBJECT-TYPE	Y	Y
OBJECTS	N	Y
OCTET	Y	Y
OF	Y	Y
ORGANIZATION	N	Y
Opaque	Y	Y
PRODUCT-RELEASE	N	Y
REFERENCE	Y	Y
REVISION	N	Y
SEQUENCE	Y	Y
SIZE	Y	Y
STATUS	Y	Y
STRING	Y	Y
SUPPORTS	N	Y
SYNTAX	Y	Y
TEXTUAL-CONVENTION	N	Y
TRAP-TYPE	Y	N
TimeTicks	Y	Y
UNITS	N	Y
Unsigned32	N	Y
VARIABLES	Y	N
VARIATION	N	Y
WRITE-SYNTAX	N	Y

objects that are required to be implemented in the current version, but may not exist in future versions of SNMP. This allows for backward compatibility during the transition between versions.

Although the REFERENCE clause was used only in an OBJECT-TYPE construct in SMIv1, it is used in many constructs in version 2.

Let us extend our hypothetical example of InfoTech Services and suppose that ISI makes a class of router products. It is given an OBJECT IDENTIFIER as isiRouter OBJECT IDENTIFIER ::= {private. enterprises.isi 1}. The class of router products can be specified at a high level using the OBJECT-IDEN-

```
MODULE-IDENTITY MACRO ::=
BEGIN
      TYPE NOTATION ::=
                        "LAST-UPDATED" value (Update UTCTime)
                        "ORGANIZATION" Text
                        "CONTACT-INFO" Text
                        "DESCRIPTION" Text
                        RevisionPart
      VALUE NOTATION ::=
                        value (VALUE OBJECT IDENTIFIER)
      RevisionPart ::=  Revisions | empty
      Revisions ::= Revision | Revisions Revision
      Revision ::=
            "REVISION" value (UTCTime)
            |"DESCRIPTION" Text
      -- uses the NVT ASCII character set
      Text ::= """ string """
END
```

Figure 6.7 MODULE-IDENTITY Macro

```
isiMIBModule      MODULE-IDENTITY
      LAST-UPDATED    "9802101100Z"
      ORGANIZATION    "InfoTech Services, Inc."
      CONTACT-INFO    "Mani Subramanian
                      Tele: 770-111-1111
                      Fax: 770-111-2222
                      email: manis@bellsouth.net"
      DESCRIPTION     "Version 1.1 of the InfoTech Services MIB module"
      Revision        "9709021500Z"
      DESCRIPTION     "Revision 1.0 on September 2, 1997 was a draft
                      version"
```

Figure 6.8 Example of MODULE-IDENTITY Macro

TITY macro as shown in Figure 6.10(a). The status of the *isiRouter* is current and is described as an 8-slot IP router. A reference is given for obtaining the details.

A specific implementation of the router in *isiRouter* class of products is *routerIsi123*. This is a managed object specified by the OBJECT-TYPE macro shown in Figure 6.10(b). We are already familiar with the OBJECT-TYPE macro by now.

Let us make sure that we clearly understand the terminology used with the term OBJECT. OBJECT IDENTIFIER defines the administrative identification of a node in the MIB. The OBJECT IDENTITY macro is used to assign an object identifier value to the object node in the MIB. The OBJECT-TYPE is a macro that defines the *type* of a managed object. It is also used to describe a new type of object. As we have learned in the previous chapters, an *object instance* is a specific instance of the *object (type)*. Thus a specific instance of the *routerIsi123* could be identified by its IP address 10.1.2.3.

```
OBJECT-IDENTITY MACRO ::=
BEGIN
    TYPE NOTATION ::=
                    "STATUS"        Status
                    "DESCRIPTION"   Text
                    ReferPart

    VALUE NOTATION ::=
                    value (VALUE OBJECT IDENTIFIER)

    Status ::=      "current" | "deprecated" | "obsolete"
    ReferPart ::=   "REFERENCE" Text | empty
    Text ::=        """string """
END
```

Figure 6.9 OBJECT-IDENTITY Macro

```
isiRouter    OBJECT-IDENTITY
    STATUS          current
    DESCRIPTION     "An 8-slot IP router in the IP router family."
    REFERENCE       "ISI Memorandum No. ISI-R123 dated January 20,
                    1997"
    ::= {private.enterprises.isi 1}
```

(a) Example of an OBJECT-IDENTITY Macro

```
routerIsi123  OBJECT-TYPE
    SYNTAX          DisplayString
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION     "An 8-slot IP router that can switch up to
                    100 million packets per second."
    ::= {isiRouter 1}
```

(b) Example of an OBJECT-TYPE Macro

Figure 6.10 Example of OBJECT-IDENTITY and OBJECT-TYPE Macros

Comparing Figure 6.10(a) with Figure 6.10(b) we observe the difference between OBJECT-IDENTITY and OBJECT-TYPE. The status clause appears in both. The description clause that also appears in both describes different aspects of the object. The OBJECT-IDENTITY describes the high-level description; whereas the OBJECT-TYPE description focuses on the details needed for implementation.

Let us now visualize the router in Figure 6.10 with several slots for interface cards. We want to define the parameters associated with each interface. The parameters that are managed objects (or entities) are defined by an aggregate object, *IfTable*. For example, the *ifNumber* for our router example could be 32 if the router has eight slots and each card has four ports.

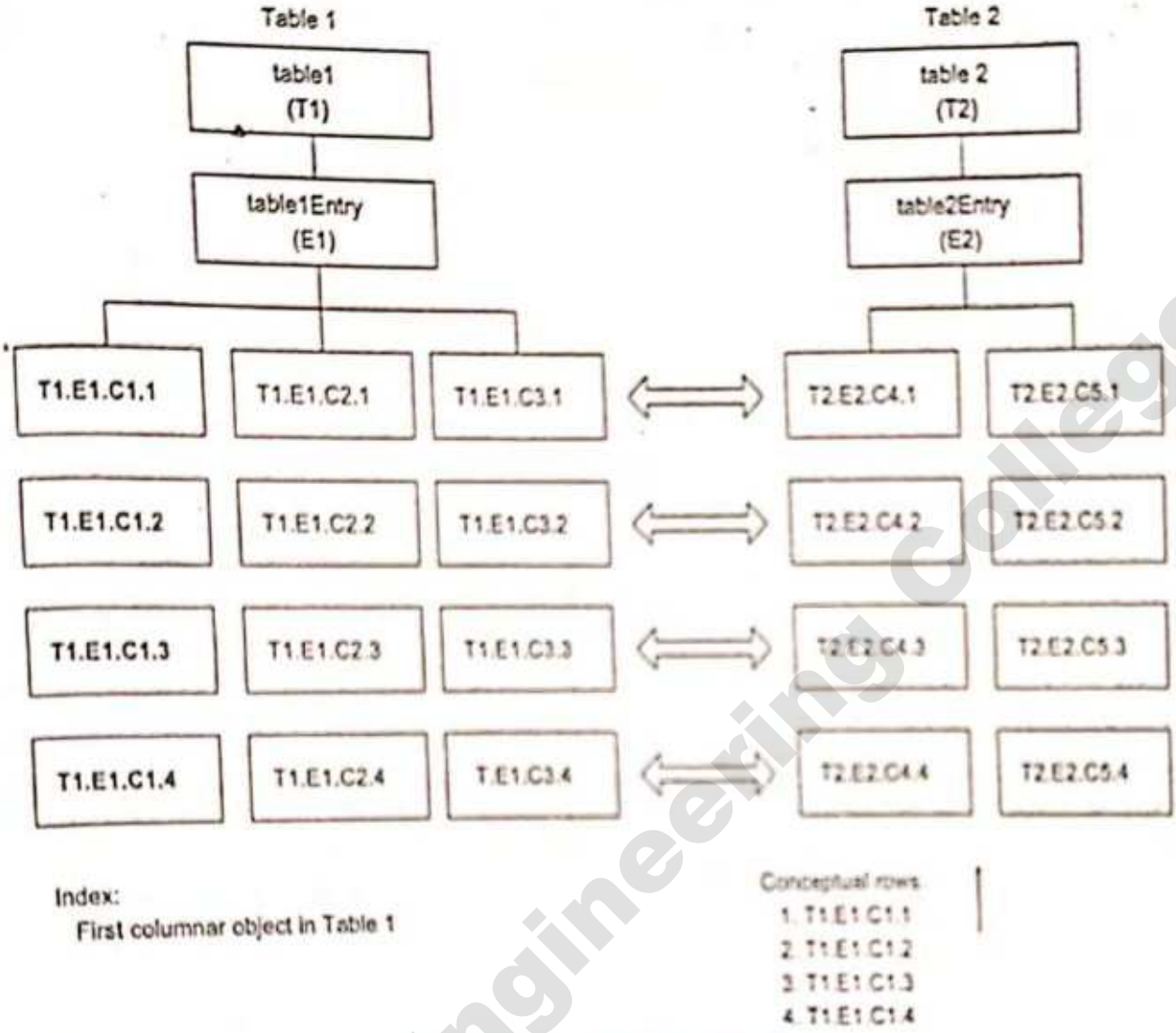


Figure 6.11 Augmentation of Tables

SMIv2 extends the concept table for an aggregate object from a single table to multiple tables. This allows for expansion of managed objects when the number of columnar objects needs to be increased, or when the objects are best organized by grouping them hierarchically. Let us first consider the case of adding columnar objects to an existing table with the following restrictions: (a) the number of conceptual rows is not affected by the addition; (b) there is one-to-one correspondence between the rows of the two tables; and (c) the INDEX of the second table is the same as that of the first table. This is shown in Figure 6.11.

Table 1 is called the aggregate object *table1* and has three columns and four rows; and Table 2 is called the aggregate object *table2* and has two columns and four rows. There is a one-to-one correspondence in rows between the two tables. The row object for table1 is *table1Entry*, and the row object for table 2 is *table2Entry*. The INDEX is defined in Table 1 for both tables and it is the columnar object T1.E1.C1. We are using the notations T1, E1, C1, etc., for easier visual conceptualization of the instance of an object in a table using the prefixes of table ID (e.g., T1) and entry (e.g., E1). The columnar object notation starts with C (e.g., C1). The value or values suffixed with the columnar object identifier uniquely identifies the row. Thus, the list of objects identified by the index T1.E1.C1.2 is the ones in the second rows of

```
table1Entry OBJECT-TYPE
        SYNTAX          TableT1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in table T1"
        INDEX           {T1.E1.C1}
                ::= { table1 1}
table2Entry OBJECT-TYPE
        SYNTAX          TableT2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in table T2"
        AUGMENTS        {table1Entry}
                ::= {table2 1}
```

Figure 6.12 ASN.1 Constructs for Augmentation of Tables

Tables 1 and 2. The value of the columnar object T2.E2.C4 in Table T2 corresponding to index T1.E1.C1.2 is T2.E2.C4.2. Table 1 is called the **base table**, and Table 2 is the **augmented table**. The indexing scheme comprises two clauses, the INDEX clause and the AUGMENTS clause. The constructs for the rows of the two tables in Figure 6.11 are shown in Figure 6.12. The object *table1Entry* has the INDEX clause and *table2Entry* has the AUGMENTS clause that refers to *table1Entry*. The combination of the two tables still provides four conceptual rows, T1.E1.C1.1 through T1.E1.C1.4 (identified by the index), the same number of rows as in the base table.

Figure 6.13 shows an example of augmentation of tables. We have augmented *ipAddrTable* in the standard MIB with a proprietary table, *IpAugAddrTable* that could add additional information to the rows of the table. *IpAddrTable* is the base table and *ipAugAddrTable* is the augmented table. In a practical case, the *ipAugAddrTable* could add two more columnar objects defining the board and port number associated with the *ipAdEntIfIndex*.

A table with a larger number of rows (**dense table**) can be augmented to the base table with combined indices of both, as shown in Figure 6.14. The INDEX clause for combining unequal-sized tables is the combined indices; i.e., combined columnar objects as the INDEX clause for the added aggregate object. In Figure 6.14, Table 1 consists of two rows and three columnar objects, T1.E1.C1, T1.E1.C2, and T1.E1.C3, with the first columnar object T1.E1.C1 being the index. Table 2 has four rows and two columnar objects, T2.E2.C4 and T2.E2.C5, with its first columnar object, T2.E2.C4, being the index. The combined index for specifying the aggregate object of Table 2 appended to Table 1 is the set of both first columnar objects, T1.E1.C1 and T2.E2.C4. Table 1 is called the **base table** and Table 2 is called the **dependent table**. As we see in Figure 6.14, the combined base table and the dependent table could have a maximum of 8 conceptual rows (multiplication of the rows of the two tables).

Figure 6.15 shows the constructs for augmenting a dense table to a base table. The two table objects, *table1* and *table2*, are nodes under the node table. The *table1Entry* defines a row in *table1* with the columnar object T1.E1.C1 as the index. The *table2Entry* is a row in *table2*. Its index is defined by the indices of both tables, namely T1.E1.C1 and T2.E2.C3.

We can visualize the application of augmentation of a dense table with an example of a router with multiple slots, each slot containing a particular type of board, for example, LEC and Ethernet shown in

```
ipAddrTable  OBJECT-TYPE
        SYNTAX          SEQUENCE OF IpAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The table ..."
        ::= {ip 20}

ipAddrEntry  OBJECT-TYPE
        SYNTAX          IpAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The addressing information"
        INDEX           {ipAdEntAddr}
        ::= {ipAddrTable 1}

ipAugAddrTable     OBJECT-TYPE
        SYNTAX          SEQUENCE OF IpAugAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The augmented table to IP Address Table defining
                        board and port numbers"
        ::= {ipAug 1}

ipAugAddrEntry     OBJECT-TYPE
        SYNTAX          IpAugAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The addressing information..."
        AUGMENTS        {ipAddrEntry}
        ::= {ipAugAddrTable 1}
```

Figure 6.13 Example of Augmentation of Tables

Figure 4.3(c). The slot and the board type will be defined in Table 1. Each board may have a different number of physical ports. The port configuration is defined by Table 2. By using the combination of the two tables, we can specify the details associated with a given port in a given slot.

The third possible scenario in appending an aggregate object to an existing aggregate object is the case where the augmented table has fewer rows than that of the base table. This is called a **sparse dependent table** case and is shown in Figure 6.16. In this example, the index for the second table is the same as that for the base table and the constructs are similar to the ones shown in Figure 6.12 except that the AUGMENTS clause is substituted with the INDEX clause for *table2Entry*. This is shown in Figure 6.17

In SNMPv2, operational procedures were introduced for the creation and deletion of a row in a table. However, prior to discussing these procedures, let us first look at the textual convention that was specified to create a new object type in designing MIB modules. We will return to row creation and deletion in Section 6.3.7.

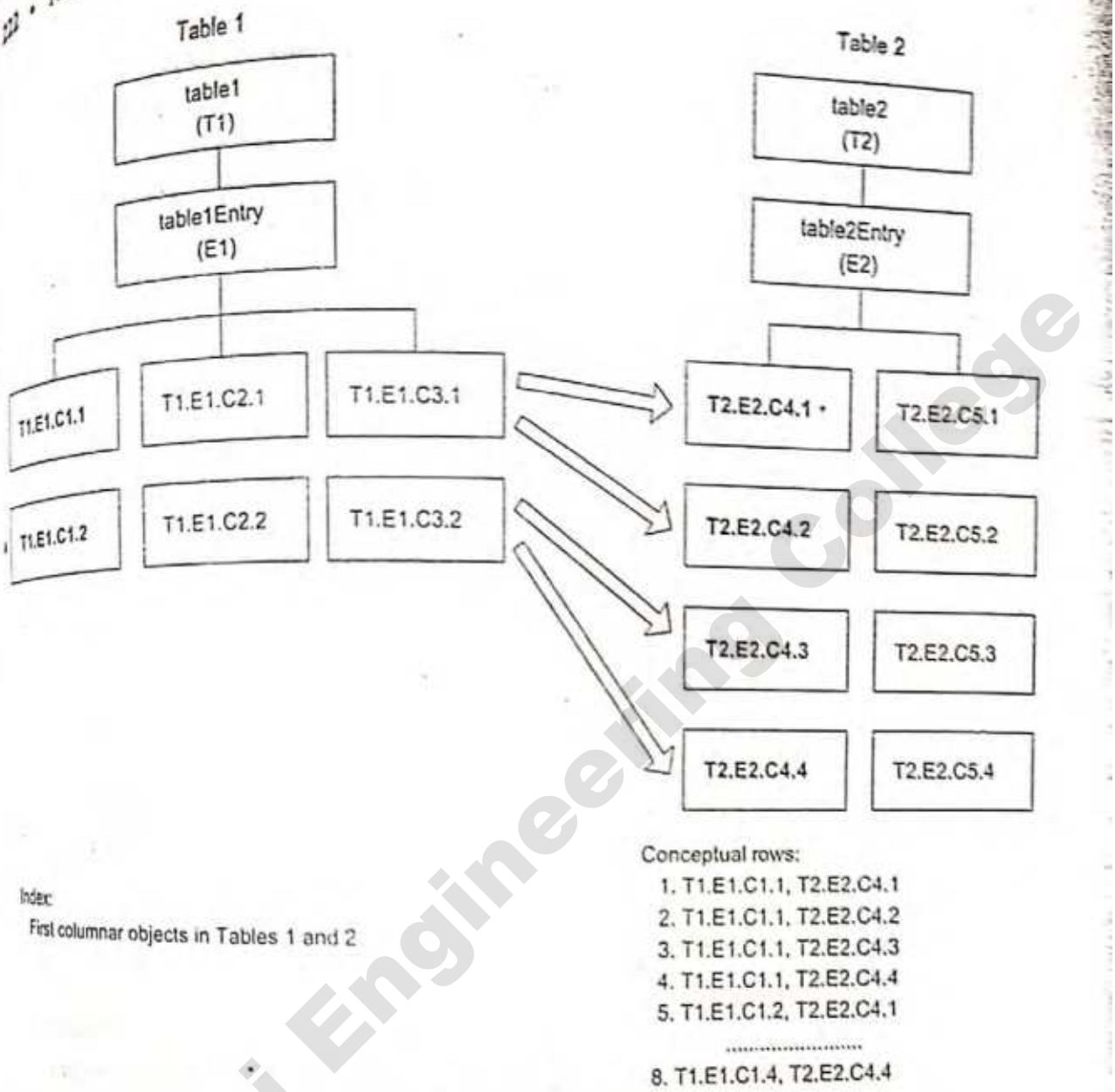


Figure 6.14 Combined Indexing of Tables

6.3.6 Textual Conventions

Textual conventions are designed to help definition of new data types following the structure defined in SMIv2. It is also intended to make the semantics consistent and clear to the human reader. Although new data types could have been created using new ASN.1 class and tag, the decision was made to use the existing defined class types and apply restrictions to them. This is accomplished by defining an ASN.1 macro, TEXTUAL-CONVENTION, in SMIv2.

The TEXTUAL-CONVENTION macro concisely conveys the syntax and semantics associated with a textual convention. SNMP-based management objects defined using a textual convention are encoded by the same Basic Encoding Rules that define their primitive types. However, they do have the special semantics as defined in the macro. For all textual conventions defined in an information module, the name shall be unique and mnemonic, similar to the data type and shall not exceed 64 characters. However, it is usually limited to 32 characters.

```
table1 OBJECT-TYPE
        SYNTAX          SEQUENCE OF table1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "Table 1 under T"
            ::= { table 1}
table1Entry OBJECT-TYPE
        SYNTAX          Table1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in Table 1"
        INDEX           {T1.E1.C1}
            ::= {table1 1}
table2 OBJECT-TYPE
        SYNTAX          SEQUENCE OF table2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "Table 2 under T"
            ::= {table 2}
table2Entry OBJECT-TYPE
        SYNTAX          Table2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in Table 2"
        INDEX           {T1.E1.C1, T2.E2.C4}
            ::= {table2 1}
```

Figure 6.15 ASN.1 Constructs for Augmenting Dense Table

Let us now compare the definition of a type in SMIv1 with SMIv2. The textual convention was defined in SNMPv1 as an ASN.1 type assignment. For example, the textual convention for data type *DisplayString* in SNMPv1, from RFC 1213, is

```
DisplayString ::= OCTET STRING
-- This data type is used to model textual information taken from the NVT
-- ASCII character set. By convention, objects with this syntax are
-- declared as having
-- SIZE (0..255).
```

The same example of DisplayString in SNMPv2 is defined as:

```
DisplayString ::= TEXTUAL-CONVENTION
        DISPLAY-HINT "255a"
        STATUS       current
        DESCRIPTION "Represents textual information taken from the NVT ASCII character
                     set, as defined in pages 4, 10-11 of RFC 854. ...."
        SYNTAX       OCTET STRING (SIZE (0..255) )
```

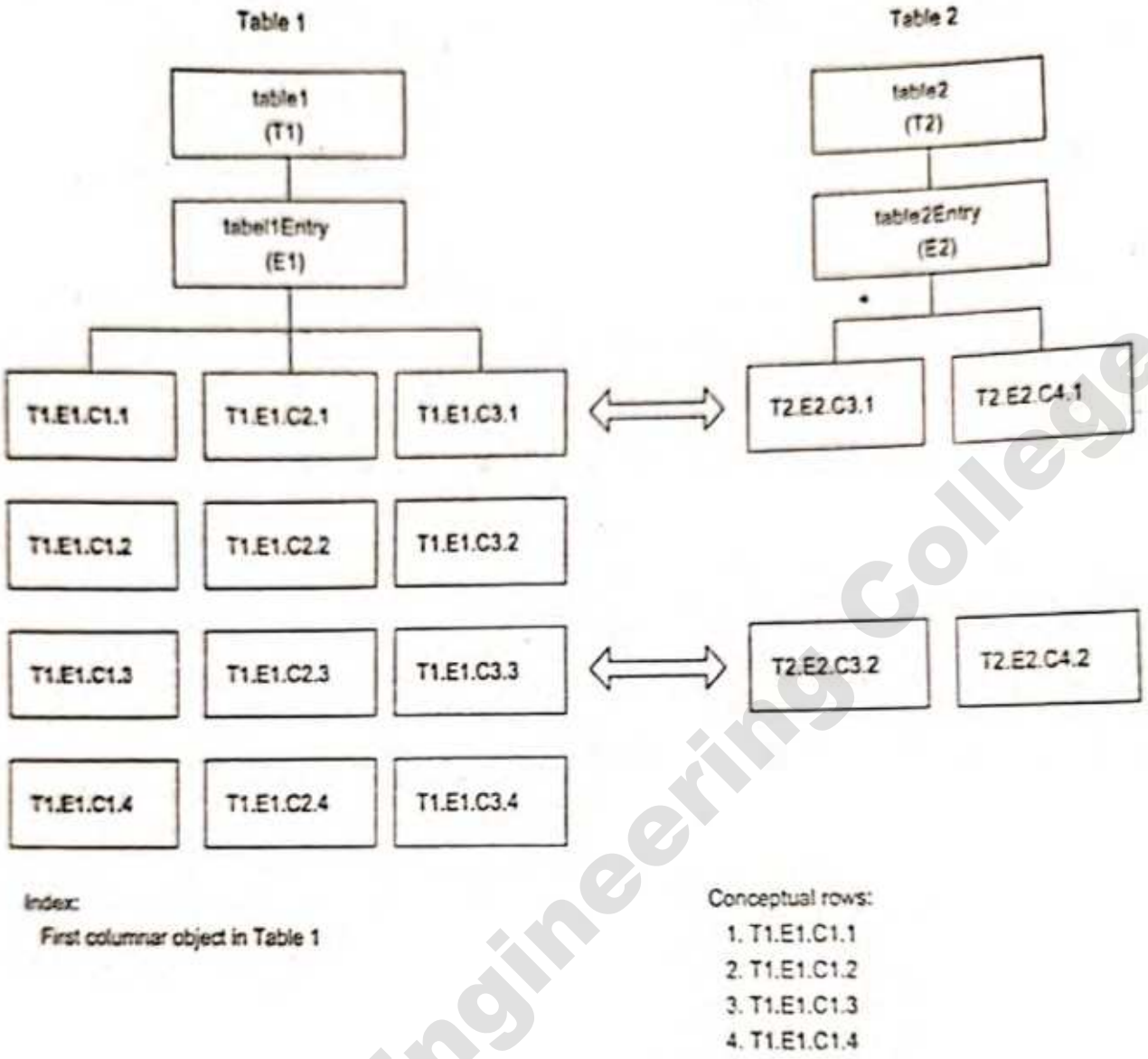


Figure 6.16 Addition of a Sparse Table to a Base Table

As we can see from the above example, the TEXTUAL-CONVENTION in SNMPv2 is defined as data type, and is used to convey the syntax and semantics of a textual convention. The macro for textual conventions is defined in RFC 1903, and a skeleton of it is presented in Figure 6.18. It has the definition of type and value notations with the formalized definition of data types.

All clauses except *DisplayPart* in the TEXTUAL-CONVENTION macro are self-explanatory and represent similar clauses as in SMIv1. The *DISPLAY-HINT* clause, which is optional, gives a hint as to how the value of an instance of an object, with the syntax defined using this textual convention, might be displayed. It is applicable to the situations where the underlying primitive type is either INTEGER or OCTET STRING.

For INTEGER type, the display consists of two parts. The first part is a single character denoting the display format: "a" for ASCII, "b" for binary, "d" for decimal, "o" for octal, and "x" for hexadecimal. It is followed by a hyphen and an integer in the case of decimal display indicating the number of decimal points. For example, a hundredths value of 1234 with DISPLAY-HINT "d-2" is displayed as 12.34.

For OCTET-STRING type, the display hint consists of one or more octet-format specifications. A brief description of each part is shown in Table 6.2. For example, the DISPLAY-HINT "255a" indicates that the *DisplayString* is an ASCII string of up to a maximum of 255 characters.

```
table1 OBJECT-TYPE      SEQUENCE OF table1Entry
    SYNTAX              not-accessible
    MAX-ACCESS          current
    STATUS              "Table 1 under T"
    DESCRIPTION
        ::= { table 1}

table1Entry OBJECT-TYPE
    SYNTAX              Table1Entry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION         "An entry (conceptual row) in Table 1"
    INDEX               {T1.E1.1}
        ::= {table1 1}

table2 OBJECT-TYPE      SEQUENCE OF table2Entry
    SYNTAX              not-accessible
    MAX-ACCESS          current
    STATUS              "Table 2 under T"
    DESCRIPTION
        ::= {table 2}

table2Entry OBJECT-TYPE
    SYNTAX              Table2Entry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION         "An entry (conceptual row) in Table 2"
    INDEX               {table1Entry}
        ::= {table2 1}
```

Figure 6.17 ASN.1 Constructs for Augmenting Sparse Table

```
TEXTUAL-CONVENTION MACRO ::=
BEGIN
    TYPE NOTATION ::=
                    DisplayPart
                    "STATUS"  Status
                    "DESCRIPTION"  Text
                    ReferPart
                    "SYNTAX"  Syntax

    VALUE NOTATION ::=
                    value (VALUE Syntax)

    DisplayPart ::=     "DISPLAY-HINT" Text | empty
    Status ::=          "current" | "deprecated" | "obsolete"

                .....
END
```

Figure 6.18 TEXTUAL-CONVENTION Macro [RFC 1903]

Table 6.2 DISPLAY-HINT for Octet-Format

1	(Optional) repeat indicator "*"	An integer, indicated by *, which specifies how many times the remainder of this octet-format should be repeated
2	Octet length	One or more decimal digits specifying the number of octets
3	Display format	"b" for binary, "x" for hexadecimal, "d" for decimal, "o" for octal, and "a" for ASCII for display
4	(Optional) display separator character	A single character other than a decimal digit or "*" produced after each application of the octet specification.
5	(Optional) repeat terminator character	A single character other than a decimal digit or "*" present if display character is present. Produced after the second and third part.

Table 6.3 shows the types for which textual conventions were specified in SMIv2. A brief description for each type is also given. They are applicable to all MIB modules. Only those textual conventions whose status is current are given in the table. One of the important textual conventions is *RowStatus*, which is used for the creation and deletion of conceptual rows, which we will discuss next.

Table 6.3 SMIv2 Textual Conventions for Initial Data Types

DisplayString	Textual information from NVT ASCII character set [RFC 854]
PhysAddress	Media- or physical-level address
MacAddress	IEEE 802 MAC address
TruthValue	Boolean value; INTEGER {true (1), false (2)}
TestAndIncr	Integer-valued information used for atomic operations
AutonomousType	An independently extensible type identification value
VariablePointer	Pointer to a specific object instance; e.g., syscontact.0, ifInOctets.3
RowPointer	Pointer to a conceptual row
RowStatus	Used to manage the creation and deletion of conceptual rows and is used as the value of the SYNTAX clause for the status column of a conceptual row
TimeStamp	Value of sysUpTime at which a specific occurrence happened
TimeInterval	Period of time, measured in units of 0.01 seconds
DateandTime	Date–time specifications
StorageType	Implementation information on the memory realization of a conceptual row as to the volatility and permanency
Tdomain	Kind of transport service
Taddress	Transport service address

6.3.7 Creation and Deletion of Rows in Tables

The creation of a row and deletion of a row are significant new features in SMIv2. This is patterned after a similar procedure that was developed for RMON, which we will cover in Chapter 8. There are two methods to create a row in a table. The first is to create a row and make it active, which is available immediately. The second method is to create the row and make it available at a later time. This means that we need to know the status of the row as to its availability.

The information on the status of the row is accomplished by introducing a new column, called the *status* column. In Table 6.3, we observe that for the textual convention, RowStatus is used as the value of the SYNTAX clause for the *status* column of a conceptual row. Table 6.4 shows the status with enumerated integer syntax for the six states associated with the row status. The last three states, along with the first one (1, 4, 5, and 6), are those that the manager uses to create or delete rows on the agent. The first three states (1, 2, and 3) are those that are used by the agent to send responses to the manager.

The MAX-ACCESS clause is extended to include "read-create" for the *status* object, which includes read, write, and create privileges. It is a superset of read-write. If a *status* columnar object is present, then no other columnar object of the same conceptual row may have a maximal access of "read-write." But it can have objects with maximum access of read-only and not-accessible. If an index object of a conceptual row is also a columnar object (it does not always have to be), it is called *auxiliary object* and its maximum access is made non-accessible. There could be more than one index object to define a conceptual row in a table.

Let us now analyze the create and delete operations using the conceptual table shown in Figure 6.19. The table, *table1*, originally has two rows and three columns. The first column, *status*, has the value of the status of the row as indicated by the enumerated integer syntax of RowStatus textual convention. The second columnar object, *index*, is the index for the conceptual row of *entry1*; and the third columnar object contains non-indexed data. We will illustrate the two types of row-creation and row-deletion operations by adding a third row and then deleting it.

As we notice from Table 6.4, there are two states for RowStatus, *createAndGo* and *createAndWait*, which are action operations. In the former, the manager sends a message to the agent to create a row and make the *status* active immediately. In the latter operation, the manager sends a message to create a

Table 6.4 RowStatus Textual Convention

STATE	ENUMERATION	DESCRIPTION
active	1	Row exists and is operational
notInService	2	Operation on the row is suspended
notReady	3	Row does not have all the columnar objects needed
createAndGo	4	This is a one-step process of creation of a row, immediately goes into the active state
createAndWait	5	Row is under creation and should not be commissioned into service
destroy	6	Same as Invalid in EntryStatus. Row should be deleted

row, but not to make it active immediately. Figure 6.20 shows the Create-and-Go operation. The manager process initiates a Set-Request-PDU to create a conceptual row with the values given for the three columnar instances of the row. The value for the index column is specified by the VarBind *index* = 3. This is suffixed to the other two columnar objects in the new row to be created. The value of *status* is specified as 4, which is the *createAndGo* state as seen in Table 6.4. The *set-request* message also specifies the default value *DefData* for *data.3*, and thus all the information needed to establish the row and

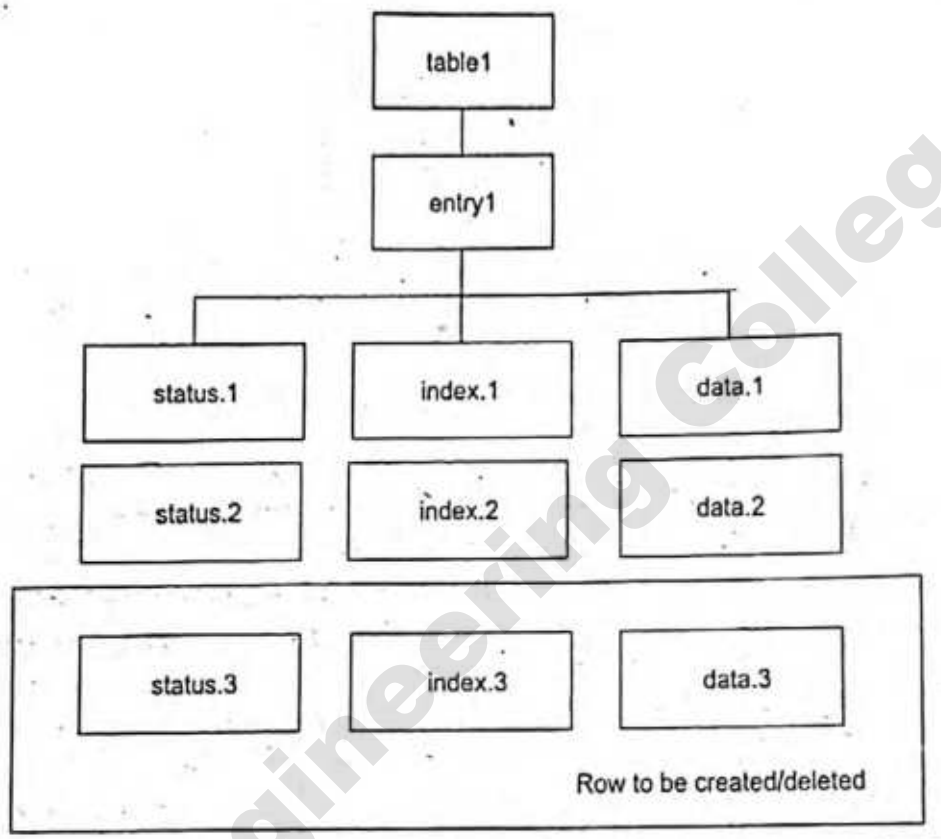


Figure 6.19 Conceptual Table for the Creation and Deletion of a Row

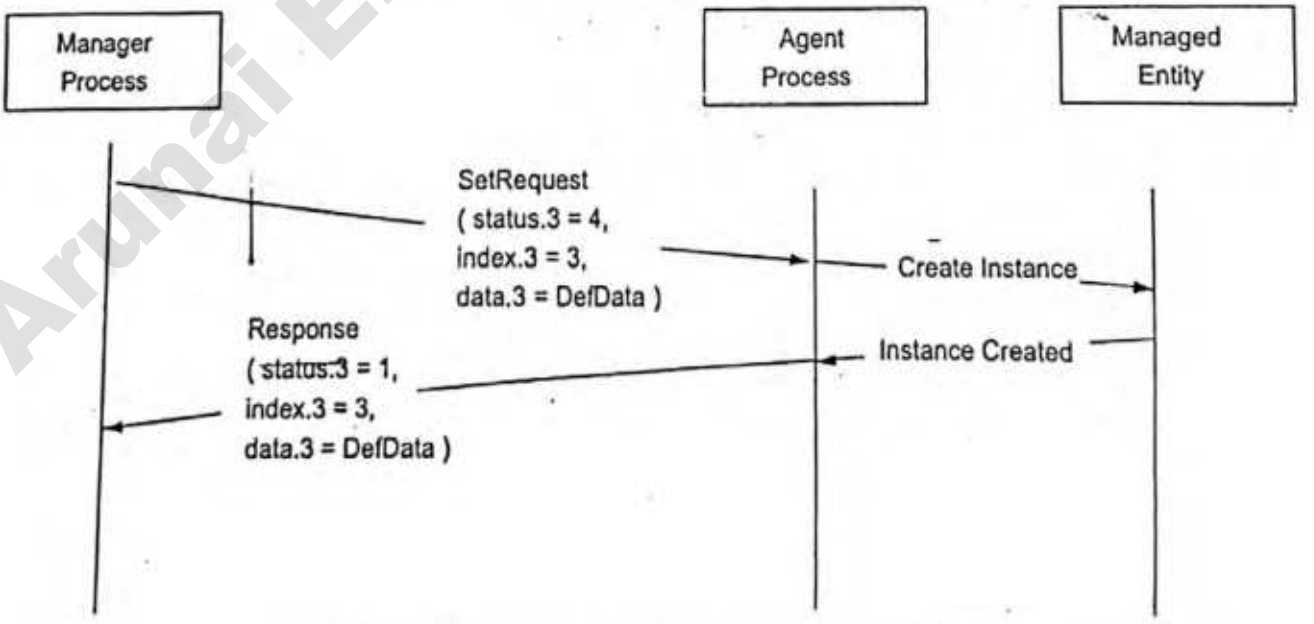


Figure 6.20 Create-and-Go Row Creation

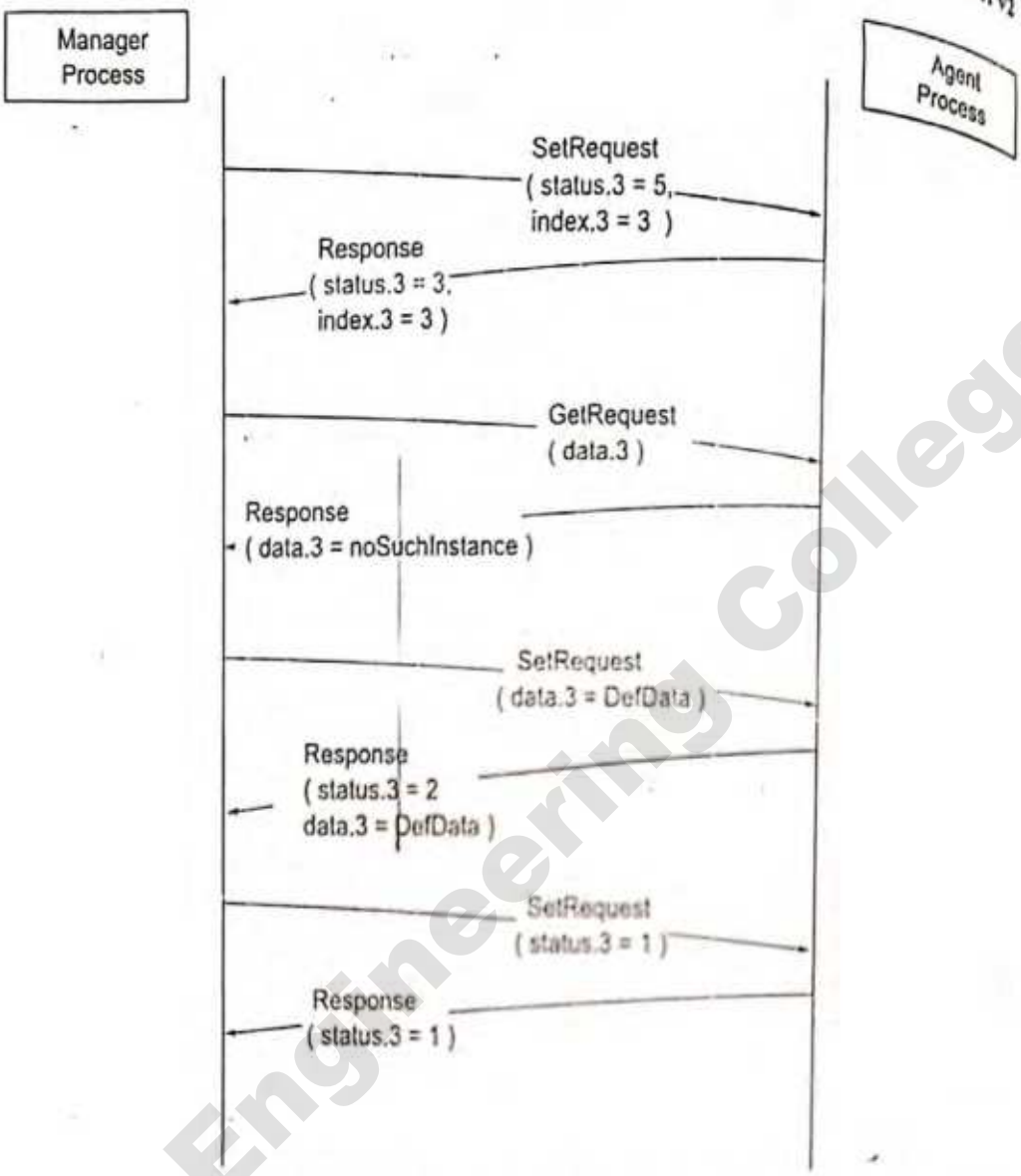


Figure 6.21 Create-and-Wait Row Creation

turn it into an active state is complete. The agent process interacts with the managed entity, creates the instance successfully, and then transmits a response to the manager process. The value of the *status* is 1, which denotes that the row is in an active state. The response also contains the values of the other columnar object instances.

Figure 6.21 presents a scenario for operational sequence in the creation of a row using the Create-and-Wait method. Again, this illustration takes the same scenario of adding the third row to the table shown in Figure 6.17. Only the manager and the agent are shown and not the managed entity in this figure. The manager process sends a Set-Request-PDU to the agent process. The value for *status* is 5, which is to create and wait. The third columnar object expects a default value, which is not in the set-request message. Hence, the agent process responds with a *status* value of 3, which is *notReady*. The manager sends a get-request to get the data for the row. The agent responds with *noSuchInstance* message, indicating that the data value is missing. The manager subsequently sends the value for *data* and receives a response of *notInService* (2) from the agent. The fourth and final exchange of messages in the figure is to activate the row with a *status* value of 1. With each message received from the manager, the agent either validates or sets the instance value on the managed entity.

Table 6.5 Table of States for Row Creation and Deletion

ACTION	A STATUS COLUMN DOES NOT EXIST	B STATUS COLUMN NOTREADY	C STATUS COLUMN NOTINSERVICE	D STATUS COLUMN ACTIVE
Set status column to createAndGo	noError -> D	inconsistent-Value	inconsistent-Value	inconsistent-Value
Set status column to createAndWait	noError, see 1 or wrongValue	inconsistent-Value	inconsistent-Value	inconsistent-Value
Set status column to active	inconsistent-Value	inconsistent-Value or see 2 ->D	noError ->D	noError ->D
Set status column to notInService	inconsistent-Value	inconsistent-Value or see 3 -> C	noError ->C	noError ->C or wrongValue
Set status column to destroy	noError ->A	noError ->A	noError ->A	noError ->A
Set any other column to some value	see 4 —	noError see 1	noError ->C	see 5 ->D

A summary of possible state transitions is given in Table 6.5. The first column lists the action; and the transitions based on the present state are listed in the next four columns.

1. goto B or C, depending on information available to the agent.
2. If other variable bindings included in the same PDU provide values for all columns, which are missing but are required, then return noError and goto D.
3. If other variable bindings included in the same PDU provide values for all columns, which are missing but are required, then return noError and goto C.
4. At the discretion of the agent, the return value may be either: *inconsistentName*: because the agent does not choose to create such an instance when the corresponding RowStatus instance does not exist, or *inconsistentValue*: if the supplied value is inconsistent with the state of some other MIB object's value, or *noError*: because the agent chooses to create the instance.

   If noError is returned, then the instance of the status column must also be created, and the new state is B or C, depending on the information available to the agent. If *inconsistentName* or *inconsistent Value* is returned, the row remains in state A.
5. Depending on the MIB definition for the column table, either *noError* or *inconsistentValue* may be returned.

   NOTE: Other-processing of the set request may result in a response other than *noError* being returned, e.g., wrongValue, *noCreation*, etc.

The operation of deletion of a row is simple. A *set-request* with a value of 6, which denotes *destroy*, for *status*, is sent by the manager process to the agent process. Independent of the current state of the row, the row is deleted and the response sent back by the agent. The instance in the managed entity is deleted in the process. This is shown in Figure 6.22.

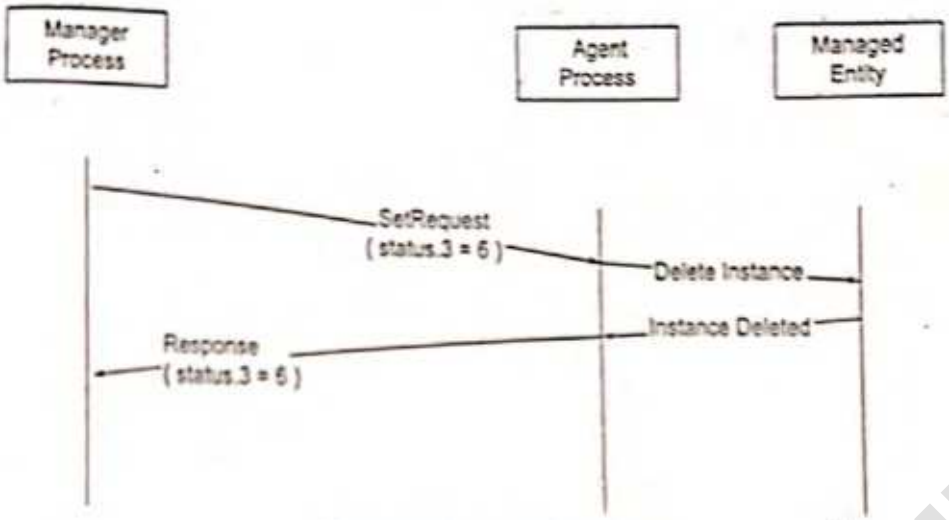


Figure 6.22 Row Deletion

6.3.8 Notification Definitions

The trap information in SMIv1 has been redefined using the NOTIFICATION-TYPE macro in SMIv2. As we will see in Section 6.5, the PDU associated with the trap information is made consistent with other PDUs. The NOTIFICATION-TYPE macro contains unsolicited information that is generated on an exception basis, for example, when set thresholds are crossed. It can be transmitted within either a SNMP-Trap-PDU from an agent or an InformRequest-PDU from a manager. Two examples of a NOTIFICATION-TYPE macro, drawn from RFC 1902 and RFC 1907 are shown in Figure 6.23. The first example, linkUp, is generated by an agent when a link that has been down comes up.

The OBJECTS clause defines the ordered sequence of MIB objects, which are included in the notification. It may or may not be present. The second example, coldStart, in Figure 6.23, has the OBJECTS clause missing and is not needed.

```
linkUp NOTIFICATION-TYPE
        OBJECTS     {ifIndex}
        STATUS      current
        DESCRIPTION
                "A linkUp trap signifies that the SNMPv2 entity, acting in an agent
                role, recognizes that one of the communication links represented in
                its configuration has come up."
        ::= {snmpTraps 4}

coldStart NOTIFICATION-TYPE
        STATUS      current
        DESCRIPTION
                "A coldStart trap signifies that the SNMPv2 entity, acting in
                an agent role, is reinitializing itself such that its configuration is
                unaltered."
        ::= {snmpTraps 1}
```

Figure 6.23 Examples of NOTIFICATION-TYPE Macro

The other two clauses, STATUS and DESCRIPTION, have the usual mappings.

We have not presented here discussions on refined syntax in some of the macros, as well as extension to informational modules. You are referred to RFC 1902 for a treatment of these, which also discusses the conversion of a managed object from the OSI to the SNMP version.

6.3.9 Conformance Statements

RFC 1904 defines SNMPv2 conformance statements for the implementation of network management standards. A product, generally, is considered to be in compliance with a particular standard when it meets the minimum set of features in its implementation. Minimum requirements for SNMPv2 compliance are called module compliance and are defined by an ASN.1 macro, MODULE-COMPLIANCE. It specifies the minimum MIB modules or a subset of modules that should be implemented. The actual MIB modules that are implemented in an agent are specified by another ASN.1 module, AGENT-CAPABILITIES. For the convenience of defining module compliance and agent capabilities, objects and traps have been combined into groups, which are subsets of MIB modules. Object grouping is defined by an ASN.1 macro, OBJECT-GROUP, and the group of traps is defined by the NOTIFICATION-GROUP macro.

Object Group. The OBJECT-GROUP macro defines a group of related objects in a MIB module and is used to define conformance specifications. It is compiled during implementation, not at run-time. The macro is shown in Figure 6.24. The implementation of an object in an agent implies that it executes the get and set operations from a manager. If an agent in SNMPv2 has not implemented an object, it returns a noSuchObject error message.

```
OBJECT-GROUP MACRO
BEGIN
    TYPE NOTATION ::=
                    ObjectsPart
                    "STATUS" Status
                    "DESCRIPTION" Text
                    ReferPart

    VALUE NOTATION ::=
                    value (VALUE OBJECT IDENTIFIER)

    ObjectsPart ::=     "OBJECTS" "{"objects"}"
    Objects ::=         Object | Objects "," Object
    Object ::=          value (Name  Object Name)
    Status ::=          "current" | "deprecated" | "obsolete"
    ReferPart ::=       "REFERENCE" Text  | empty

    -- uses the NVT ASCII character set
    Text ::=            """ string """
END
```

Figure 6.24 OBJECT-GROUP Macro

```
systemGroup     OBJECT-GROUP
    OBJECTS     {sysDescr, sysObjectID, sysUpTime, sysContact, sysName,
                sysLocation, sysServices, sysORLastChange, sysORID,
                sysORUptime, sysORDesc}

    STATUS      current
    DESCRIPTION "The system group defines objects that are common
                to all managed systems."

    ::= {snmpMIBGroups 6}
```

Figure 6.25 Example of an OBJECT-GROUP Macro

The OBJECTS clause names each object contained in the conformance group. Each of the named objects is defined in the same informational module as the OBJECT-GROUP macro and has a MAX-ACCESS clause of "accessible-for-notify," "read-only," "read-write," or "read-create." Every object that is defined in an informational module with a MAX-ACCESS clause other than "not-accessible" is present in at least one object group. This prevents the mistake of adding an object to an information module, but forgetting to add it to a group.

The STATUS, DESCRIPTION, and REFERENCE clauses have the usual interpretations.

An example of an OBJECT-GROUP, systemGroup in SNMPv2, is shown in Figure 6.25. The system group defines the objects, which pertain to overall information about the system. Since it is so basic, it is implemented in all agent and management systems. All seven entities defined as values for OBJECTS should be implemented. There are some new entities, such as *sysORLastChange*, in the group that were not in SNMPv1. These will be addressed when we discuss SMPv2 MIB in the next section.

Notification Group. The notification group contains notification entities, or what was defined as traps in SMIv1. The NOTIFICATION-GROUP macro is shown in Figure 6.26. The macro is compiled during implementation, not during run-time. The value of an invocation of the NOTIFICATION-GROUP macro is the name of the group, which is an OBJECT IDENTIFIER.

An example of NOTIFICATION-GROUP, *snmpBasicNotificationsGroup*, is shown in Figure 6.27. According to this invocation, the conformance group, *snmpBasicNotificationsGroup*, has two notifications: *coldStart* and *authenticationFailure*.

Module Compliance. The MODULE-COMPLIANCE macro, shown in Figure 6.28, defines the minimum set of requirements for implementation of one or more MIB modules. The expansion of the MODULE-COMPLIANCE macro is done during the implementation and not during run-time. The MODULE-COMPLIANCE macro can be defined as a component of the information module or as a companion module.

The STATUS, DESCRIPTION, and REFERENCE clauses are self-explanatory.

The MODULE clause is used to name each module for which compliance requirements are specified. Modules are identified by the module name and its OBJECT IDENTIFIER. The latter can be dropped if the MODULE-COMPLIANCE is invoked within an MIB module and refers to the encompassing MIB module.

There are two CLAUSES of groups that are specified by the MODULE-COMPLIANCE macro. They are MANDATORY-GROUPS and GROUP. As the name implies, the MANDATORY-CLAUSE

```
NOTIFICATION-GROUP MACRO
BEGIN
    TYPE NOTATION ::=
                        NotificationsPart
                        "STATUS" Status
                        "DESCRIPTION" Text
                        ReferPart

    VALUE NOTATION ::=
                        value (VALUE OBJECT IDENTIFIER)

NotificationsPart ::=   "NOTIFICATIONS" "{"Notifications"}"
Notifications ::=       Notification | Notifications "," Notification
Notification ::=        value (Name NotificationName)

    Status ::=          "current" | "deprecated" | "obsolete"
    ReferPart ::=       "REFERENCE" Text | empty

    -- uses the NVT ASCII character set
    Text ::= """ string """

END
```

Figure 6.26 NOTIFICATION-GROUP Macro

```
snmpBasicNotificationsGr    oup NOTIFICATION-GROUP
    NOTIFICATIONS               {coldStart, authenticationFailure}
    STATUS                      current
    DESCRIPTION             "The two notifications which an SNMP-2 entity is
                             required to implement."
    ::= {snmpMIBGroups 7}
```

Figure 6.27 Example of a NOTIFICATION-GROUP Macro

modules have to be implemented for the system to be SNMPv2 compliant. The group specified by the ROUP clause is not mandatory for the MIB module, but helps vendors define specifications of the ances that have been implemented.

When both WRITE-SYNTAX and SYNTAX clauses are present, restrictions are placed on the syntax for the object mentioned in the OBJECT clause. These restrictions are tabulated in Section 9 of RFC 1902.

The *snmpBasicCompliance* macro is an example of a *MODULE-COMPLIANCE* macro and is part of the SNMPv2 MIB presented in Figure 6.6. A system is defined as SNMPv2 compliant if and only if *snmpGroup*, *snmpSetGroup*, *systemGroup*, and *snmpBasicNotificationsGroup* are implemented. The GROUP, *snmpCommunityGroup*, is optional.

```
MODULE-COMPLIANCE MACRO
BEGIN
        TYPE NOTATION ::=
                        "STATUS" Status
                        "DESCRIPTION" text
                        ReferPart
                        ModulePart

        VALUE NOTATION ::=
                        value (VALUE OBJECT IDENTIFIER)

        Status  ::=             "current" | "deprecated" | "obsolete"
        ReferPart ::=           "REFERENCE" Text | empty
        ModulePart ::=          Modules | empty
        Modules ::=             Module | Modules Module
        Module ::=              -- name of module --
                                "MODULE" ModuleName
                                Mandatory Part
                                CompliancePart
        ModuleName ::=          moduleReference ModuleIdentifier | empty
-- must not be empty unless contained in MIB module
ModuleIdentifier ::= value (ModuleID OBJECT IDENTIFIER) | empty
        MandatoryPart ::=       "MANDATORY-GROUPS" "{" Groups"}"
                                | empty
        Groups ::=              Group | Groups "," Group
        Group ::=               value (Group OBJECT IDENTIFIER)
        CompliancePart ::=      Compliances | empty
        Compliances ::=         Compliance | Compliances compliance
        Compliance ::=          ComplianceGroup | Object
        ComplianceGroup ::=     "GROUP" value (Name OBJECT IDENTIFIER)
                                "DESCRIPTION" Text
        Object ::=              "OBJECT" value (Name ObjectName)
                                SyntaxPart
                                WriteSyntaxPart
                                AccessPart
                                "DESCRIPTION" Text
        --must be a refinement for object's SYNTAX clause
        SyntaxPart ::=          "SYNTAX" type (SYNTAX) | empty
        --must be a refinement for object's SYNTAX clause
WriteSyntaxPart ::= "WRITE-SYNTAX" type (WriteSYNTAX) | empty
        AccessPart ::=          "MIN-ACCESS" Access | empty
        Access ::=              "not-accessible" | "accessible-for-notify" |
                                "read-only" | "read-write" | "read-create"
                                -- uses the NVT ASCII character set
                              * Text ::= """ string """

END
```

Figure 6.28 MODULE-COMPLIANCE Macro

```
AGENT-CAPABILITIES
BEGIN
        TYPE NOTATION ::=
                        "PRODUCT-RELEASE" Text
                        "STATUS" Status
                        "DESCRIPTION" Test
                        ReferPart
                        ModulePart

        VALUE NOTATION ::=
                        Value (VALUE OBJECT IDENTIFIER)

        Status ::=          "current" | "obsolete"
        ReferPart ::=       "REFERENCE" | empty
        ModulePart ::=      Modules | empty
        Modules ::=         Module | Modules Module
        Module ::=          -- name of module --
                            "SUPPORT" ModuleName
                            "INCLUDES" "{"Groups"}"
                            VariationsPart

        ...          ...          ...          ...

END
```

Figure 6.29 AGENT-CAPABILITIES Macro (Skeleton)

Agent Capabilities. The AGENT-CAPABILITIES macro is lengthy and the reader is referred to RFC 1904 for exact specifications. A skeleton of the macro and significant points of the macro are covered here and are shown in Figure 6.29.

The AGENT-CAPABILITIES macro for the router example given in Figure 6.10 is shown in Figure 6.30. Note that *snmpMIB* model, which is SNMPv2-MIB, includes *system* and *snmp* MIBs. Those MIBs and the associated groups are supported by the router. Other standard MIBs and groups supported by the router are indicated in Figure 6.30.

6.4 SNMv2 MANAGEMENT INFORMATION BASE

As mentioned in Section 6.2 and shown in Figure 6.5 two new MIB modules, security and SNMPv2, have been added to the Internet MIB. The SNMPv2 module has three submodules: snmpDomains, snmpProxys, and snmpModules. snmpDomains extends the SNMP standards to send management messages over transmission protocols other than UDP, which is the predominant and preferred way of transportation [RFC 1906]. Since UDP is the preferred protocol, systems that use another protocol need a proxy service to map on to UDP. Not much work has been done on snmpProxys, as of now.

There are changes made to the core MIB-II defined in SNMPv1. Figure 6.31 presents an overview of the changes to the Internet MIB and their relationship. The system module and the *snmp* module unde

```
routerIsi123 AGENT-CAPABILITIES
    PRODUCT-RELEASE     "InfoTech Router IsiRouter123 release"1.0"
    STATUS              current
    DESCRIPTION         "InfoTech High Speed Router"
    SUPPORTS            snmpMIB
        INCLUDES        {systemGroup, snmpGroup, snmpSetGroup,
                        snmpBasicNotificationsGroup}

        VARIATION       coldStart
            DESCRIPTION "A coldStart trap is generated on all
                        reboots."

    SUPPORTS            IF-MIB
        INCLUDES        {ifGeneralGroup, ifPacketGroup}
    SUPPORTS            IP MIB
        INCLUDES        {ipGroup, icmpGroup}
    SUPPORTS            TCP-MIB
        INCLUDES        {tcpGroup}
    SUPPORTS            UDP-MIB
        INCLUDES        {udpGroup}
    SUPPORTS            EGP-MIB
        INCLUDES        {egpGroup}
    ::= { isiRouter 1 }
```

Figure 6.30 Example of an AGENT-CAPABILITIES Macro

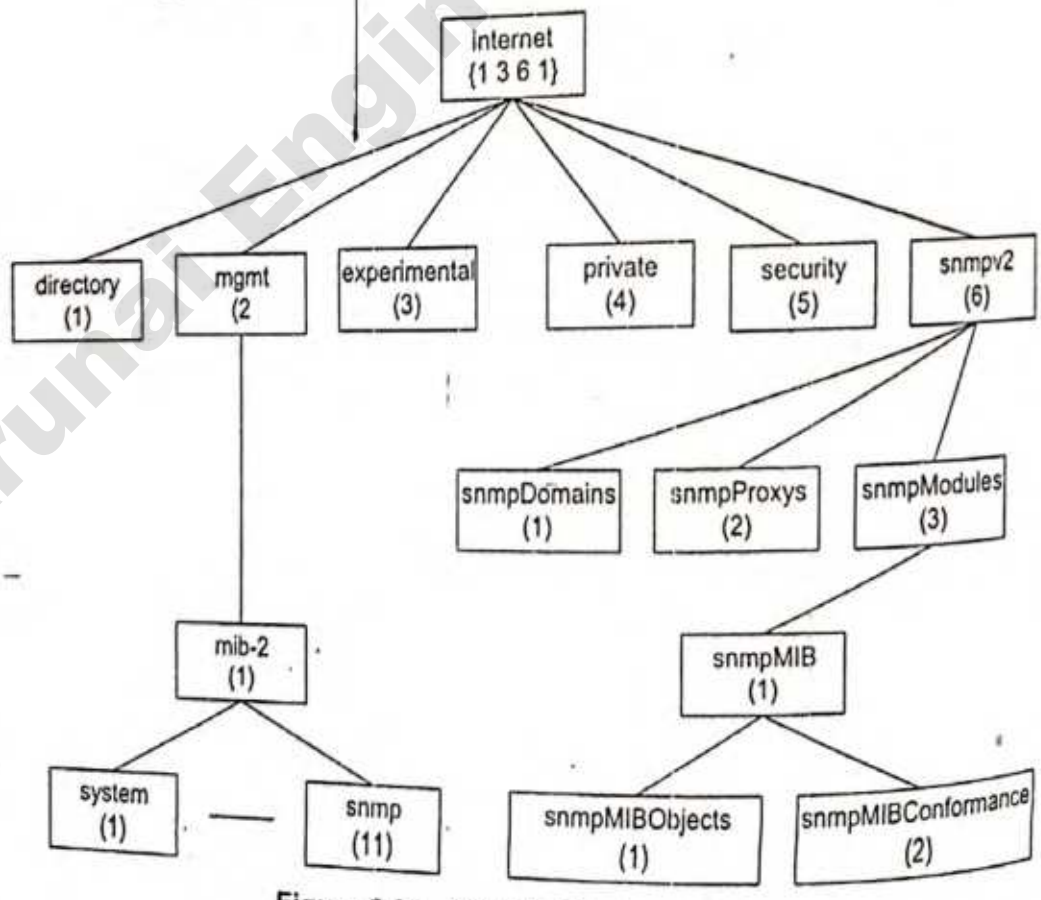


Figure 6.31 SNMPv2 Internet Group

mib-2 have significant changes as defined in RFC 1907. A new module *snmpMIB* has been defined which is {snmpModules 1}. There are two modules under *snmpMIB*: *snmpMIBObjects* and *snmpMIB Conformance*.

The MIB module *snmpMIBObjects* addresses the new objects introduced in SNMPv2, as well as those that are obsolete. This is primarily concerned with trap, which has been brought into the same format as other PDUs. Also, many of the unneeded objects in the SNMP group have been made obsolete.

We discussed conformance specifications and object groups in the previous section. These are specified under the *snmpMIBconformance* module. As SNMPv2 is currently defined, there is a strong coupling between system, *snmp*, *snmpMIBObjects*, and *snmpMIBconformance* modules. With this picture in mind, it will be a lot easier to follow RFC 1907, which discusses all these modules.

6.4.1 Changes to the System Group in SNMPv2

There are seven entities or objects in SNMPv2, which are common to a system. Additional information is added to the System group in SNMPv2, which contains a collection of objects that support various MIB modules. These are called object resources and are configurable both statically and dynamically. Figure 6.32 shows the MIB tree for the System group in SNMPv2. The *sysORLastChange* entity and *sysORTable* have been added to the set of objects in the System group. Table 6.6 presents the entity, OID, and a brief description of each entity for the System group.

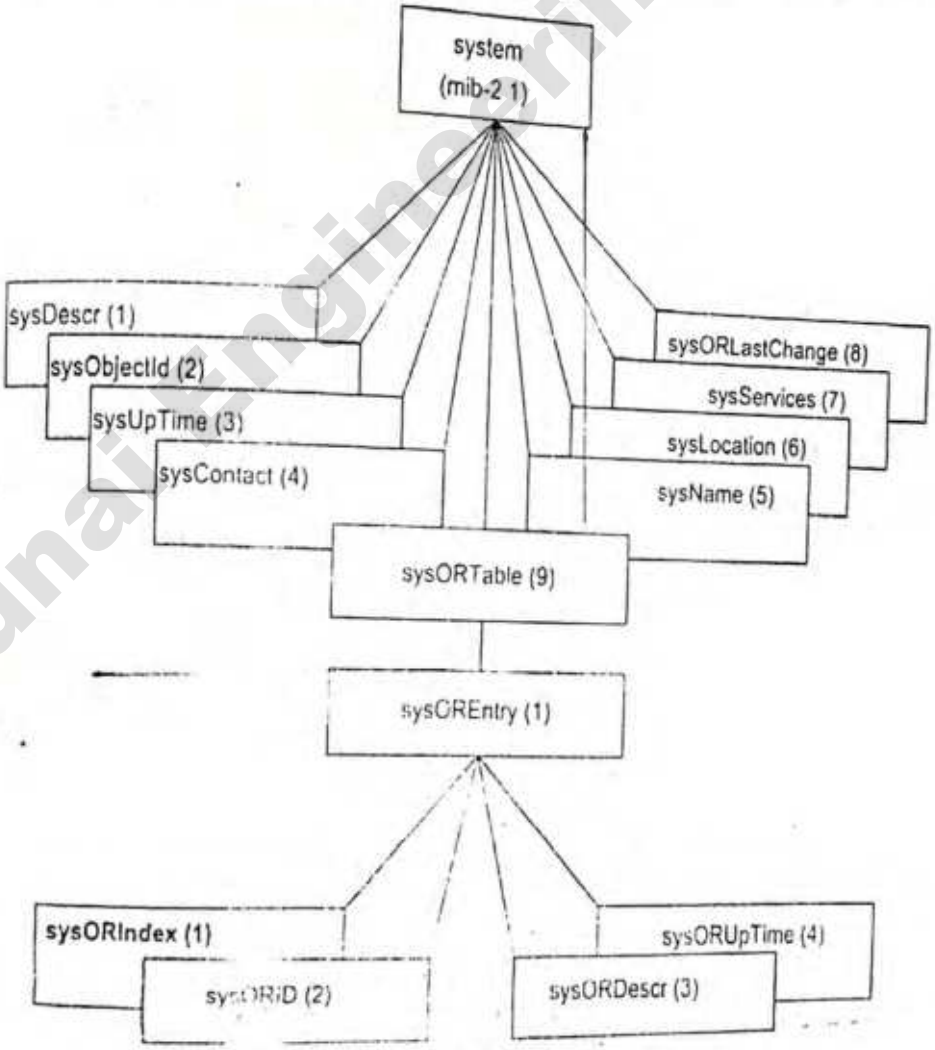


Figure 6.32 SNMPv2 System Group

Table 6.6 SNMPv2 System Group

ENTITY	OID	DESCRIPTION (BRIEF)
sysDescr	system 1	Textual description
sysObjectID	system 2	OBJECT IDENTIFIER of the entity
sysUpTime	system 3	Time (in hundredths of a second since last reset)
sysContact	system 4	Contact person for the node
sysName	system 5	Administrative name of the system
sysLocation	system 6	Physical location of the node
sysServices	system 7	Value designating the layer services provided by the entity
sysORLastChange	system 8	SysUpTime since last change in state or sysORID change
sysORTable	system 9	Table listing system resources that the agent controls; manager can configure these resources through the agent.
sysOREntry	sysORTable 1	An entry in the sysORTable
sysORIndex	sysOREntry 1	Row index, also index for the table
sysORID	sysOREntry 2	ID of the resource module
sysORDescr	sysOREntry 3	Textual description of the resource module
sysORUpTime	sysOREntry 4	System up-time since the object in this row was last instantiated

6.4.2 Changes to the SNMP Group in SNMPv2

The SNMP group in SNMPv2 has been considerably simplified from SNMPv1 by eliminating a large number of entities that were considered unnecessary. The simplified SNMP group is shown in Figure 6.33 (compare with Figure 5.21!). It has only eight entities, six old ones (1,3,4,5,6,30) and

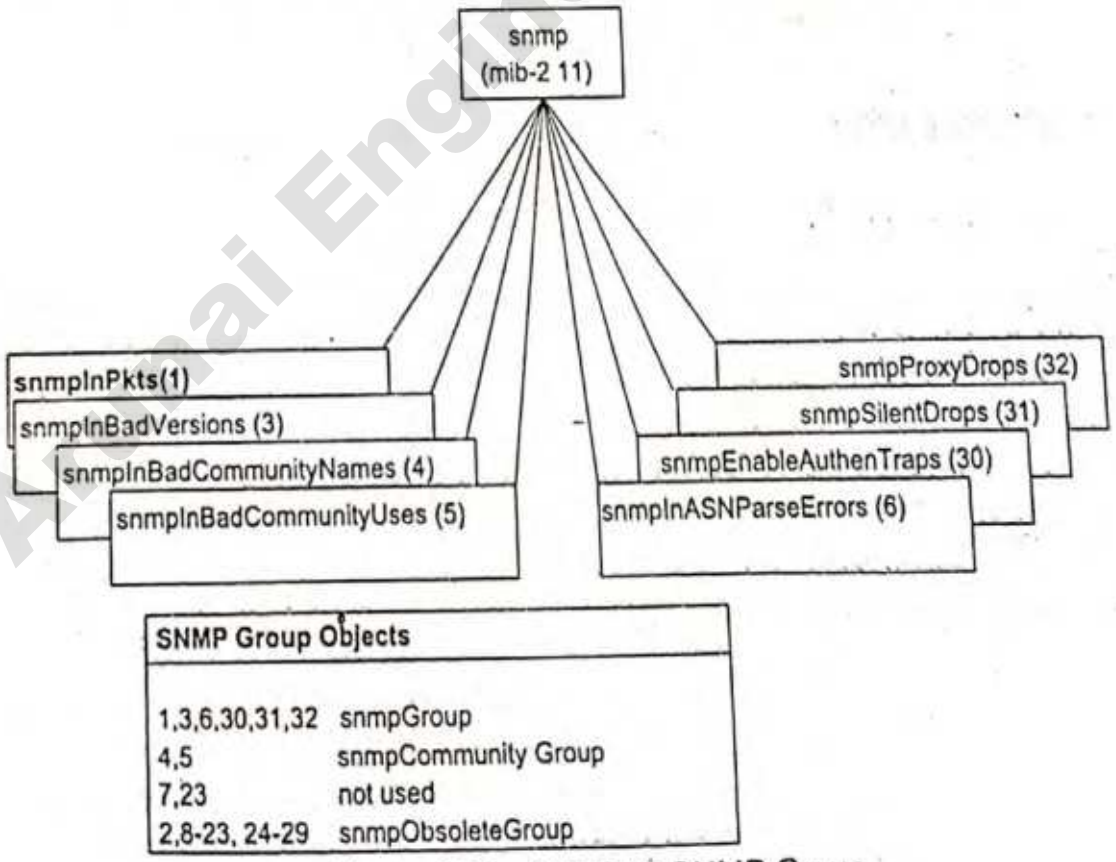


Figure 6.33 SNMPv2 SNMP Group

two new ones (31,32). Figure 6.33 also presents the four groups of all SNMP entities: *snmpGroup*, *snmpCommunityGroup*, *snmpObsoleteGroup*, and the group of two objects, 7 and 23, not used even in version 1. We will soon see that the *snmpGroup* is mandatory to implement for compliance of SNMPv2 and the *snmpCommunityGroup* is optional. The *snmpObsoleteGroup* is self-explanatory.

The SNMPv2 SNMP group table is shown in Table 6.7. All the unused and obsolete entities have been omitted for clarity.

6.4.3 Information for Notification in SNMPv2

Information on traps in SNMPv1 has been restructured in version 2 to conform to the rest of the PDUs. The macro TRAP-TYPE, used in version 1 and described in RFC 1215, has been made obsolete in SNMPv2. At the same time, enhancement to specifications has been made, and the terminology has been generalized to "notification," as the subheading indicates.

The information on notifications is defined under *snmpMIBObjects* and is shown in Figure 6.34. There are three modules under the *snmpMIBObjects* node: *snmpTrap* (4), *snmpTraps* (5), and *snmpSet* (6). The subnode designations 1, 2, and 3 under *snmpMIBObjects* have been made obsolete. A brief description of the subnodes and leaf objects under *snmpMIBObjects* is given in Table 6.8.

The *snmpTrap* group contains information on the OBJECT IDENTIFIERs of the trap and the enterprise responsible to send the trap. A new value, *accessible-for-notify*, has been added to the MAX-ACCESS clause to define objects under *snmpTrap*.

The entities under *snmpTraps* are the well-known traps that are currently in extensive use in SNMPv1. The *snmpSetSerialNo* is a single entity under *snmpSet* and is used by coordinating manager objects to

Table 6.7 SNMPv2 SNMP Group

ENTITY	OID	DESCRIPTION (BRIEF)
snmpInPkts	snmp (1)	Total number of messages delivered from transport service
snmpInBadVersions	snmp (3)	Total number of messages from transport service that are of unsupported version
snmpInBadCommunityNames	snmp (4)	Total number of messages from transport service that are of unknown community name
snmpInBadCommunityUses	snmp (5)	Total number of messages from transport service, of not allowed operation by the sending community
snmpInASNParseErrs	snmp (6)	Total number of ASN.1 and BER errors
snmpEnableAuthenTraps	snmp (30)	Override option to generate authentication failure traps
snmpSilentDrops	snmp (31)	Total number of the five types of received PDUs that were silently dropped due to exceptions in var-binds or max. message size
snmpProxyDrops	snmp (32)	Total number of the five types of received PDUs that were silently dropped due to inability to respond to a target proxy

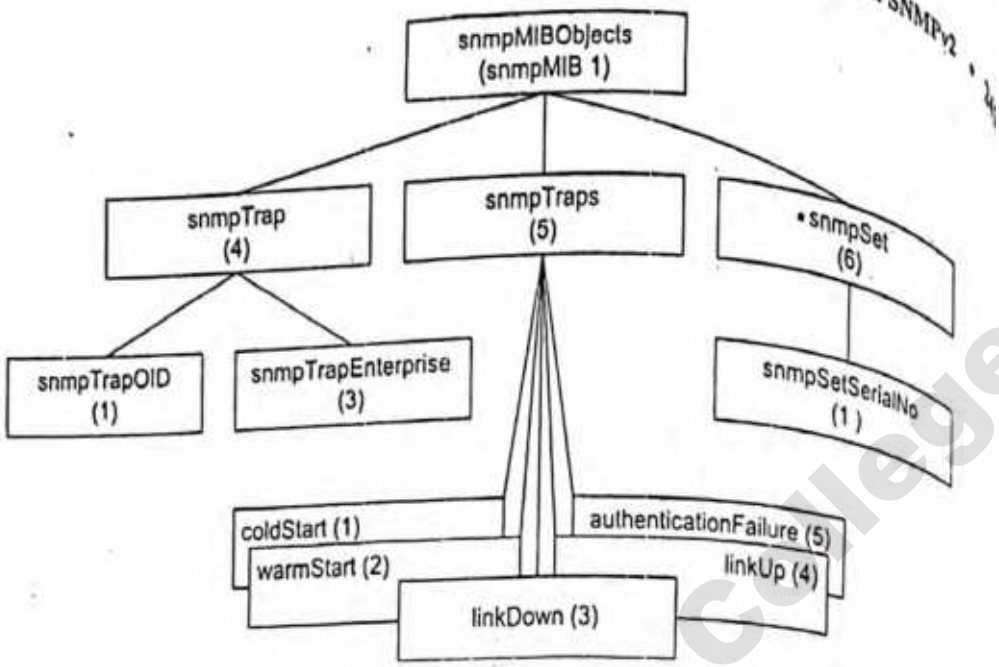


Figure 6.34 MIB Modules under snmpMIBObjects

Table 6.8 snmpMIBObjects MIB

ENTITY	OID	DESCRIPTION (BRIEF)
snmpTrap	snmpMIBObjects 4	Information group containing trap ID and enterprise ID
snmpTrapOID	snmpTrap 1	OBJECT IDENTIFIER of the notification
snmpTrapEnterprise	snmpTrap 2	OBJECT IDENTIFIER of the enterprise sending the notification
snmpTraps	snmpMIBObjects 5	Collection of well-known traps used in SNMPv1
coldStart	snmpTraps 1	Trap informing of a cold start of the object
warmStart	snmpTraps 2	Trap informing of a warm start of the object
linkDown	snmpTraps 3	Agent detecting a failure of a communication link
linkUp	snmpTraps 4	Agent detecting coming up of a communication link
authentificationFailure	snmpTraps 5	Agent reporting receipt of an unauthenticated protocol message
snmpSet	snmpMIBObjects 6	Manager-to-Manager notification messages
snmpSetSerialNo	snmpSet 1	Advisory lock between managers to coordinate set operation

perform the set operation. This is intended as coarse coordination only; fine-grain coordination may require more MIB objects in appropriate groups.

6.4.4 Conformance Information in SNMPv2

Conformance information is defined by the *snmpMIBConformance* module, as shown in Figure 6.35. It consists of two submodules, *snmpMIBcompliances* and *snmpMIBGroups*. The *snmpMIBCompliances* module has been extensively covered in Section 6.3.9. Units of conformance are defined in terms of OBJECT-GROUPS, mentioned in Section 6.3.9. Table 6.9 presents the various OBJECT-GROUPs defined in SNMPv2 and associated OBJECTS for all but *snmpObsoleteGroup*, which is shown in Figure 6.33.

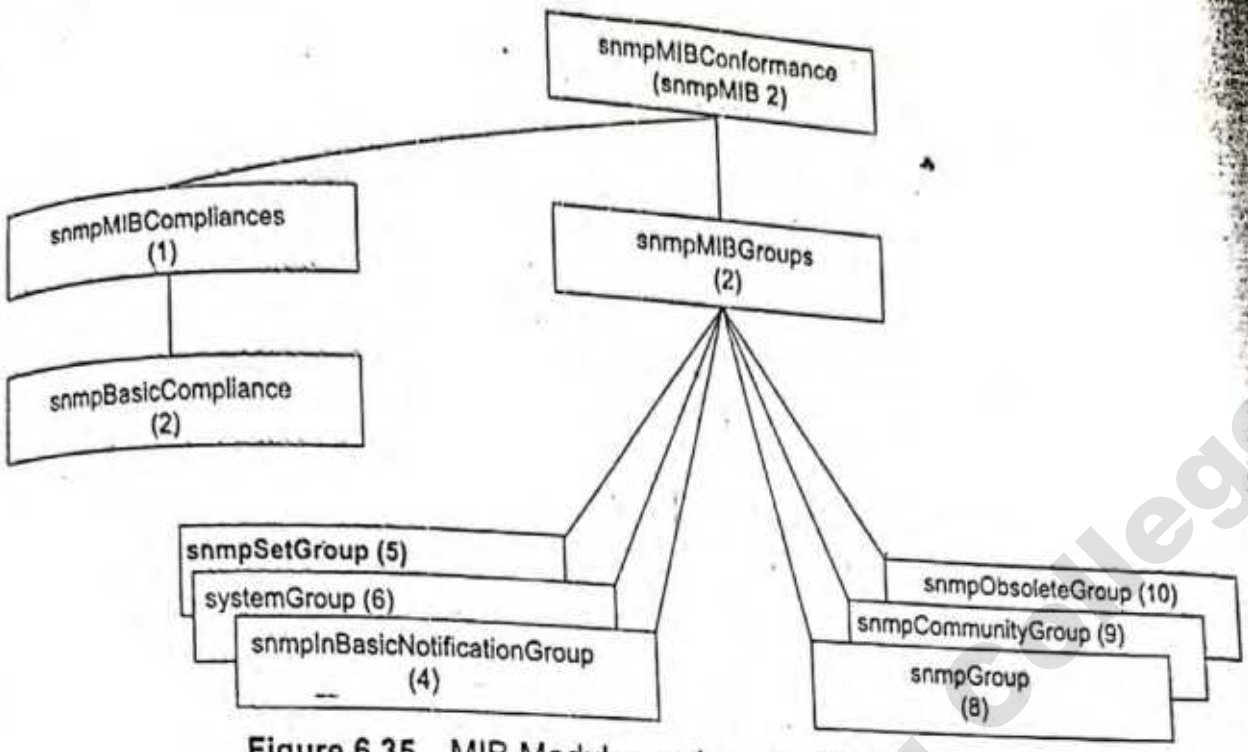


Figure 6.35 MIB Modules under snmpMIBConformance

6.4.5 Expanded Internet MIB-II

As SNMP network management expands covering legacy as well as new technology, MIB modules are continuously increasing. Figure 6.36 shows an expanded MIB-II when SNMPv2 was released and has more modules than those covered in RFC 1213. It is not intended to be an exhaustive list but includes RMON MIB module that we will be addressing in this textbook. Table 6.10 gives a description of each group in the MIB.

6.5 SNMPv2 PROTOCOL

SNMPv2 protocol operations are based on a community administrative model, which is the same as in SNMPv1. This was discussed in Section 5.2.2. We presented SNMPv2 protocol operations from a system architecture view in Section 6.2. In this section we will discuss details of PDU data structures and protocol operations.

6.5.1 Data Structure of SNMPv2 PDUs

The PDU data structure in SNMPv2 has been standardized to a common format for all messages. This improves the efficiency and performance of message exchange between systems. The significant improvement is bringing the trap data structure in the same format as the rest. The generic PDU message structure is shown in Figure 6.37 and is the same as Figure 5.8 of SNMPv1. The PDU type is indicated by an INTEGER. The error-status and error-index fields are either set to zero or ignored in the get-request, get-next-request, and set messages. The error-status is set to zero in the get-response message if there is no error; otherwise the type of error is indicated. The PDU and error-status are listed in Table 6.11. The error-index is set to zero if there is no error. If there is an error, it identifies the first variable binding in the variable-binding list that caused the error message. The first variable binding in a request's variable-binding list is index one, the second is index two, etc.

Table 6.9 SNMPv2 OBJECT-GROUPs

OBJECT-GROUPS	OID	OBJECTS
snmpSetGroup	snmpMIBGroups 5	snmpSetSerialNo
systemGroup	snmpMIBGroups 6	sysDescr
		sysObjectID
		sysUpTime
		sysContact
		sysName
		sysLocation
		sysServices
		sysORLastChange
		sysORID
		sysORUpTime
		sysORDescr
snmpBasicNotification Group	snmpMIBGroups 7	coldStart
		authenticationFailure
snmpGroup	snmpMIBGroups 8	snmpInPkts
		snmpInBadVersions
		snmpInASNParseErrs
		snmpSilentDrops
		snmpProxyDrops
		snmpEnableAuthenTraps
snmpCommunityGroup	snmpMIBGroups 9	snmpInBadCommunityNames
		snmpInBadCommunityUses
snmpObsoleteGroup	snmpMIBGroups 10	Please see Figure 6.33

There is a difference in usage of the error-status and error-index fields between SNMPv1 and SNMPv2. In version 1, any error encountered by the agent in responding to requests from the manager generates a non-zero value in either the error-status field or in both the error-status and error-index fields. Values in variable bindings are returned only under non-error conditions.

However, in SNMPv2, if only the error-status field of the Response-PDU is non-zero, the value fields of the variable binding in the variable-binding list are ignored. If both the error-status field and the error-index field of the Response-PDU are non-zero, then the value of the error-index field is the index of the variable binding (in the variable-binding list of the corresponding request) for which the request failed. Values in other variable bindings in the variable-binding list are returned with valid values and processed by the manager.

The generic PDU format is applicable to all SNMPv2 messages except the Get-Bulk-Request PDU, for which the format is shown in Figure 6.38. It can be seen that the format of the structure is the same in both cases, except that in the get-bulk-request message, the third and fourth fields are different. The third field, the error-status field, is replaced by non-repeaters; and the fourth field, the error-index

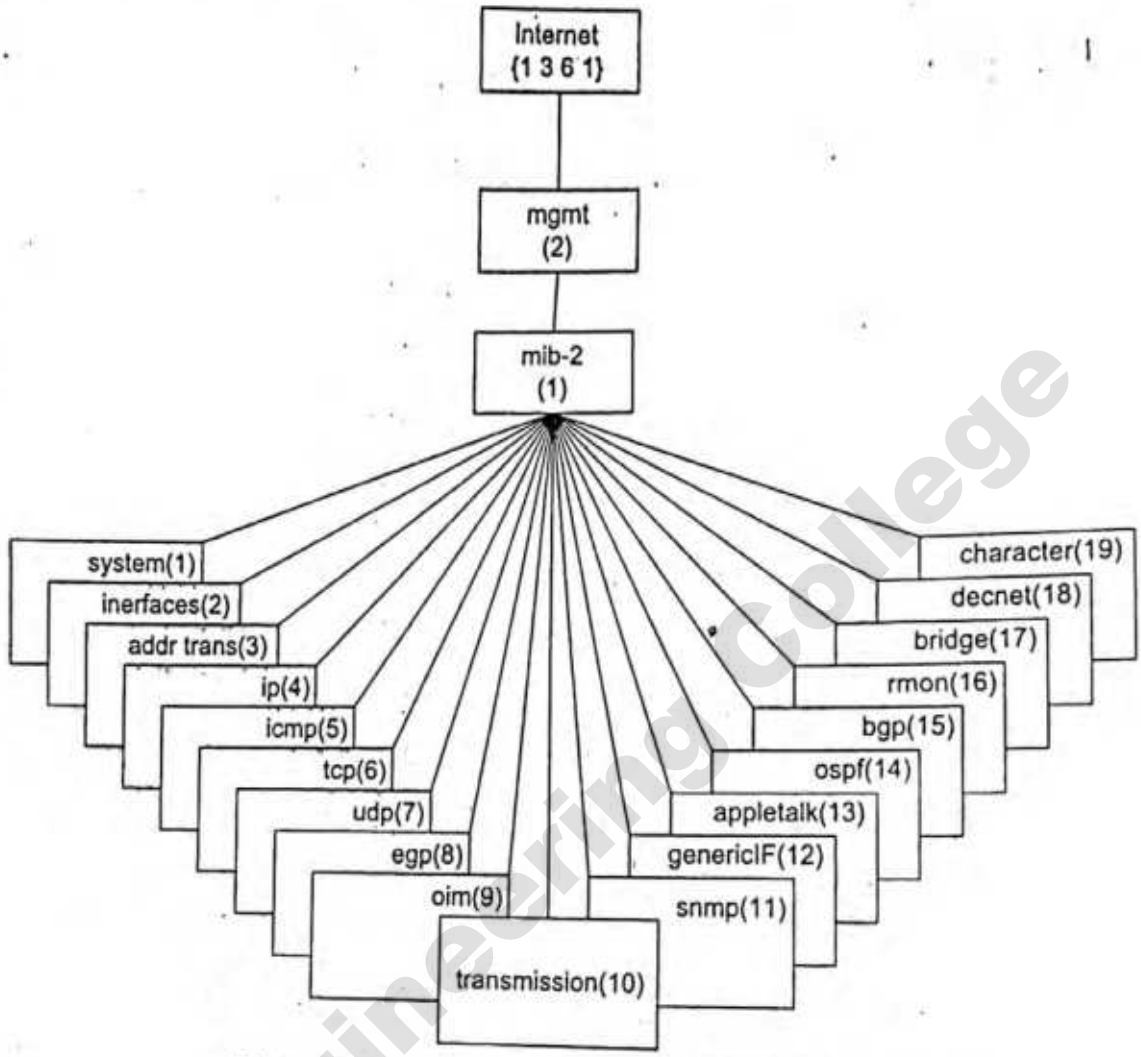


Figure 6.36 Expanded Internet MIB-II Group

Table 6.10 Expanded MIB-II Group

GROUP	OID	DESCRIPTION (BRIEF)
ifMIB	mib-2 31	Extension to interfaces group for new technologies
appletalk	mib-2 13	MIB for appletalk networks
ospf	mib-2 14	Open Shortest Path First routing protocol MIB
bgp	mib-2 15	MIB for Border Gateway Protocol for inter-autonomous network routing
rmon	mib-2 16	MIB for remote monitoring using RMON probe; there are MIBs under this for Ethernet and Token Ring networks
bridge	mib-2 17	MIB for bridges
decnet	mib-2 18	Digital Equipment Corporation DECnet MIB
character	mib-2 19	MIB for ports with character stream output for computer peripheral

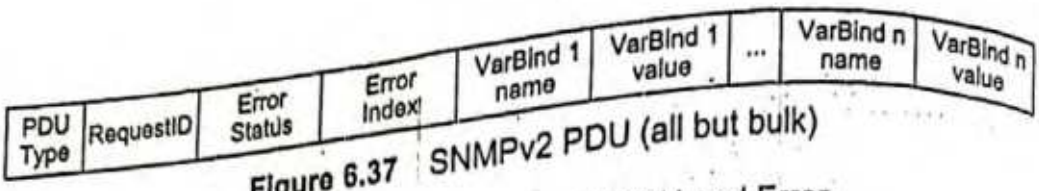


Figure 6.37 SNMPv2 PDU (all but bulk)

Table 6.11 Values for Types of PDU and Error-status Fields in SNMPv2 PDU

FIELD	TYPE	VALUE
PDU	0	Get-Request-PDU
	1	GetNextRequest-PDU
	2	Response-PDU
	3	Set-Request-PDU
	4	obsolete
	5	GetBulkRequest-PDU
	6	InformRequest-PDU
	7	SNMPv2-Trap-PDU
Error Status	0	noError
	1	tooBig
	2	noSuchName
	3	badValue
	4	readOnly
	5	genErr
	6	noAccess
	7	wrongType
	8	wrongLength
	9	wrongEncoding
	10	wrongValue
	11	noCreation
	12	inconsistentValue
	13	resourceUnavailable
	14	commitFailed
	15	undoFailed
	16	authorizationError
	17	notWritable
	18	inconsistentName

field, is replaced by max-repetitions. As we mentioned in Section 6.2, the get-bulk-request enables us to retrieve data in bulk. We can retrieve a number of both non-repetitive scalar values and repetitive tabular values with a single message. Non-repeaters indicate the number of non-repetitive field values

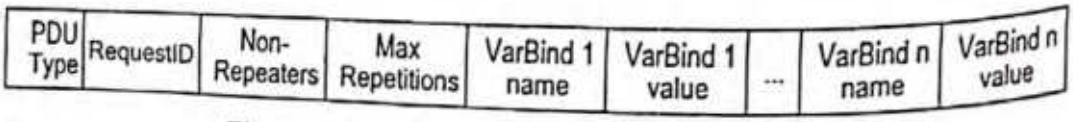


Figure 6.38 SNMPv2 GetBulkRequest PDU

requested; and the max-repetitions field designates the maximum number of table rows requested. We will next look at the SNMPv2 operations using PDUs.

6.5.2 SNMPv2 Protocol Operations

There are seven protocol operations in SNMPv2, as discussed in Section 6.2. We will ignore the report operation, which is not used. The messages, *get-request*, *get-next-request*, *set-request*, and *get-response*, are in both SNMPv1 and SNMPv2 versions and operate in a similar fashion. The two additional messages that are in SNMPv2, which are not in version 1, are the *GetBulkRequest* and *InformRequest*. The command, *get-bulk-request*, is an enhancement of *get-next request* and retrieves data in bulk efficiently. This is covered in the next subsection. The *InformRequest* is covered in a subsequent section along with SNMPv2-Trap, which has been modified in version 2.

GetBulkRequest PDU Operation. The *get-bulk-request* operation is added in SNMPv2 to retrieve bulk data from a remote entity. Its greatest benefit is in retrieving multiple rows of data from a table. The basic operation of get-bulk-request is the same as get-next-request. The third and fourth field positions are used in get-bulk-request message PDU as *non-repeaters* and *max-repetitions*, as shown in Figure 6.38. The non-repeaters field indicates the number of non-repetitive (scalar) objects to be retrieved. The max-repetitions field defines the maximum number of instances to be returned in the response message. This would correspond to the number of rows in an aggregate object. The value for the max-repetitions field is operation-dependent and is determined by such factors as the maximum size of the SNMP message, or the buffer size in implementation, or the expected size of the aggregate object table.

The data structure of the response for the get-bulk-request operation differs from other get and set operations. Successful processing of the get-bulk-request produces variable bindings (larger array of VarBind) in the response PDU, which is larger than that contained in the corresponding request. Thus, there is no one-to-one relationship between the VarBindList of the request and response messages.

Figure 6.39 shows a conceptual MIB to illustrate the operation of get-next-request and get-bulk-request shown in Figure 6.40 and Figure 6.41. It is similar to Figure 5.12 with two additional rows added to the table. To notice the difference in improvement of get-bulk-request over get-next-request, we will look at Figure 6.40, which shows the sequence of operations for get-next-request for the MIB shown in Figure 6.39. The sequence starts with a get-request message from the manager process with a VarBindList array of two scalar variables A and B. It is subsequently followed by the get-next-request message with three columnar OBJECT IDENTIFIERS T.E.1, T.E.2, and T.E.3. The get-response returns the first instance values T.E.1.1, T.E.2.1, and T.E.3.1. The sequence of operation continues until the fourth instance is retrieved. The last get-next-request message with the OBJECT IDENTIFIERS T.E.1.4, T.E.2.4, and T.E.3.4 generates the values T.E.2.1, T.E.3.1, and Z. This is because there are no more instances of the table. It retrieves the three objects, which are logically the next lexicographically higher objects—namely, T.E.2.1 (next to T.E.1.4), T.E.3.1 (next to T.E.2.4), and Z (next to T.E.3.4). The manager would stop the sequence at this message. However, if it continues the operation, it would receive a *noSuchName* error message.

Figure 6.41 shows the sequence of operations to retrieve the MIB shown in Figure 6.39 using the get-bulk message. The entire MIB data are retrieved in two requests. The first message GetBulkRequest (2, 3, A, B, T.E.1, T.E.2, T.E.3) is a request for receiving two non-repetitive objects (the first variable (2) in the request command) and three repetitive instances (the second operand (3) in the command) of the columnar objects (T.E.1, T.E.2, and T.E.3). The response returns values of A and B for the non-repetitive

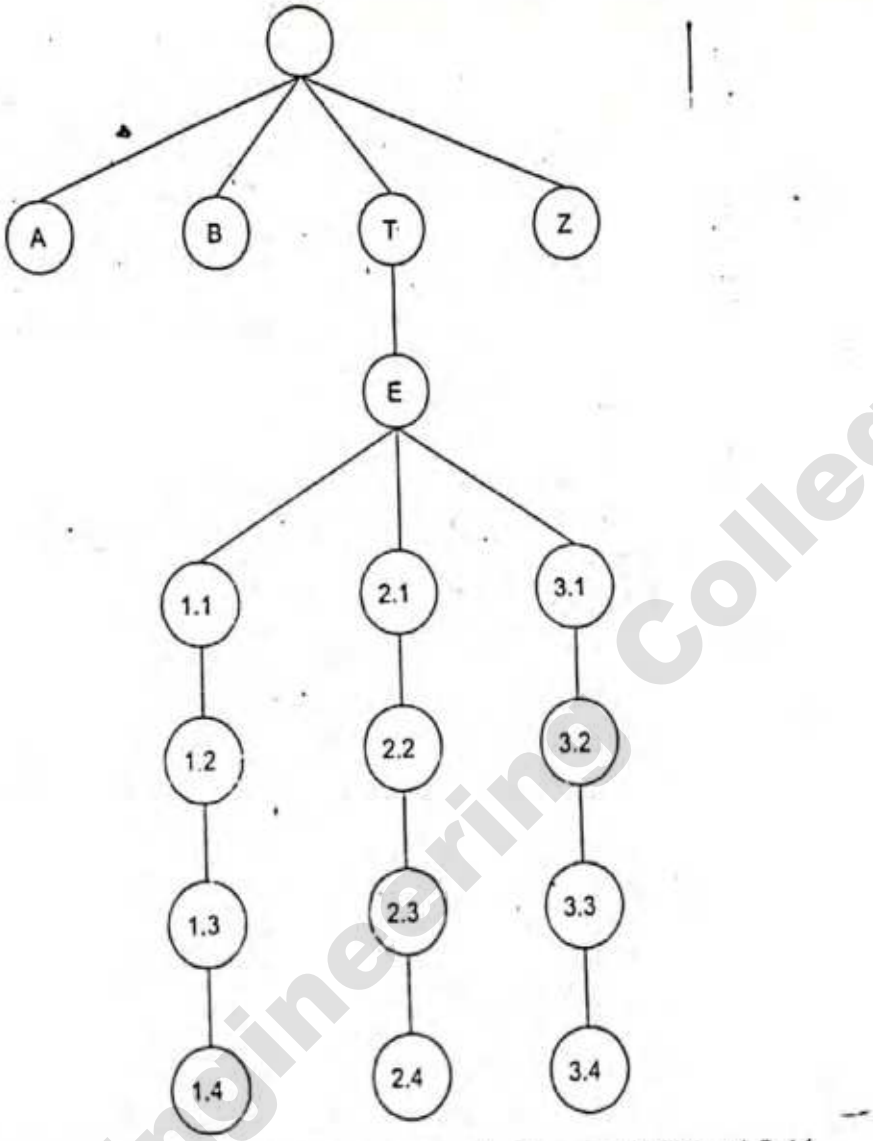


Figure 6.39 MIB for Operation Sequences in Figures 6.40 and 6.41

objects, and the first three rows of the aggregate object table. The second request is for three more rows of the table. Since there is only one more row left to send, the response message contains the information in the last row, the next lexicographic entity, Z, and the error message *endOfMibView*. The manager interprets this as end of the table.

Figure 6.42 shows the retrieval of the Address Translation table shown in Figure 5.16 using the get-bulk-request operation. Instead of four sets of get-next-request and get-response messages, only two get-bulk-request and response messages are needed in the get-bulk-request operation.

SNMPv2-Trap and InformRequest PDU Operations. The SNMPv2-Trap PDU performs the same function as in version 1. As we notice, the name has been changed, as well as its data structure to the generic format shown in Figure 6.37. The variable bindings in positions 1 and 2 are specified as *sysUpTime* and *snmpTrapOID*, as shown in Figure 6.43. The destination(s) to which a trap is sent is implementation-dependent.

A trap is defined by using a NOTIFICATION-TYPE macro. If the macro contains an OBJECTS clause, then the objects defined by the clause are in the variable bindings in the order defined in the

7.1 SNMPv3 KEY FEATURES

One of the key features of SNMPv3 is the modularization of architecture and documentation. The design of the architecture integrated SNMPv1 and SNMPv2 specifications with the newly proposed SNMPv3. This enables continued usage of legacy SNMP entities along with SNMPv3 agents and manager. That is good news as there are tens of thousands of SNMPv1 and SNMPv2 agents in the field.

An SNMP engine is defined with explicit subsystems that include dispatch and message-processing functions. It manages all three versions of SNMP to coexist in a management entity. Application services and primitives have been explicitly defined in SNMPv3. This formalizes the various messages that have been in use in the earlier versions.

Another key feature is the improved security feature. The configuration can be set remotely with secured communication that protects against modification of information and masquerade by using encryption schemes. It also tries to ensure against malicious modification of messages by reordering and time delaying of message streams, as well as protects against eavesdropping of messages.

The access policy used in SNMPv1 and SNMPv2 is continued and formalized in the access control in SNMPv3, designated VACM. The SNMP engine defined in the architecture checks whether a specific type of access (read, write, create, notify) to a particular object (instance) is allowed.

7.2 SNMPv3 DOCUMENTATION ARCHITECTURE

The numerous SNMP documents have been organized by IETF to follow a document architecture. The SNMP document architecture addresses how existing documents and new documents could be designed to be autonomous and, at the same time, be integrated to describe the different SNMP frameworks. The representation shown in Figure 7.1 reflects the contents of the specifications, but it is another perspective of what is given in RFC [3410]. It can be correlated with what we presented in Figure 4.4. Two sets of documents are of general nature. One of them is the set of documents on roadmap, applicability statement, and coexistence and transition.

The other set of documents, SNMP frameworks, comprises the three versions of SNMP. An SNMP framework represents the integration of a set of subsystems and models. A model describes a specific design of a subsystem. The implementation of an SNMP entity is based on a specific model based on a specific framework. For example, a message in an SNMP manager is processed using a specific message processing mode (we will discuss these later) in a specific SNMP3 framework. The SNMP frameworks document set is not explicitly shown in the pictorial presentation in RFC [2271], as we have done here. RFC [1901] in SNMPv2 and RFC [2271] in SNMPv3 are SNMP framework documents.

The information model and MIBs cover Structure of Management Information (SMI), textual conventions, and conformance statements, as well as various MIBs. These are covered in STD 16 and STD 17 documents along with SMIv2 documents [RFC 2578–2580].

Message Handling and PDU Handling sets of documents address transport mappings, message processing and dispatching, protocol operations, applications, and access control. These would correspond to the SNMP STD 15 documents and the draft documents on SNMPv2 [RFC 1905–1907] shown in Figure 4.9. RFCs [2573–2575] address these in SNMPv3.

7.3 ARCHITECTURE

An SNMP management network consists of several nodes, each with an SNMP entity. They interac with each other to monitor and manage the network and resources. The architecture of an SNMP enti

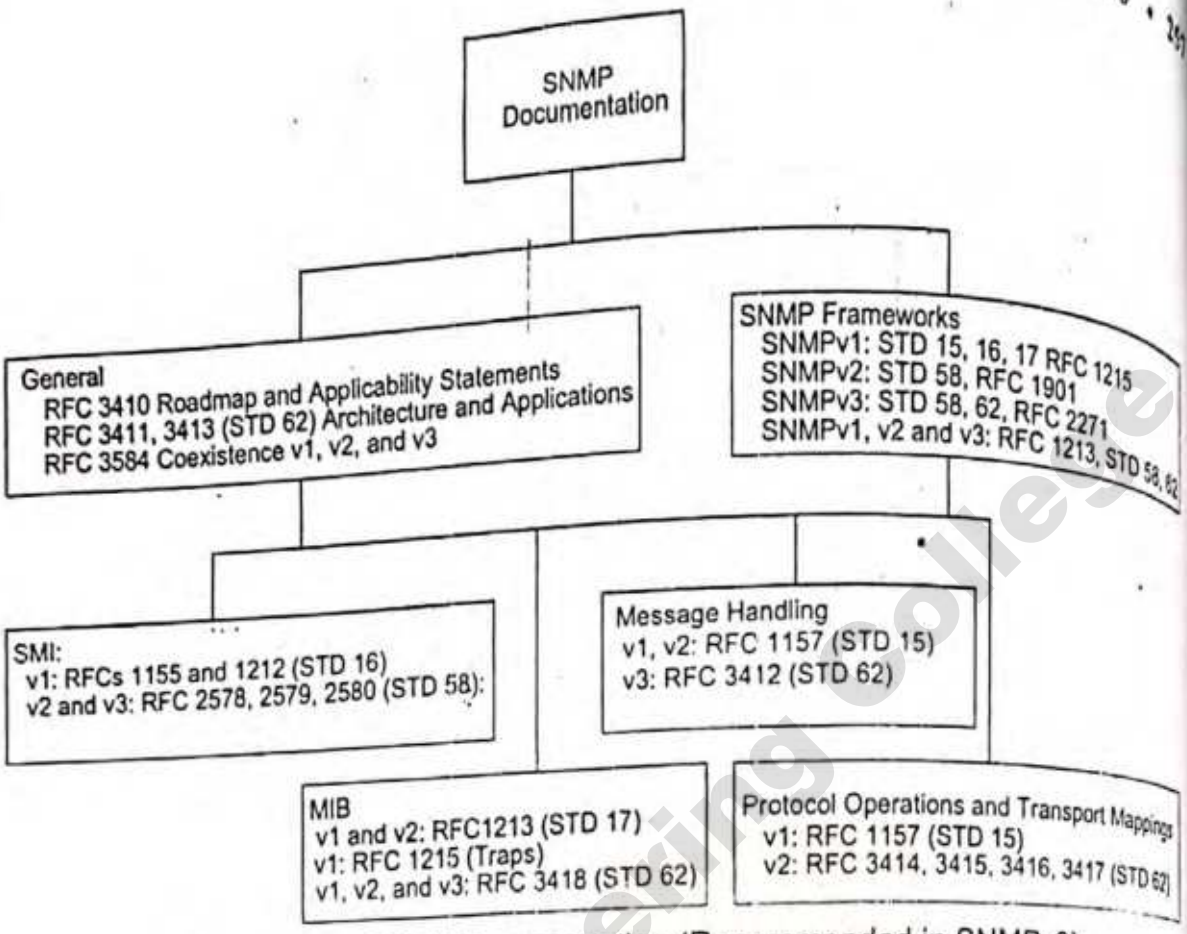


Figure 7.1 SNMP Documentation (Recommended in SNMPv3)

is defined as the elements of an entity and the names associated with them. There are three kinds of naming: naming of entities, naming of identities, and naming of management information. Let us first look at the elements of an entity, including its naming.

7.3.1 Elements of an Entity

The elements of the architecture associated with an SNMP entity, shown in Figure 7.2, comprise an SNMP engine and a set of applications. The SNMP engine, named *snmpEngineID*, comprises a dispatcher, message processing subsystem, security subsystem, and an access control subsystem.

SNMP Engine. As shown in Figure 7.2, an SNMP entity has one SNMP engine, which is uniquely identified by an *snmpEngineID*. The SNMP engine ID is made up of octet strings. The length of the ID is 12 octets for SNMPv1 and SNMPv2, and is variable for SNMPv3. This is shown in Figure 7.3. The first four octets in both formats are set to the binary equivalent of the agent's SNMP management private enterprise number. The first bit of the four octets is set to 1 for SNMPv3 and 0 for earlier versions. For example, if Acme Networks has been assigned {enterprises 696}, the first four octets would read '800002b8'H in SNMPv3 and '000002b8'H in SNMPv1 and SNMPv2.

The fifth octets for SNMPv1 and SNMPv2 indicate the method that the enterprise used for deriving the SNMP engine ID and 6–12 octets function of the method. For a simple entity, it could be just the IP address of the entity.

The fifth octet of the SNMPv3 engine ID indicates the format used in the rest of the variable number of octets. Table 7.2 shows the values of the fifth octet for SNMPv3.

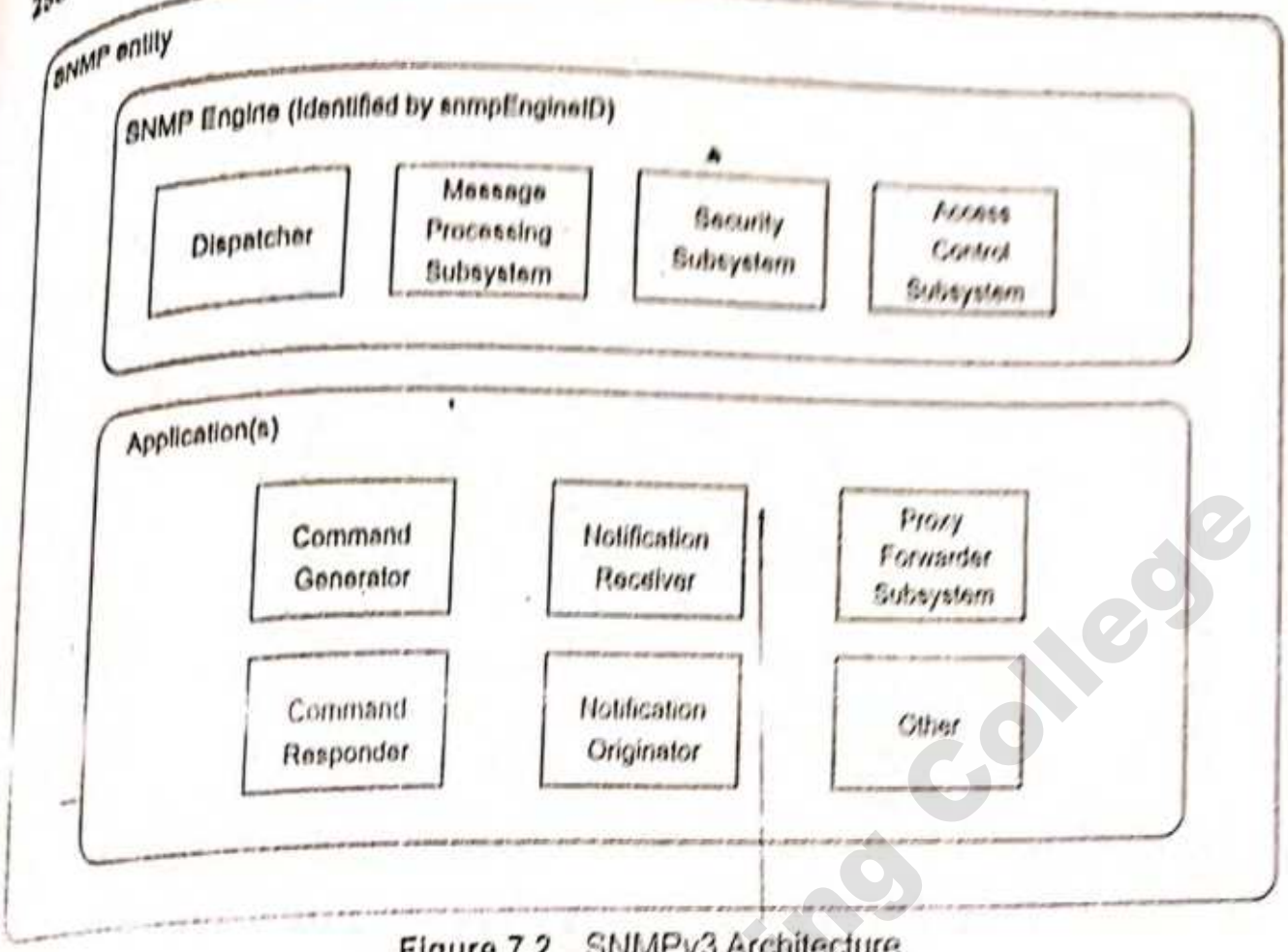


Figure 7.2 SNMPv3 Architecture

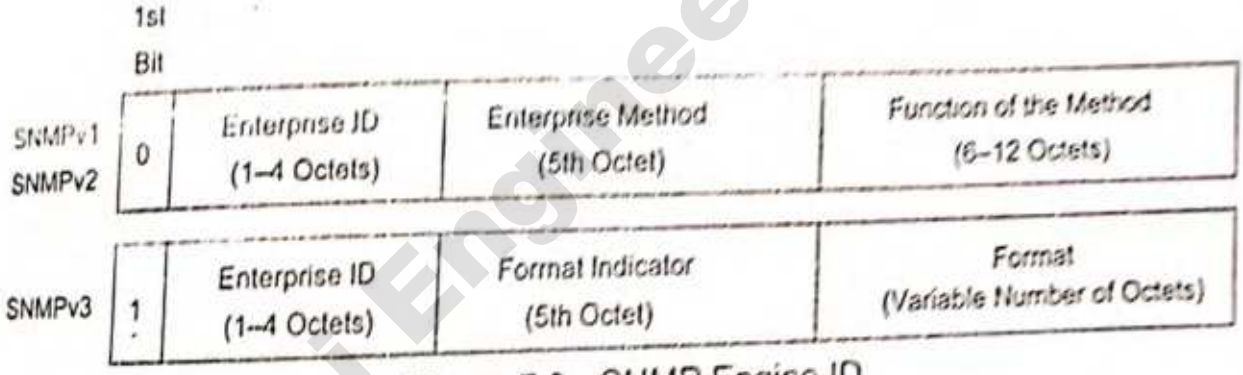


Figure 7.3 SNMP Engine ID

Dispatch Subsystem. There is only one dispatcher in an SNMP engine and it can handle multiple versions of SNMP messages. It does the following three sets of functions. First, it sends messages to and receives messages from the network. Second, it determines the version of the message and interacts with the corresponding MPM. Third, it provides an abstract interface (described in Section 7.3.3) to SNMP applications to deliver an incoming PDU to the local application and to send a PDU from the local application to a remote entity.

The three separate functions in the dispatcher subsystem are accomplished using: (1) a transport mapper; (2) a message dispatcher; and (3) a PDU dispatcher. The transport mapper delivers the message over the appropriate transport protocol of the network. The message dispatcher routes the outgoing and incoming messages to the appropriate module of the message processor. If a message is received for an SNMP version, which is not handled by the message processing subsystem, it would be rejected by the

Table 7.2 SNMPv3 Engine ID Format (5th Octet)

0	Reserved, unused
1	IPv4 address (4 octets)
2	IPv6 (16 octets) Lowest non-special IP address
3	MAC address (6 octets) Lowest IEEE MAC address, canonical order
4	Text, administratively assigned Maximum remaining length 27
5	Octets, administratively assigned Maximum remaining length 27
6–127	Reserved, unused
128–255	As defined by the enterprises Maximum remaining length 27

message dispatcher. The PDU dispatcher within an SNMP entity handles the traffic routing of PDUs between applications and the Message Processor Model.

Message Processing Subsystem. The SNMP message processing subsystem of an SNMP engine interacts with the dispatcher to handle version-specific SNMP messages. It contains one or more MPMs. The version is identified by the version field in the header.

Security and Access Control Subsystems. The security subsystem provides security services at the message level in terms of authentication and privacy protection. The access control subsystem provides access authorization service.

Applications Module. The application(s) module is made up of one or more applications, which comprise command generator, notification receiver, proxy forwarder, command responder, and notification originator. The first three applications are normally associated with an SNMP manager and the last two with an SNMP agent. The application(s) module may also include other applications. as indicated by the box, "Other," in Figure 7.2.

7.3.2 Names

Naming of entities, identities, and management information is part of SNMPv3 specifications. We already mentioned the naming of an entity by its SNMP engine ID, *snmpEngineID*. Two names are associated with identities, *principal* and *securityName*. *Principal* is the "who" requesting services. It could be a person or an application. The *securityName* is a human readable string representing a principal. The principal could be a single user; for example, name a network manager or a group of users, such as names of operators in the network operations center. It is made non-accessible. It is hidden and is based on the security model (SM) used. However, it is administratively given a security name: for example, User 1 or Admin, which is made readable by all.

A management entity can be responsible for more than one managed object. For example, a management agent associated with a managed object at a given node could be managing a neighboring node besides its own. Each object is termed *context* and has a *contextEngineID* and a *contextName*. When there is a one-to-one relationship between the management entity and the managed object, *contextEngineID* is the same as *snmpEngineID*. A *scopedPDU* is a block of data containing a *contextEngineID*, a *contextName*, and a PDU. An example of this would be a switched hub where a common SNMP agent in the hub is accessed to manage the interfaces of the hub. The agent would have an *snmpEngineID* and each interface card would have a context Engine ID. In contrast, in a non-switched hub with each interface card being managed individually, the snmpEngineID and contextID are the same.

7.3.3 Abstract Service Interfaces

The subsystems in an SNMP entity communicate with each other across an interface, a subsystem providing a service and the other using the service. We can define a service interface between the two. If the interface is defined such that it is generic and independent of specific implementation, it becomes a conceptual interface, termed *abstract service interface*. These abstract services are defined by a set of primitives that define the services. Figure 7.4(a) shows subsystem A sending a request for service using the primitive *primitiveAB* to subsystem B. The *primitiveAB* is associated with the receiving subsystem B, which is the one that is providing the service in this illustration. A primitive has IN and OUT as operands or parameters, which are data values. These are indicated by a1 and a2, and b1 and b2, respectively. The IN parameters are input values to the called subsystem from the subsystem calling for service. The OUT parameters are the responses expected from the called subsystem to the calling subsystem. The OUT parameters are sent unfilled in the message format by the calling system (remember Get-Request PDU?) and are returned filled (Get-Response) by the called subsystem. When the calling subsystem expects a response from the called subsystem, there are directed messages in both directions with a two-directional arrow coupling the two, as shown in Figure 7.4(a). In this case the primitive *primitiveAB* is only indicated in the forward direction. In addition to returning the OUT parameters, the called subsystem could also return a value associated with the result of the request in terms of *statusInformation* or result, as shown in Figure 7.4(a). Because of the execution of *primitiveAB*, subsystem B may initiate a request for service from another subsystem, subsystem C, using *primitiveBC* over the abstract service interface between subsystems B and C.

In general, except for dispatcher, primitives are associated with the receiving subsystem. Dispatcher primitives are used in receiving messages from and to the application modules, as well as registering and unregistering them, and in transmitting and receiving messages from the network.

Figure 7.4(b) shows the example of the application, command generator, sending a request *sendPdu* (destined for a remote entity) to the dispatcher. The dispatcher, after successful execution of the service requested and sending it on the network, returns to the application *sendPduHandle*. The *sendPduHandle* will be used by the command generator to correlate the response from the remote entity. There are no OUT parameters to be filled in this primitive except the status information. However, the command generator does expect the status information, hence the coupling arrow indicator in the figure. The dispatcher sends an error indicator instead of *sendPduHandle* for the status information if the *sendPdu* transaction is a failure. The dispatcher also generates a request to the MPM, *prepareOutgoingMessage*. The *prepareOutgoingMessage* has both IN and OUT parameters and hence information flows in both directions. The numerous IN and OUT parameters associated with primitives are not identified in the figure for the sake of simplicity.

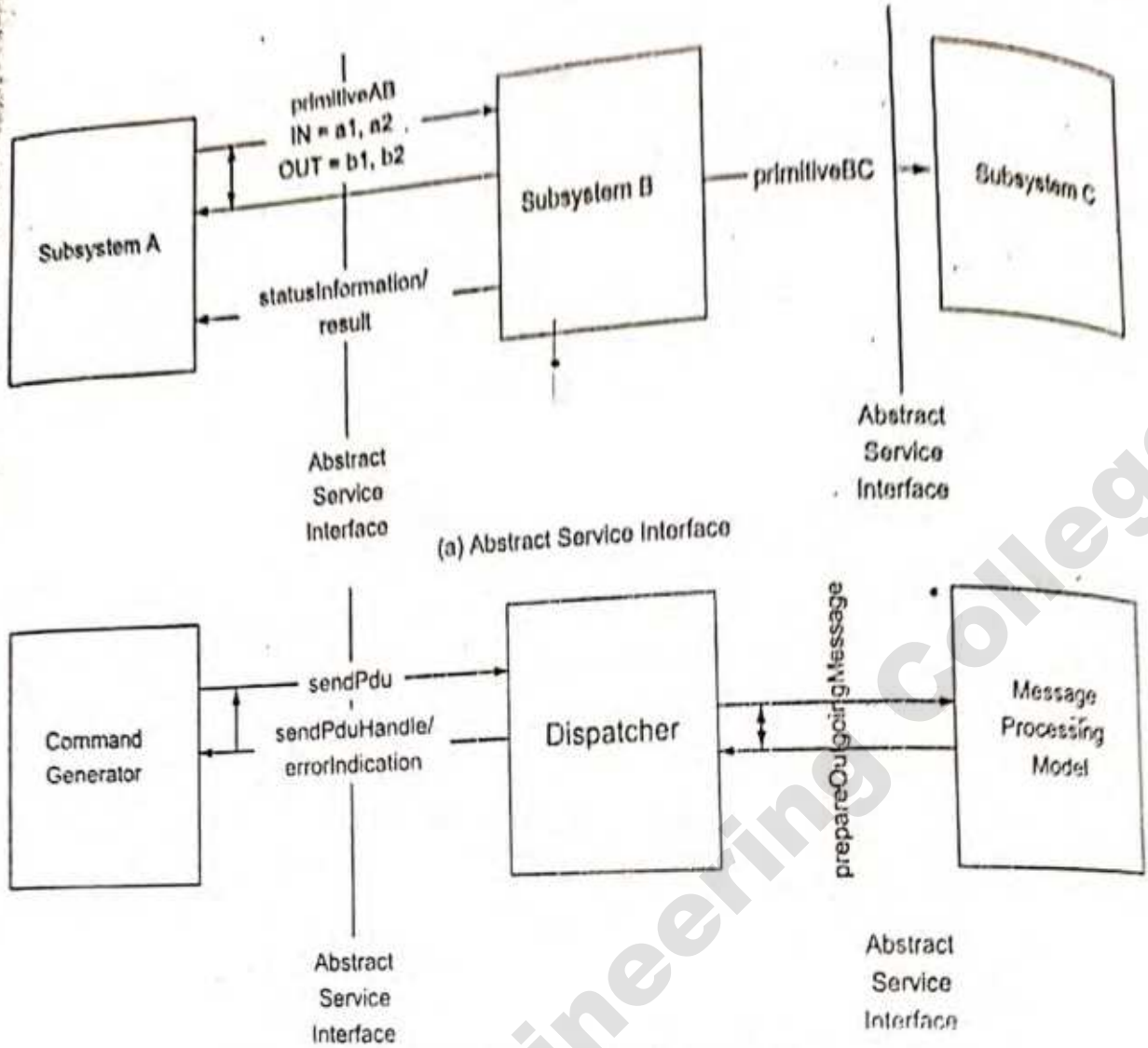


Figure 7.4 Abstract Service Interfaces

Table 7.3 lists the primitives served by the dispatcher, message processing subsystem, security, and access control subsystems. A brief description is presented for each primitive on the service provided and the user of service.

7.4 SNMPv3 APPLICATIONS

SNMPv3 formally defines five types of applications. These are not the same as the functional model that the OSI model addresses. These may be considered as the application service elements that are used to build applications. They are command generator, command responder, notification originator, notification receiver, and proxy forwarder. These are described in RFC 2273.

7.4.1 Command Generator

A command generator application is used to generate get-request, get-next-request, get-bulk, and set-request messages. The command generator also processes the response received for the command sent. Typically, the command generator application is associated with the network manager process.

Table 7.3 List of Primitives

MODULE	PRIMITIVE	SERVICE PROVIDED
Dispatcher	sendPdu	Processes request from application to send a PDU to a remote entity
Dispatcher	processPdu	Processes incoming message from a remote entity
Dispatcher	returnResponsePdu	Processes request from application to send a response PDU
Dispatcher	processResponsePdu	Processes incoming response from a remote entity
Dispatcher	registerContextEngineID	Registers request from a context engine
Dispatcher	unregisterContextEngineID	Unregisters request from a context engine
Message Processing Model	prepareOutgoingMessage	Processes request from dispatcher to prepare outgoing message to a remote entity
Message Processing Model	prepareResponseMessage	Processes request from dispatcher to prepare outgoing response to a remote entity
Message Processing Model	prepareDataElements	Processes request from dispatcher to extract data elements from an incoming message from a remote entity
Security Model	generateRequestMsg	Processes request from message processing model to generate a request message
Security Model	processIncomingMsg	Processes request from message processing model to process security data in an incoming message
Security Model	generateResponseMsg	Processes request from message processing model to generate a response message
Intra-Security Model	authenticateOutgoingMsg	Processes request to authentication service to authenticate outgoing message
Intra-Security Model	authenticateIncomingMsg	Processes request for authentication service to incoming message
Intra-Security Model	encryptData	Processes request from security model to privacy service to encrypt data
Intra-Security Model	decrypt Data	Processes request for privacy service to decrypt an incoming message
Access Control Model	isAccessAllowed	Processes request from application to access and authorize service requested

Figure 7.5 shows the use of the command generator application using the get-request example. In the top half of the figure, the get-request message is sent after it passes through the dispatcher, the MPM, and the SM. The command generator sends the *sendPdu* primitive to the dispatcher, which requests the

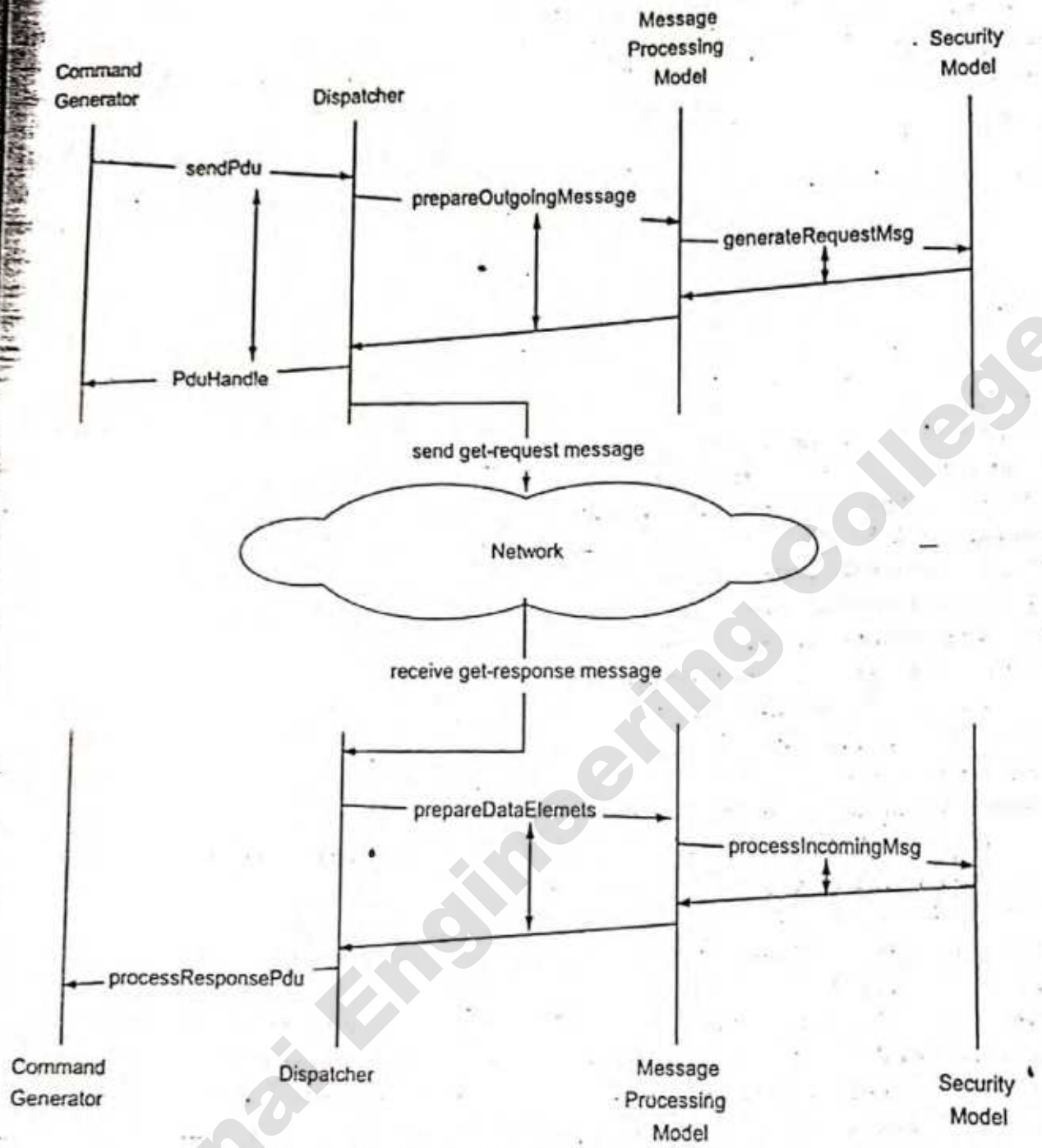


Figure 7.5 Command Generator Application

MPM to prepare an outgoing message. The dispatcher also sends a *sendPduHandle* to the command generator to track the request. The details on the information exchanged between MPM and SM are covered in Section 7.6. The SM is used to generate the outgoing message, including authentication and privacy parameters. The dispatcher then sends the message on the network.

The bottom half of Figure 7.5 presents the role of the command generator when the get-response message is received from the remote entity. The dispatcher receives the message from the network and requests MPM to prepare data elements, which are addressed in Section 7.6. The SM validates the authenticity and privacy parameters. The dispatcher receives the returned message from SM and forwards it to the command generator to process the response.

An example of the command generator transaction is an SNMPv1 get-request command from a network management system (NMS) to an agent requesting the values of System group (see Figure 5.17a). The command generator sends the command and OIDs to the dispatcher along with version number (SNMPv1) and security information. It also sends a tracking ID, *sendPduHandle*, to the NMS. This would be *Request ID* (=1) shown in Figure 5.17(a). When the MPM returns the outgoing message, which could be a secured (authenticated and encrypted) message, the dispatcher delivers it to the network using user datagram protocol (UDP) to be transmitted to the agent. The command generator receives the response from the dispatcher (asynchronously) sent by the agent. The primitive *processResponsePdu* would deliver the PDU containing the values for the System group shown in Figure 5.17(b) to the command generator. The command generator matches the response PDU received with *Request ID* = 1 with the one that was sent.

7.4.2 Command Responder

A command responder processes the get and set requests destined for it and is received from a legitimate non-authoritative remote entity. It performs the appropriate action of get or set on the network element, prepares a get-response message, and sends it to the remote entity that made the request. This is shown in Figure 7.6. In contrast to Figure 7.5, in which the top and bottom half processes run on two remote objects, the top and bottom of Figure 7.6 belong to the same object. Typically, the command responder is in the management agent associated with the managed object.

Before the get-request could be processed by the command-responder application, the context that the SNMP engine is responsible for must register with the SNMP engine. It does this by using the *registerContextEngineID*. Once this is in place, the get-request (same example as used in command generator) is received by the dispatcher, data elements are prepared by the MPM, security parameters are validated by the SM, and the *processPdu* is passed on to the command responder. This set of processes is presented in the top half of Figure 7.6.

Once the request is processed by the Command Responder, it prepares the get-response message as shown in the bottom half of Figure 7.6. The message is passed to the Dispatcher using the *returnResponsePdu*. The MPM prepares the response message, the SM performs the security functions, and the Dispatcher eventually transmits the get-response message on the network.

Continuing the example discussed in Section 7.4.1 for the command generator, the dispatcher in the SNMP agent receives the message. The message is processed by the MPM and the SM and is returned to the dispatcher. Assuming that the managed object has registered its context engine ID with the dispatcher using *registerContextEngineID*, the message is delivered to the command responder using *processPdu*. When the command responder acquires the System group information, it fills the PDU received with System group object values shown in Figure 5.17(b). The *returnResponsePdu* primitive is used by the command generator to deliver the message to the dispatcher. The dispatcher, after processing the get-response message through the MPM and the SM, transmits it across the network using UDP protocol.

7.4.3 Notification Originator

The notification originator application generates either a trap or an inform message. Its function is somewhat similar to the command responder, except that it needs to find out where to send the message and what SNMP version and security parameters to use. Further, the notification generator must determine the *contextEngineID* and context name of the context that has the information to be sent. It obtains

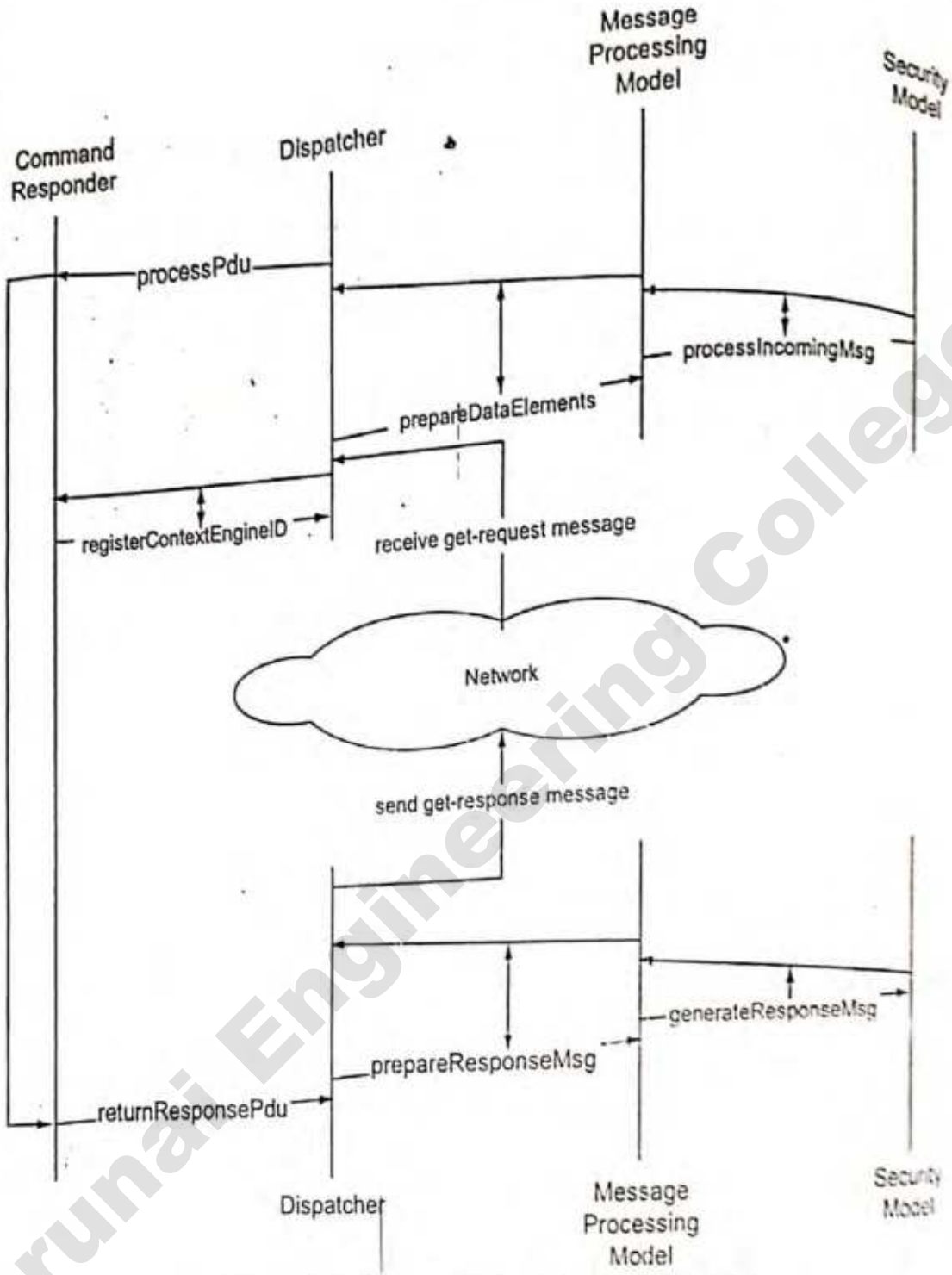


Figure 7.6 Command Responder Application

these data using newly created MIBs for the notification group and the target group, as well as using other modules in the system. We will learn about the new MIBs defining the new groups in Section 7.5. The notification group contains information on whether a notification should be sent to a target and, if so, what filtering should be used on the information. The target that the notification should be sent to is obtained from the target group.

7.4.4 Notification Receiver

The notification receiver application receives SNMP notification messages. It registers with the SNMP engine to receive these messages, just as the command responder does to receive get and set messages.

7.4.5 Proxy Forwarder

The proxy forwarder application performs a function similar to what we discussed in Chapter 6 on the proxy server. However, the proxy definition has been clearly defined and restricted in SNMPv3 specifications. The term "proxy" is used to refer to a proxy forwarder application that forwards SNMP requests, notifications, and responses without regard for what managed objects are contained in those messages. Non-SNMP object translation does not fall under this category. The proxy forwarder handles four types of messages: messages generated by the command generator, command responder, notification generator, and those that contain a report indicator. The proxy forwarder uses the translation table in the proxy group MIB created for this purpose.

7.5 SNMPv3 MANAGEMENT INFORMATION BASE

The new objects defined in SNMPv3 follow the textual convention specified in SNMPv2 and described in Section 6.3. Refer to the RFCs listed in Table 7.1 for complete details on managed objects and MIB groups in SNMPv3. We will address a subset of the MIBs here. Figure 7.7 shows the MIB of the new object groups. They are nodes under *snmpModules* {1.3.6.1.6.3}, shown in Figure 6.31. There are seven new MIB groups. The *snmpFrameworkMIB*, node 10 under *snmpModules*, describes the SNMP Management architecture. The MIB group *snmpMPDMIB* (node 11) identifies objects in message processing and dispatching subsystems.

There are three groups defined under *snmpModules* for applications. They are *snmpTargetMIB* (node 12), *snmpNotificationMIB* (node 13), and *snmpProxyMIB* (node 14). The first two are used for notification generator. The *snmpTargetMIB* defines MIB objects, which are used to remotely configure the parameters used by a remote SNMP entity. There are two tables in that MIB, which are of specific interest for us. They are shown in Figure 7.8. The *snmpTargetAddrTable*, which is in *snmpTargetObjects* group, contains the addresses to be used in the generation of SNMP messages. There are nine columnar objects in the table, which are listed in Table 7.4.

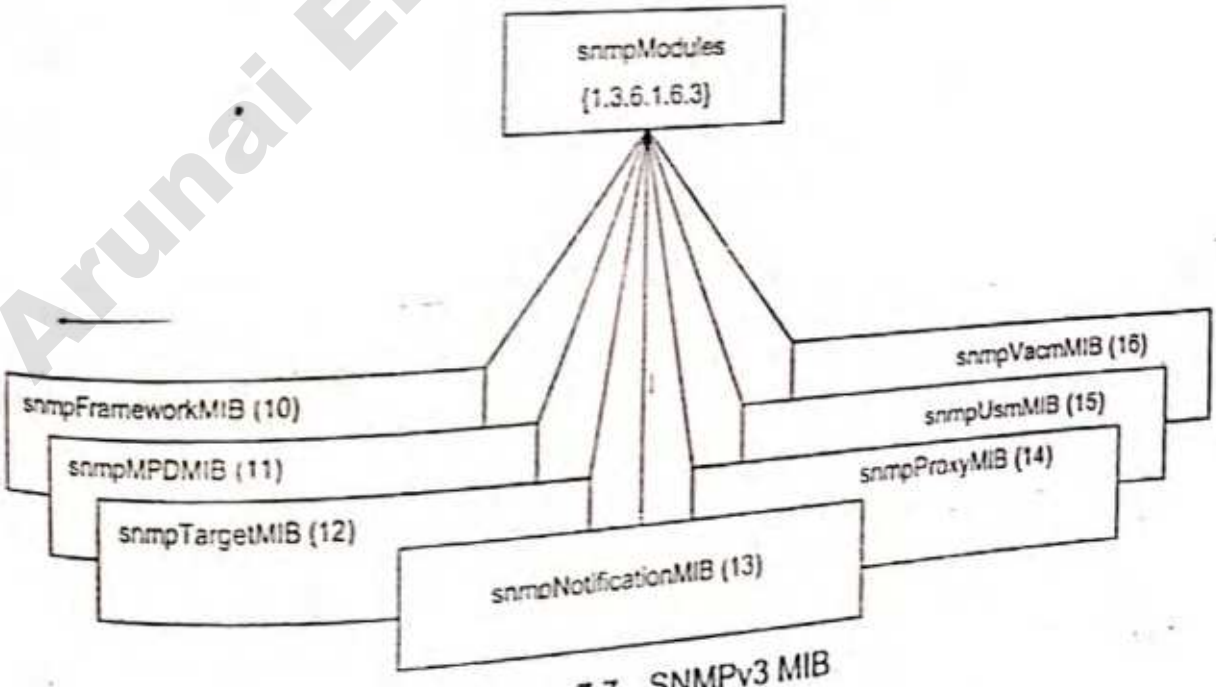


Figure 7.7 SNMPv3 MIB

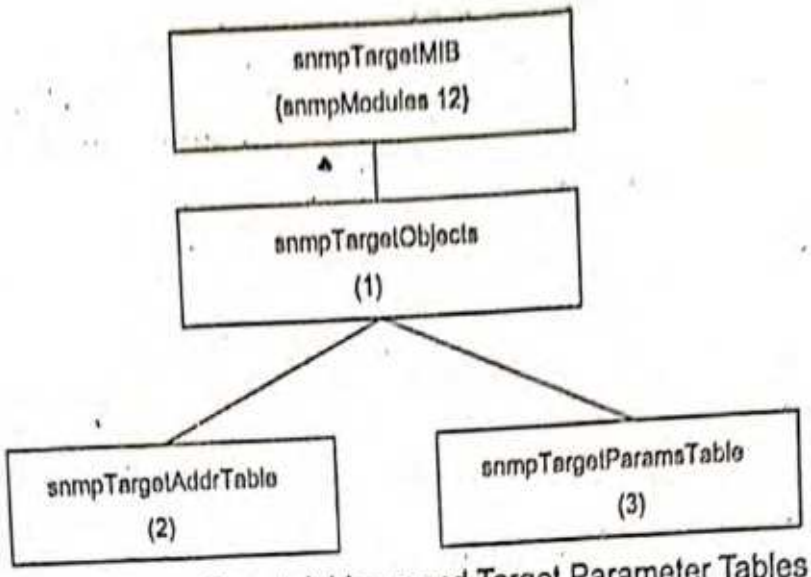


Figure 7.8 Target Address and Target Parameter Tables

Table 7.4 SNMP Target Address Table

ENTITY	OID	DESCRIPTION (BRIEF)
snmpTargetAddrTable	snmpTargetObjects 2	Table of transport addresses
snmpTargetAddrEntry	snmpTargetAddrTable 1	Row in the target address table
snmpTargetAddrName	snmpTargetAddrEntry 1	Locally administered name associated with this entry
snmpTargetAddrTDomain	snmpTargetAddrEntry 2	Transport type of the addresses
snmpTargetAddrTAddress	snmpTargetAddrEntry 3	Transport address
snmpTargetAddrTimeOut	snmpTargetAddrEntry 4	Reflects the expected maximum round-trip time
snmpTargetAddrRetryCount	snmpTargetAddrEntry 5	Number of retries
snmpTargetAddrTagList	snmpTargetAddrEntry 6	List of tag values used to select the target addresses for a particular operation
snmpTargetAddrParams	snmpTargetAddrEntry 7	Value that identifies an entry in the snmpTargetParams Table
snmpTargetAddrStorageType	snmpTargetAddrEntry 8	Storage type for this row
snmpTargetAddrRowStatus	snmpTargetAddrEntry 9	Status of the row

The second table in the *snmpTargetObjects* group is the *snmpTargetParamsTable*. The lead into this table is by using the columnar object *snmpTargetAddrParams* in the *snmpTargetAddrTable*. This contains the security parameters on authentication and privacy. The columnar objects in *snmpTargetParamsTable* are listed in Table 7.5.

The *snmpNotificationMIB* shown in Figure 7.9 deals with MIB objects for the generation of notifications. There are three tables in this group—namely, the notification table, the notification filter profile table, and the notification filter table. They are under the node *snmpNotifyObjects*. The SNMP notification table, *snmpNotifyTable*, is used to select management targets that should receive notifications, as well as the type of notification to be sent. Table 7.6 shows the columnar objects in the group.

The notification profile table group, *snmpNotifyProfileTable*, is used to associate a notification filter profile with a particular set of target parameters. The third group, the notification filter table,

Table 7.5 SNMP Target Parameters Table

ENTITY	OID	DESCRIPTION (BRIEF)
snmpTargetParamsTable	snmpTargetObjects 3	Table of SNMP target information to be used
snmpTargetParamsEntry	snmpTargetParamsTable 1	A set of SNMP target information
snmpTargetParamsName	snmpTargetParamsEntry 1	Locally administered name associated with this entry
snmpTargetParamsMPModel	snmpTargetParamsEntry 2	Message processing model to be used
snmpTargetParamsSecurityModel	snmpTargetParamsEntry 3	Security model to be used
snmpTargetParamsSecurityName	snmpTargetParamsEntry 4	Security name of the principal
snmpTargetParamsSecurityLevel	snmpTargetParamsEntry 5	Level of security
snmpTargetParamsStorageType	snmpTargetParamsEntry 6	Storage type for the row
snmpTargetParamsRowStatus	snmpTargetParamsEntry 7	Status of the row

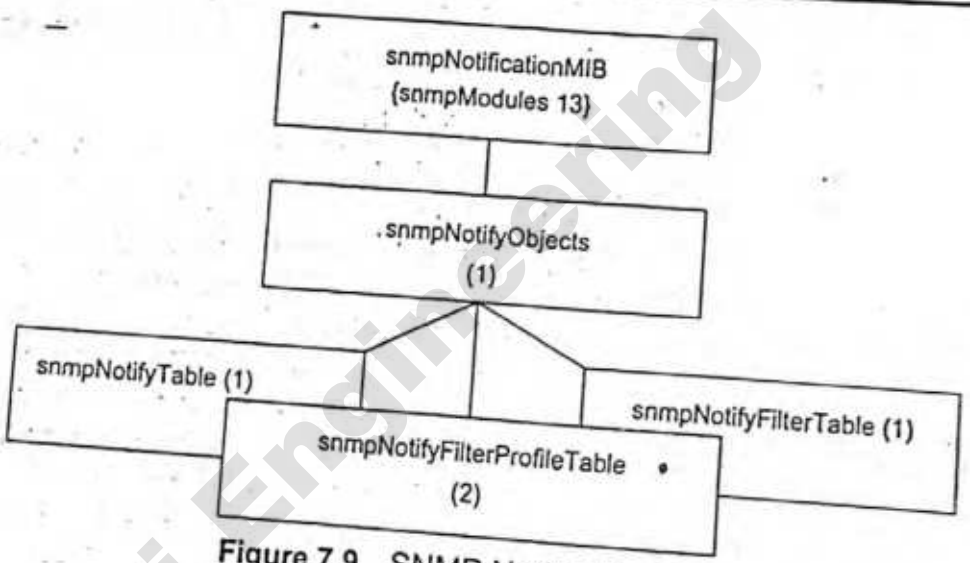


Figure 7.9 SNMP Notification Tables

Table 7.6 SNMP Notification Table

ENTITY	OID	DESCRIPTION (BRIEF)
snmpNotifyTable	snmpNotifyObjects 1	List of targets and notification types
snmpNotifyEntry	snmpNotifyTable 1	Set of management targets and the type of notification
snmpNotifyName	snmpNotifyEntry 1	Locally administered name associated with this entry
snmpNotifyTag	snmpNotifyEntry 2	A single value that is used to select entries in the snmpTargetAddrTable
snmpNotifyType	snmpNotifyEntry 3	Selects trap or inform to send
snmpNotifyStorageType	snmpNotifyEntry 4	Storage type for the row
snmpNotifyRowStatus	snmpNotifyEntry 5	Status of the row

snmpNotifyFilterTable, contains table profiles of the targets. The profile specifies whether a particular target should receive particular information.

The *snmpProxyMIB* is concerned with objects in a proxy forwarding application, such as the SNMPv2 proxy server shown in Figure 6.46. It contains a table of translation parameters used by the proxy forwarder application for forwarding SNMP messages.

The SNMP USM objects are defined in *snmpUsmMIB* module (node 15). Lastly, the objects for VACM for SNMP are defined in the *snmpVacmMIB* module (node 16). We will discuss the details of these MIBs later when we address security in the next section and access control in Section 7.8.

7.6 SECURITY

One of the main objectives, if not the main objective, in developing SNMPv3 is the addition of security features to SNMP management. Authentication and privacy of information, as well as authorization and access controls, have been addressed in SNMPv3 specifications. We will cover the authentication and privacy issues in this section and in Section 7.7. We will deal with access control in Section 7.8.

SNMPv3 architecture permits flexibility to use any protocol for authentication and privacy of information. However, the IETF SNMPv3 working group has specified a USM for its security subsystem. It is termed user-based as it follows the traditional concept of a user, identified by a user name with which to associate security information. The working group has specified HMAC-MD5-96 and HMAC-SHA-96 (see Section 7.7.1 for an explanation) as the authentication protocols. Cipher Block Chaining mode of Data Encryption Standard (CBC-DES) has been adopted for privacy protocol.

We will discuss the general aspects of security associated with the types of threats, the security modules, the message data format to accommodate security parameters, and the use and management of keys in this section. We will specifically address the USM in the next section.

7.6.1 Security Threats

Four types of threats exist to network management information while it is being transported from one management entity to another: (1) modification of information, (2) masquerade, (3) message stream modification, and (4) disclosure. These are shown in Figure 7.10, where the information is transported from management entity A to management entity B. For the first three threats, the signal has to be intercepted by an intruder to tamper with it, whereas for the disclosure threat, the signal can just be copied and not intercepted.

Modification of information is the threat that some unauthorized user may modify the contents of the message while it is in transit. Data contents are modified, including falsifying the value of an object. It does not include changing the originating or destination address. The modified message is received by entity B, which is unaware that it has been modified. For example, the response by an SNMP agent to a request by an SNMP manager could be altered by this threat.

Masquerade is when an unauthorized user sends information to another assuming the identity of an authorized user. This can be done by changing the originating address. Using the masquerade and modification of information, an unauthorized user can perform operation on a management entity, which he or she is not permitted to do. The SNMP set operation should be protected against this attack.

The SNMP communication uses connectionless transport service, such as UDP. This means that the message could be fragmented into packets with each packet taking a different path. The packets could

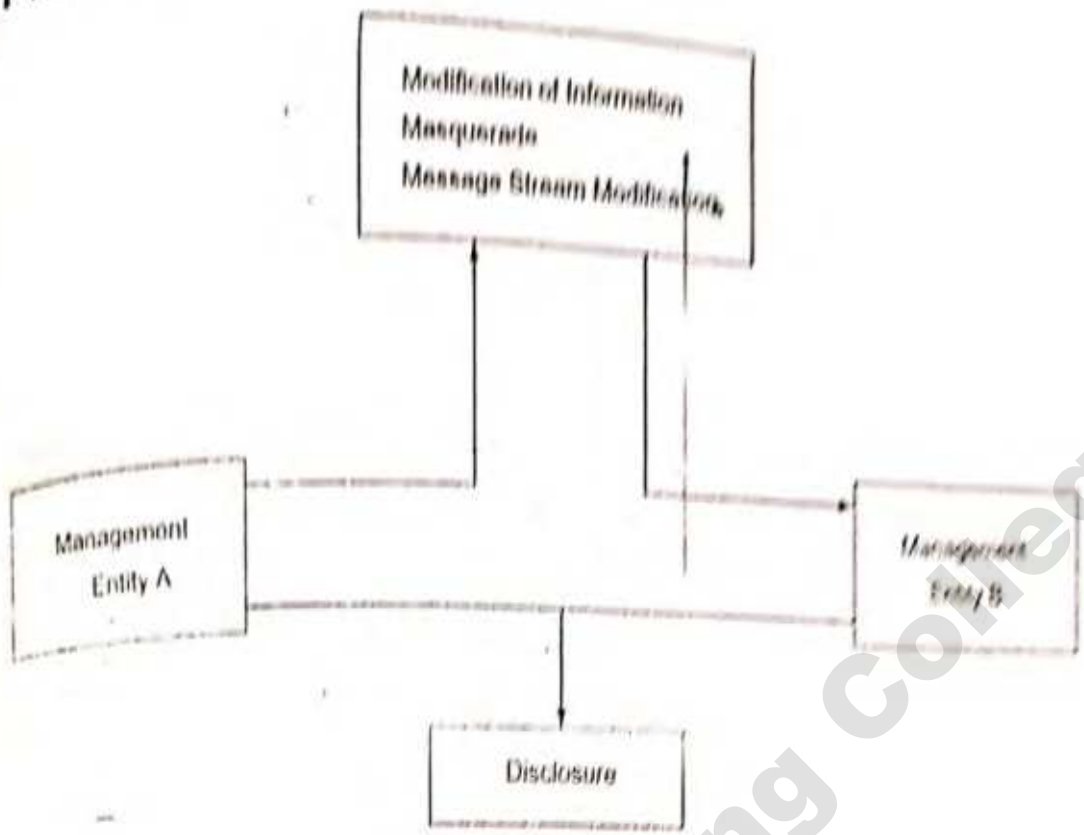
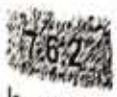


Figure 7.10 Security Threats to Management Information

arrive at the destination out of sequence and have to be reordered. The threat here is that the intruder may manipulate the message stream (**message stream modification**) and maliciously reorder the data packets to change the meaning of the message. For example, the sequence of data of a table could be reordered to change the values in the table. The intruder could also delay messages so that those messages arrive out of sequence. The message could be interrupted, stored, and replayed at a later time by an unauthorized user.

The fourth and last threat that is shown in Figure 7.10 is **disclosure** of management information. The message need not be intercepted for this, but just eavesdropped. For example, the message stream of accounting could be promiscuously monitored by an employee with a TCP/IP dump procedure, and then the information could be used against the establishment.

There are at least two more threats that would be considered as threats in traditional data communication, but the SNMP SM has classified them as being non-threats. The first is denial of service, when an authorized user is denied service by a management entity. This is considered as not being a threat, as a network failure could cause such a denial. It is the responsibility of the protocol to address this issue. The second threat that is not considered a management threat is traffic analysis by an unauthorized user. It was determined by the IETF SNMPv3 working group that there is no significant advantage achieved by protecting against this attack.



Security Model

In normal operational procedures, the MPM in the message processing subsystem interacts with security subsystem models. For example, in Figure 7.2, an outgoing message is generated by an application, which is first handled by a dispatcher subsystem, then by an MPM, and finally by the SM. If the message is to be authenticated, the SM authenticates it and forwards it to the MPM. Similarly for an incoming message, the MPM requests the service of the security subsystem to authenticate the user ID.

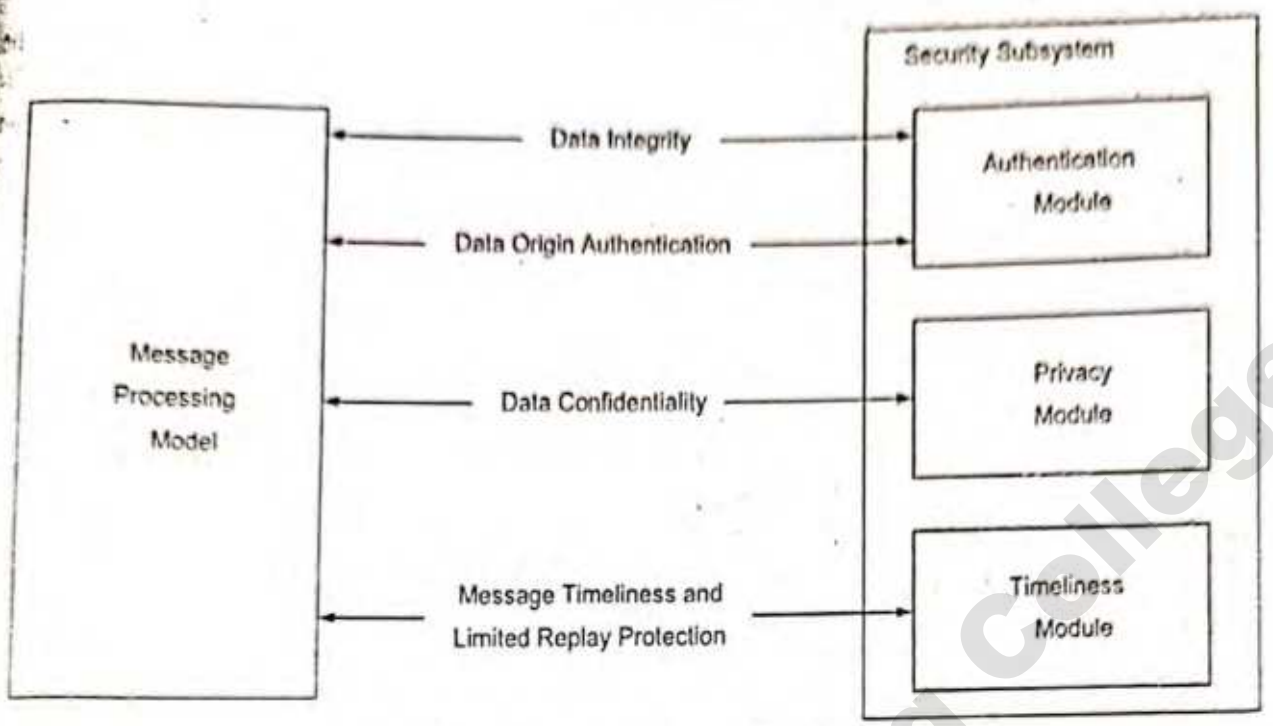


Figure 7.11 Security Services

Figure 7.11 shows the services provided by the three modules in the security subsystem to the MPM. They are the authentication module, the privacy module, and the timeliness module.

Authoritative SNMP Engine. When two management entities communicate, the services provided by each are determined by the role they play, i.e., whether the entity is authorized to perform the service. This led to the concept of authoritative and non-authoritative SNMP engines. This is dependent on which SNMP engine controls the communication between the two entities. SNMPv3 architecture defines that in a communication between two SNMP engines, one acts as an *authoritative engine* and the other as a *non-authoritative engine*. There is a well-defined set of rules as to who is the authoritative SNMP engine for each message that is communicated between two SNMP engines. For get-request, get-next-request, get-bulk-request, set-request or inform messages, the receiver of the message is the authoritative SNMP engine. Since these messages are originated by a manager process in a network management system (NMS), the receiver is the SNMP agent. Thus, the agent is the authoritative SNMP engine. For trap, get-response, and report messages, the sender or the agent is the authoritative SNMP engine. Thus, an SNMP engine that acts in the role of an agent is the designated authoritative SNMP engine. The SNMP engine that acts in the role of a manager is the non-authoritative engine. In general, an SNMP agent is the authoritative SNMP engine in SNMP communication.

An authoritative SNMP engine is responsible for the accuracy of the time-stamp and a unique SNMP engine ID in each message. This requires that every non-authoritative SNMP engine keep a table of the time and authoritative engine ID of every SNMP engine that it communicates with.

Security Authentication. Communication between two entities could satisfy the condition of authoritative and non-authoritative pair. However, it should be the right set of pairs. Thus, the source from which the message is received should be authenticated by the receiver. Further, authentication is needed for the security reasons discussed in Section 7.6.1. Security authentication is done by the authentication module in the security subsystem.

The *authentication module* provides two services, *data integrity* and *data origin authentication*. The data integrity service provides the function of authenticating a message at the originating end and validating it at the receiving end, ensuring that it has not been modified in the communication process by an unauthorized intruder. Authentication validation also catches any non-malicious modification of data in the communication channel. The authentication scheme uses authentication protocols, such as HMAC-MD5-96 or HMAC-SHA-96 in SNMPv3 or any other protocol in place of it.

The second service that is provided by the authentication module is *data origin authentication*. This ensures that the claimed identity of the user on whose behalf the message was sent is truly the originator of the message. The authentication module appends to each message a unique identifier associated with an authoritative SNMP engine.

Privacy of Information. The second module in the security subsystem in Figure 7.11 is the *privacy module*, which provides *data confidentiality service*. Data confidentiality ensures that information is not made available or disclosed to unauthorized users, entities, or processes. The privacy of the message is accomplished by encrypting the message at the sending end and decrypting it at the receiving end.

Timeliness of Message. The *timeliness module* is the third module in the security subsystem and provides the function of checking *message timeliness* and thus prevents message redirection, delay, and replay. Using the concept of an authoritative SNMP engine, a window of time is set in the receiver to accept a message. The travel time between the sender and the receiver should be within this time window interval. The time clock in both the sender and the receiver is synchronized to the authoritative SNMP engine. The recommended value for the window time in SNMPv3 is 150 seconds.

For implementation of the timeliness module, the SNMP engine maintains three objects: *snmpEngineID*, *snmpEngineBoots*, and *snmpEngineTime*. The *snmpEngineID* uniquely identifies the authoritative SNMP engine. The *snmpEngineBoots* is a count of the number of times the SNMP engine has re-booted or re-initialized since *snmpEngineID* was last configured. The *snmpEngineTime* is the number of seconds since the *snmpEngineBoots* counter was last initialized or reset.

The timeliness module also checks the message ID of a response with the request message and drops the message if they do not match.

We will next look at the message format in SNMPv3 in general, and the security parameters contained in it in particular.

7.6.3 Message Format

The SNMPv3 message format is shown in Figure 7.12. It consists of four groups of data. Details of the fields in each group except security parameters are given in Table 7.7. The first group is a single field, which is the version number and is in the same position as in SNMPv1 and SNMPv2.

Global (header) data defined by the data type header are the second group of data in the message format. They contain administrative parameters of the message, which are message ID, message maximum size, message flag, and message SM. It is worth noting that an SNMP engine can handle many models concurrently in the message processing subsystem. The dispatcher subsystem examines the version number in the message and sends it to the appropriate message processing module in the message subsystem. For example, if the version is set to snmpv2, the SNMPv2 message processing module would be invoked.

The third group of data, security parameter fields, are used by the SM in communicating between sending and receiving entities. The values of the parameters depend on the message SM set in the header data. The parameters are shown in Figure 7.12 and will be discussed in Section 7.7.

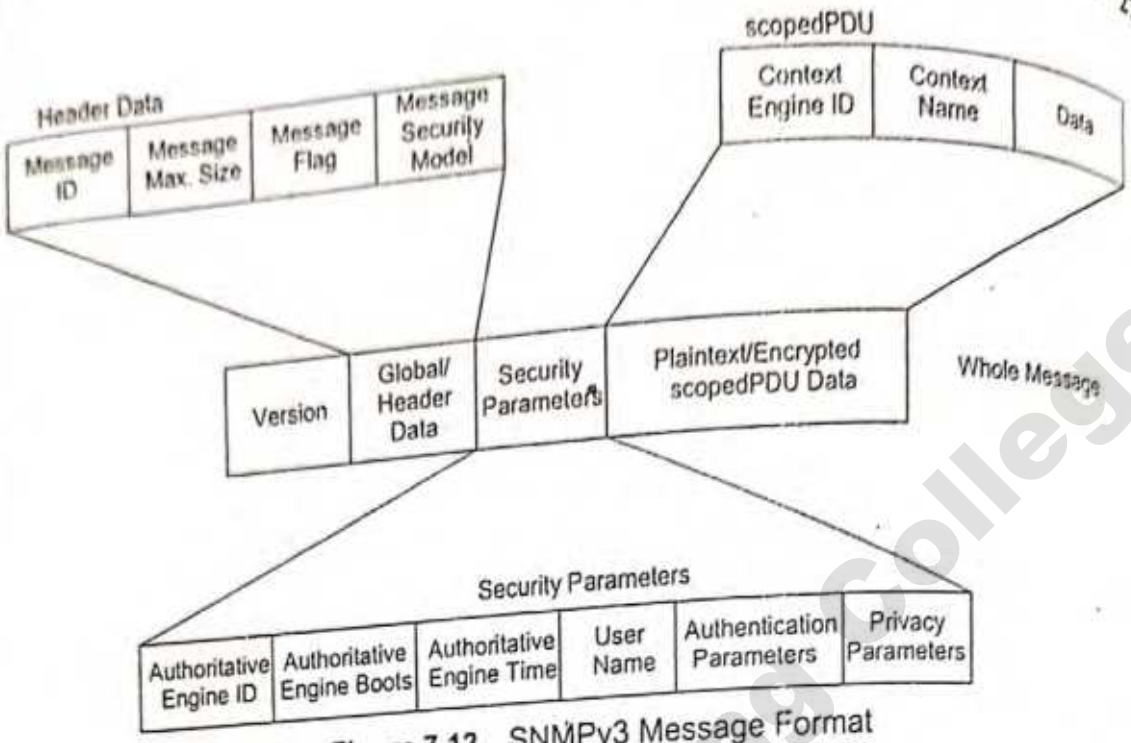


Figure 7.12 SNMPv3 Message Format

Table 7.7 SNMPv3 Message Format

FIELD	OBJECT NAME	DESCRIPTION
Version	msgVersion	SNMP version number of the message format
Message ID	msgID	Administrative ID associated with the message
Message max. size	msgMaxSize	Maximum size supported by the sender
Message flags	msgFlags	Bit fields identifying report, authentication, and privacy of the message
Message security model	msgSecurityModel	Security model used for the message; concurrent multiple models allowed
Security parameters (See Table 7.8)	msgSecurityParameters	Security parameters used for communication between sending and receiving security modules
Plaintext/encrypted scopedPDU data	scopedPduData	Choice of plaintext or encrypted scopedPDU; scopedPDU uniquely identifies context and PDU
Context engine ID	contextEngineID	Unique ID of a context (managed entity) with a context name realized by an SNMP entity
Context name	contextName	Name of the context (managed entity)
PDU	data	Contains unencrypted PDU

The fourth and final group of fields in the whole message record shown in Figure 7.12 is the plaintext/encrypted *scopedPDU* data. The *scopedPduData* field contains either unencrypted or encrypted *scopedPDU*. If the privacy flag is set to zero (no privacy) in the message flag (see header data), then this field contains plaintext *scopedPDU*, which is unencrypted *scopedPDU*. The plaintext *scopedPDU*

comprises the context engine ID, context name, and the PDU. A management entity can be responsible for multiple instances of managed objects. For example, in an ATM switch, a single managed entity acts as the agent for all the network interface cards in all its ports. We could treat each interface card as a context with a context engine ID and a context name. Thus, a particular context name, in conjunction with a particular context engine ID, identifies the particular context associated with the management information contained in the PDU portion of the message. The object name for PDU is *data*.

7.7 SNMPv3 USER-BASED SECURITY MODEL

The security subsystem for SNMPv3 is a USM, which is based on the traditional user name concept. Just as we have defined abstract service interfaces between various subsystems in an SNMP entity, we can define abstract service interfaces in USM. They define conceptual interfaces between generic USM services and self-contained authentication and privacy services. There are two primitives associated with authentication service, one to generate an outgoing authenticated message (*authenticateOutgoingMsg*) and another to validate the authenticated incoming message (*authenticateIncomingMsg*). Similarly, there are two primitives associated with privacy services, *encryptData* and *decryptData*, for the encryption of outgoing messages and the decryption of incoming messages. These were included in the list of primitives in Table 7.3.

Services provided by authentication and privacy modules in the security subsystem for outgoing and incoming messages are shown in Figures 7.13 and 7.14, respectively. Looking at the overall picture, the MPM invokes the USM in the security subsystem. The USM in turn invokes, based on the security level set in the message, the authentication and privacy modules. Results are returned to the MPM by the USM.

In Figure 7.13 that shows the process of an outgoing message, we will assume that both privacy and authentication flags are set in the message flag in the header data. The MPM inputs the MPM information, header data, security data, and *scopedPDU* to the security subsystem. The USM invokes the privacy module first, providing the encryption key and *scopedPDU* as input. The privacy module outputs

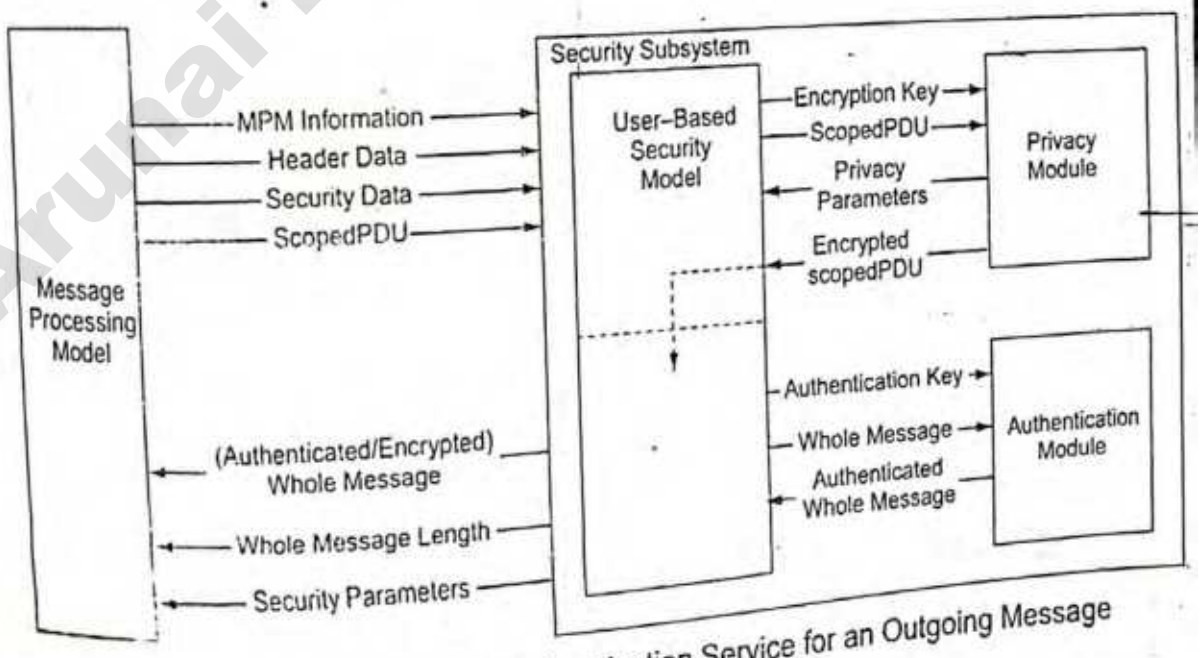


Figure 7.13 Privacy and Authentication Service for an Outgoing Message

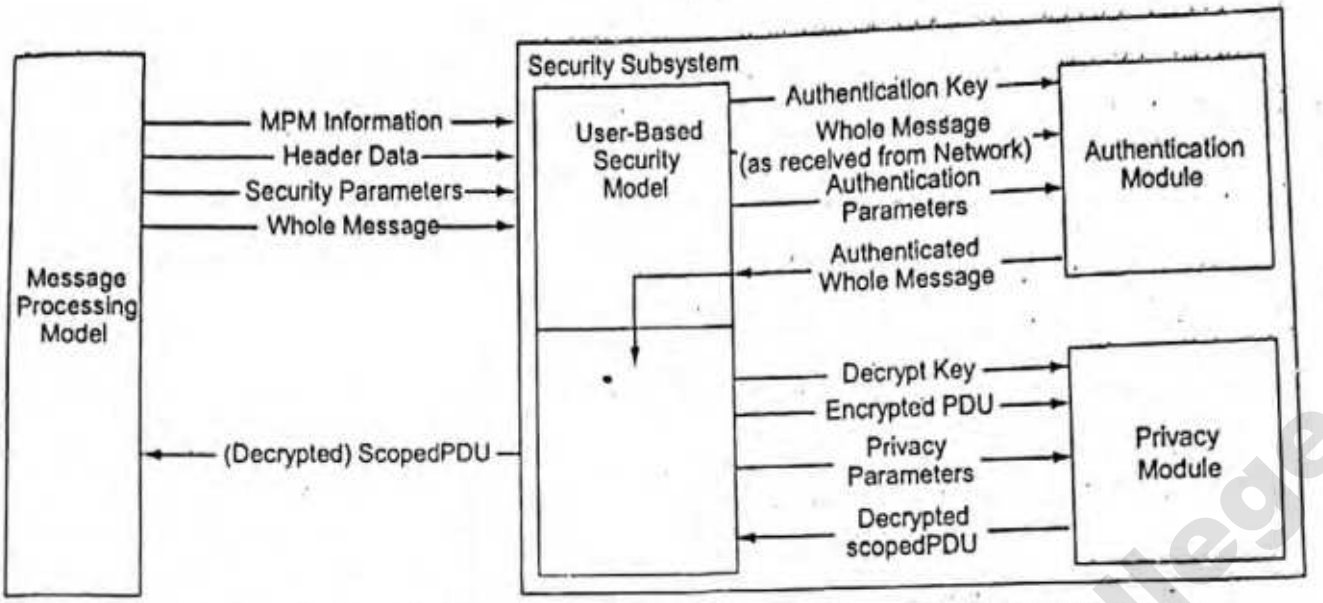


Figure 7.14 Privacy and Authentication Service for an Incoming Message

privacy parameters that are sent as part of the message and the encrypted *scopedPDU*. The USM passes the unauthenticated whole message with encrypted *scopedPDU* to the authentication module along with the authentication key. The authentication module returns the authenticated whole message to USM. The security subsystem returns the authenticated and encrypted whole message along with the message length and security parameters to the MPM.

Figure 7.14 shows the reverse process of an incoming message going through the authentication validation first, and then decryption of the message by the privacy module.

The security parameters used in the SM are shown in Figure 7.12. Table 7.8 lists the parameters and the corresponding SNMPv3 MIB objects. The position of the relevant MIB objects associated with the security parameters belongs to the two modules, *snmpFrameworkMIB* and *snmpUsmMIB*, under *snmpModulesMIB* shown in Figure 7.7. The details of the position of the objects in the MIB are presented in Figure 7.15.

We have already discussed the first three parameters in Table 7.8 associated with engine ID, number of boots, and time since the last boot. They are in the *snmpEngine* group shown in Figure 7.15. The last three parameters in the table are in *usmUserTable* in the *usmUser* group shown in Figure 7.15.

The fourth parameter is the user (principal) on whose behalf the message is being exchanged. The authentication parameters are defined by the authentication protocol columnar object in the *usmUserTable*. The *usmUserTable* describes the users configured in the SNMP engine and the authentication parameters the type of authentication protocol used. Likewise, the privacy parameters describe the type of privacy protocol used.

The *usmUserSpinLock* is an advisory lock that is used by SNMP command generator applications to coordinate their use of the set operation in creating or modifying secrets in *usmUserTable*.

Now that we have a broad picture, let us return to Figure 7.13 and follow through the detailed data flow and processes involved in the USM. Figure 7.13 shows the operation for an outgoing message, which could be either a Request message or a Response message. The MPM inputs information on the *message processing model* to be used (normally SNMP version number), leader data, security data (SM, SNMP engine ID, security name, and security level) and *scopedPDU* to the Security Subsystem (SS). This information is received by the User-base Security Model (UCM) in SS.

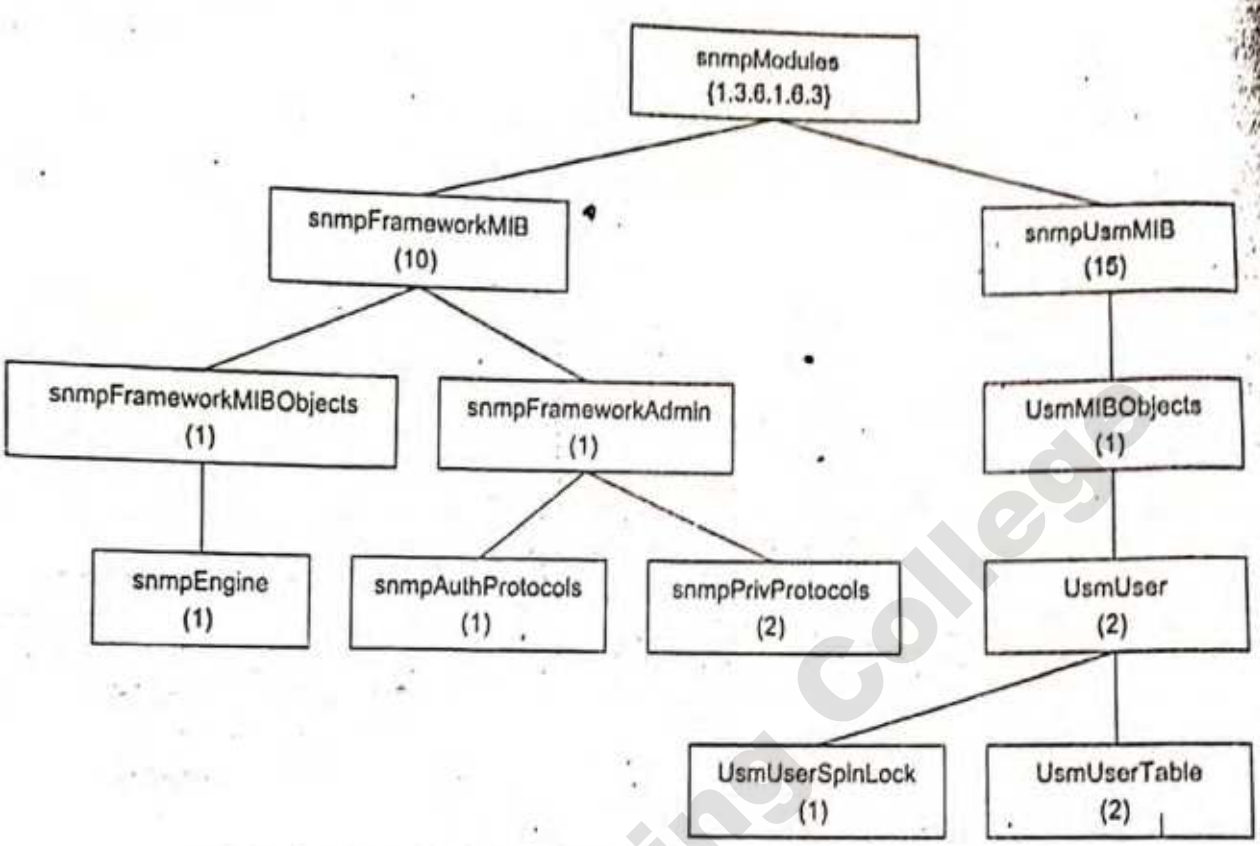


Figure 7.15 SNMPv3 MIB Objects for Security Parameters

Table 7.8 Security Parameters and Corresponding MIB Objects

SECURITY PARAMETERS	USM USER GROUP OBJECTS
msgAuthoritativeEngineID	snmpEngineID (under snmpEngine Group)
msgAuthoritativeEngineBoots	snmpEngineBoots (under snmpEngine Group)
msgAuthoritativeEngineTime	snmpEngineTime (under snmpEngine Group)
msgUserName	usmUserName (in usmUserTable)
msgAuthenticationParameters	usmUserAuthProtocol (in usmUserTable)
msgPrivacyParameters	usmUserPrivProtocol (in usmUserTable)

In the USM, the security-level settings for privacy and authentication determine the modules invoked. The encryption key and *scopedPDU* (context engine ID, context name, and PDU) are fed into the privacy module, which encrypts the PDU and returns the encrypted PDU along with privacy parameters to the calling module, USM.

The USM then communicates with the authentication module. The USM inputs the encrypted whole message along with the authentication key. The authentication module returns the authenticated whole message to USM. The USM passes the authenticated and encrypted whole message, whole message length, and securities parameters back to the MPM.

The operation for an incoming message is shown in Figure 7.14. Inputs to the security subsystem are the MPM information, header data, security parameters for the received message, and the whole message. The output of the security subsystem is *scopedPDU* in plaintext format.

Within the security subsystem, the operational sequence of authentication and privacy for an incoming message are reversed from that of the outgoing message. The message is first sent to the authentication module with the authentication key, the whole message received from the network, and authentication parameters received from the network as inputs. It outputs an authenticated whole message to the calling module in USM. The USM then feeds the decrypt key, privacy parameters, and the encrypted *scopedPDU* and receives in return the decrypted *scopedPDU*. The decrypted *scopedPDU* is then passed on to the message processing module.

7.7.1 Authentication Protocols

The secret to security using the authentication and privacy schemes is the secret key that is shared between the sender and the user. There is a secret key for authentication and a secret key for encryption and decryption. The secret key for the User-based Security Module (USM) is developed from the user password. Two algorithms are recommended in SNMPv3 for developing the key from the password. They are HMAC-MD5-96 and HMAC-SHA-96. The first letter in the designation stands for the cryptographic hash function (H) used for generating the Message Access Code (MAC). The second part in the designation is the hashing algorithm used, the first one being the MD5 hashing algorithm, and the second one the SHA-1 hashing algorithm to generate MAC. The MAC is derived by truncating the hashing code generated to 96 bits as indicated by the last set of characters in the designation.

Authentication Key. The authentication key, the secret key for authentication, is derived from a chosen password of the user. The user in our case is the non-authoritative SNMP engine, which is generally an NMS. In both MD5 and SHA-1 algorithms, the password is repeated until it forms exactly a string of 2^{20} octets (1,048,576 octets), truncating the last repetition, if necessary. This result is called *digest0* [Stallings, 1998]. In the second step, the *digest0* is hashed using either the MD5 or the SHA-1 algorithm to derive *digest1*. MD5 algorithm yields a 16-octet *digest1*, and SHA-1 results in a 20-octet *digest1*. A second string is formed by concatenating the authoritative SNMP engine ID and *digest1*. This string is fed into the respective hashing algorithm to derive *digest2*. The derived *digest2* is the user's authentication key, *authKey*, which is input to the authentication modules shown in Figures 7.13 and 7.14. You are referred to RFC [2104] and Stallings [1998] for details on MD5 and SHA-1 algorithms.

The choice between the 16-octet MD5-based *authKey* and the 20-octet SHA-1-based *authKey* is based on the implementation. In the 20-octet key, it is harder to break the code than in the 16-octet key. However, the processing is faster with the 16-octet key. Further, the same 16-octet key derived from the same password could be used for the privacy key, although it is recommended that the same key not be used for both.

HMAC Procedure. The 96-bit long code MAC is derived using the HMAC procedure described in RFC 2104 and RFC 2274. First, two functions K1 and K2 are derived using *authKey* obtained above, and two fixed but different strings, *ipad* and *opad*, as defined in the following manner. A 64-byte *extendedAuthKey* is derived by supplementing *authKey* with zeros.

ipad = the hexadecimal byte 0x36 (00110110) repeated 64 times
opad = the hexadecimal byte 0x5c (01011100) repeated 64 times
K1 = *extendedAuthKey* XOR *ipad*
K2 = *extendedAuthKey* XOR *opad*

HMAC is computed by performing the following nested hashing functions on K1, K2, and *wholeMsg*, which is the unauthenticated whole message shown in Figure 7.13:

H (K2, H (K1, *wholeMsg*))

The first 12 octets of this final digest are the MAC. These are the authentication parameters, *msgAuthenticationParameters*, which are shown in Figure 7.12 and are included as part of the authenticated whole message, *authenticatedWholeMsg* shown in Figure 7.13.

Key Management. A user (NMS) has only one password and hence one *secret* key *digest1* mentioned in the authentication key discussion earlier. However, it communicates with all the authoritative SNMP engines (all the agents in the network). The shared information is again a secret between the two communicating engines. The concept of a localized key is introduced to accomplish this instead of storing a separate password for each pair of communicating engines. A hash function, which is the same hashing function that is used to generate the secret key, is employed to generate the localized key.

Localized key = H (*secret, authoritativeSnmpEngineID, secret*)

where *secret* is the secret key (*digest1*) and the *authoritativeSnmpEngineID* is the SNMP engine ID of the authoritative SNMP engine with which the local user is communicating. This localized key, different for each authoritative engine, is stored in each authoritative engine with which the user communicates. Notice that the localized key is the same as *authKey*.

SNMPv3 permits the operation of changes and modification in a key, but not the creation of keys to ensure that the secret key does not become stale.

Discovery. One important function of an NMS as a user is the discovery of agents in the network. This is accomplished by generating a Request message with a security level of no-authentication and no-privacy, a user name of "initial," an authoritative SNMP engine ID of zero length, and a varBind list that is empty. The authoritative engines respond with Response messages containing the engine ID and the security parameters filled in. Additional information is then obtained via pair-wise communication messages.

7.7.2 Encryption Protocol

The encryption generates non-readable *ciphertext* from a readable text, *plaintext*. The SNMPv3 recommendation for data confidentiality is to use the CBC-DES Symmetric Encryption Protocol. The USM specifications require the *scopedPDU* portion of the message be encrypted. A secret value in combination with a timeliness value is used to create the encryption/decryption key and initialization vector (IV). Again, the secret value is user-based, and hence is associated typically with an NMS. The 16-octet privacy key, *privKey*, is generated from the password as described in the generation of authentication code using MD-5 hashing algorithm.

The first eight octets of the 16-octet privacy key are used to create the DES key. The DES key is only 56 bits long and hence the least significant bit of each octet in the privacy key is discarded. The 16-octet IV is made up of two parts, an 8-octet pre-IV concatenated with an 8-octet *salt*. The pre-IV is the last eight octets of the privacy key. The *salt* is added to ensure that two identical instances of ciphertext are not generated from two different plaintexts using the same key. The *salt* is generated by an SNMP engine by concatenating a 4-octet *snmpEngineBoots* with a locally generated integer. The *salt* is the privacy parameter shown in Figures 7.12, 7.13, and 7.14

The encryption process first divides the plaintext of *scopedPDU* into 64-bit blocks. The plaintext of each block is XOR-ed with the ciphertext of the previous block, and the result is encrypted to produce a ciphertext for the current block. For the first block, the IV is used instead of the ciphertext of the previous block.

7.8 ACCESS CONTROL

We have covered security considerations in network management with regard to data integrity, message authentication, data confidentiality, and the timeliness of message in the previous two sections. We will now address access control, which deals with who can access network management components and what they can access. In SNMPv1 and SNMPv2, this subject has been covered using the community-based access policy. In SNMPv3, access control has been made more secure and more flexible. It is called VACM.

VACM defines a set of services that an application in an agent can use to validate command requests and notification receivers. It validates command requests as to the sending sources and their access privilege. It assumes that authentication of the source has been done by the authentication module. In order to perform the services, a local database containing access rights and policies has been created in the SNMP entity, called Local Configuration Datastore (LCD). This is typically in an agent or in a manager functioning in an agent role when it communicates with another manager.

The LCD needs to be configured remotely and hence security considerations need to be introduced. A MIB module for VACM has been introduced toward achieving this.

7.8.1 Elements of the Model

Five elements comprise VACM: (1) groups, (2) security level, (3) contexts, (4) MIB views and view families, and (5) access policy. We will define each of them now.

Groups. A group, identified as *groupName*, is a set of zero or more SM (*vacmSecurityModel*)—security name (*vacmSecurityName*) pairs on whose behalf SNMP management objects can be accessed. A security name is a principal as defined in Section 7.3.2 and is independent of the SM used. All elements belonging to a group have identical access rights. Equivalent of a group in SNMPv1 is the community name. Thus, all NMSs (security names) in SNMPv1 (SM) with a community name public (group) would have equal access privilege to an agent.

Security Level. Security level (*vacmAccessSecurityLevel*) is the level of security of the user, namely no authentication–no privacy, authentication–no privacy, and authentication–privacy. This is set by the message flag shown in Figure 7.12. A member using a specific SM and with a given security name in a group could have different access rights by using different security levels.

Contexts. As mentioned in Section 7.3, an SNMP context is a collection of management information accessible by an SNMP entity. An SNMP entity has access to potentially more than one context. Each SNMP engine has a context table that lists the locally available contexts by *contextName*.

MIB Views and View Families. As in SNMPv1 and SNMPv2, access rights to contexts are controlled by a MIB view (see Figure 5.2). A MIB view is defined for each group and it details the set of managed object types (and optionally, the specific instances of object types). Following the approach of the

tree-like naming structure for MIB, the MIB view is defined as a combination of a set of *view subtrees*, where each view subtree is a subtree within the managed object naming tree. A simple MIB view could be all nodes defined under an OBJECT IDENTIFIER, for example, *system*. A view subtree is identified by the OBJECT IDENTIFIER value, which is the longest OBJECT IDENTIFIER prefix common to all (potential) MIB object instances in that subtree. For the system example, it is {1.3.6.1.2.1.1}.

An example of a complex MIB view could be all information relevant to a particular network interface. This can be represented by the union of multiple view subtrees, such as a set of *system* and *interfaces* to view all managed objects under the System and Interfaces groups.

A more complex view is a situation where all the columnar objects in a conceptual row of a table appear in separate subtrees, one per column, each with a similar format. Because the formats are similar, the required set of subtrees can be aggregated into one structure, called a *family of view subtrees*. A family of view subtrees is a pairing of an OBJECT IDENTIFIER value (called the family name) together with a bit string value (called the family mask). The family mask indicates which subidentifiers of the associated family name are significant to the family's definition. A family of view subtrees can either be included or excluded from the MIB view.

Access Policy: The access policy determines the access rights to objects as *read-view, write-view,* and *notify-view*. For a given *groupName*, *contextName*, *securityModel*, and *securityLevel*, that group's access rights are defined by either the combination of the three views, or *not-accessible*. The read-view is used for get-request, get-next-request, and get-bulk-request operations. The write-view is used with the set-request operation. The notify-view represents the set of object instances authorized for the group when sending objects in a notification.

7.8.2 VACM Process

The VACM process is presented as a flowchart in Figure 7.16. We will explain the process in terms of an SNMP agent with an SNMP engine having responsibility for many contexts. The tables shown in the figure are addressed in the next section on VACM MIB. As RFC 2275 describes, the VACM process answers the six questions related to access of management information. They are:

1. Who are you (group comprising security model and security name)?
2. Where do you want to go (context to be accessed)?
3. How secured are you to access the information (security model and security level)?
4. Why do you want to access the information (to read, write, or send notification)?
5. What object (object type) do you want to access?
6. Which object (object instance) do you want to access?

The first question is answered by the introduction of the group concept. The group that the requester belongs to is determined by the VACM from the SM and the security name. It uses the security-to-group table for validating the principal and deriving the group name.

The second question is answered by checking whether the context that needs to be accessed is within the responsibility of the agent. If the first two questions are answered in the affirmative, then the results of those, namely *group name* and *context name*, along with the *security model* and the *security level* (answer to how), are fed into the "access allowed?" process. It is assumed by VACM that the SM and the security level (the third question) have already been validated by the security module. Given these four

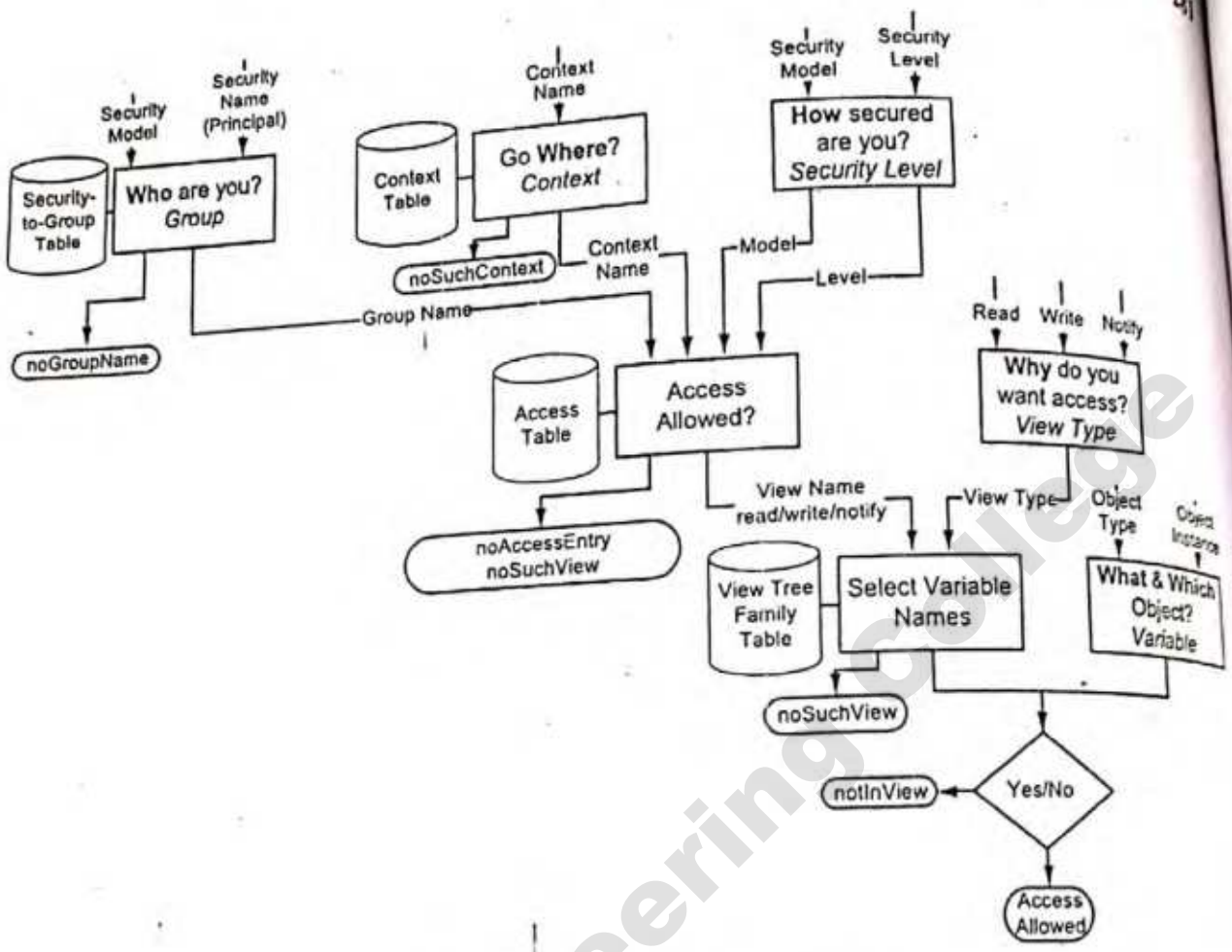


Figure 7.16 VACM Process

inputs as indices, the access table provides the-views permitted, *view name*. It comprises one or more of the views, read-view, write-view, and notify-view.

The answer to the fourth question regarding why access is needed is used by the "select variable names" process to select the family of view subtrees eligible to be accessed. The view tree family table is applicable for this selection. A match is made between the result of this process and the answers to the last two questions as to what (object type) and which (object instance), to make a decision on whether access is allowed or not.

Each process puts out an error message based on the validation as shown in Figure 7.16.

7.8.3 VACM MIB

The processes in VACM use the tables to perform the functions mentioned. A VACM MIB has been defined specifying the newly created objects. This is shown in Figure 7.17. The *snmpVacmMIB* is a node under *snmpModules* shown in Figure 7.7. The three tables defining the context, group, and access are nodes under *vacmMIBObjects*, which is a node under *snmpVacmMIB*.

The *vacmContextTable* is a list of *vacmContextNames*. The *vacmSecurityToGroupTable* has columnar objects, *vacmSecurityModel*, *vacmSecurityName*, as indices to retrieve *vacmGroupName*.

The VACM Access Table, shown in Figure 7.18, is used to determine the access permission and the *viewName*. It has *vacmGroupName* from the *vacmSecurityToGroupTable* as one of the index

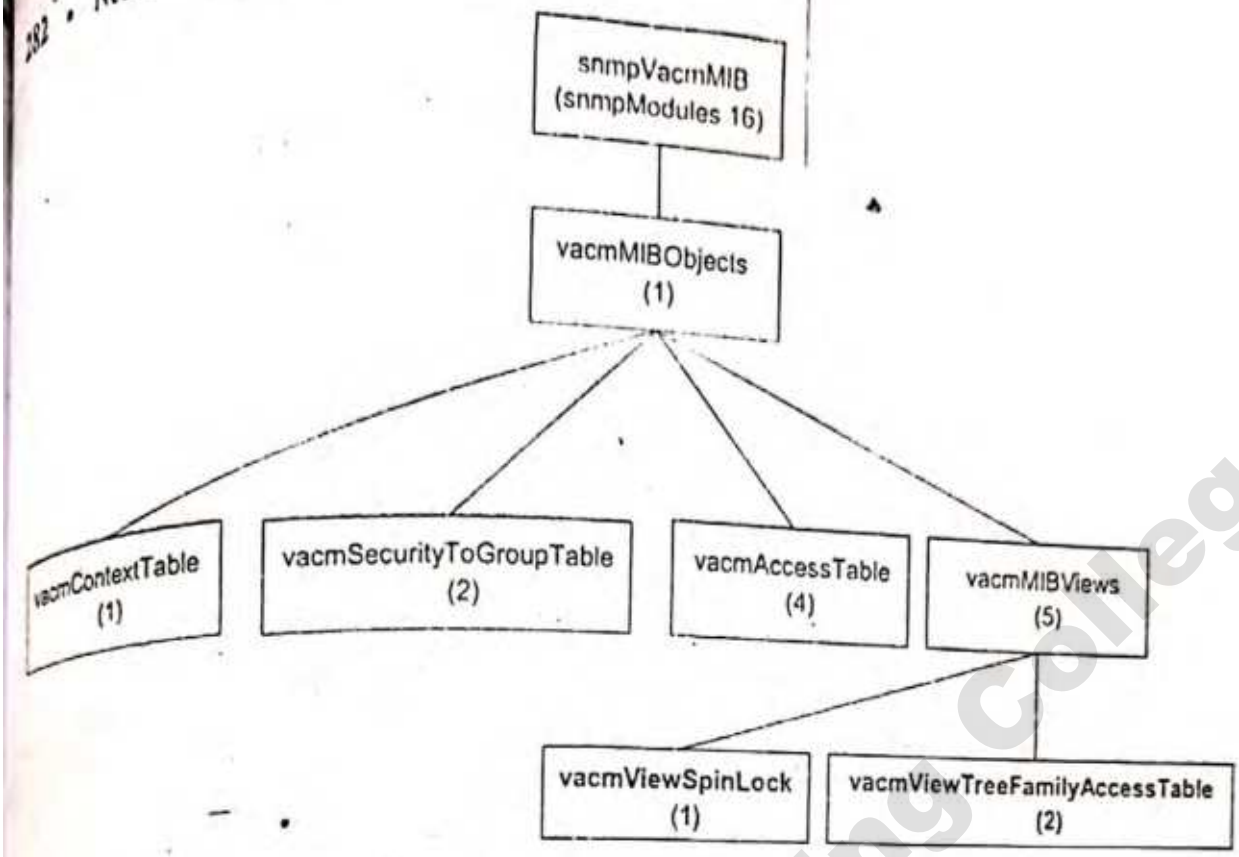


Figure 7.17 VACM MIB

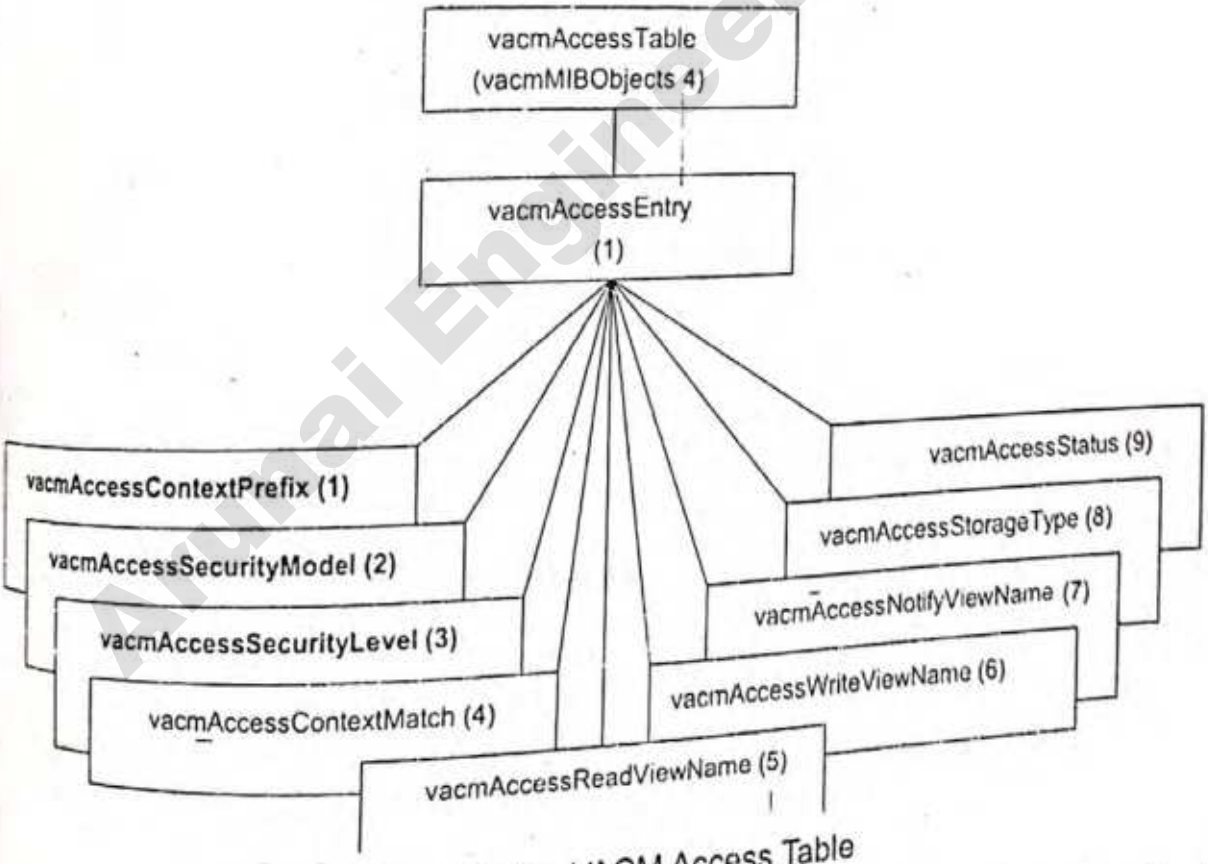


Figure 7.18 VACM Access Table

The other three indices from this table are *vacmAccessContextPrefix*, *vacmAccessSecurityModel*, and *vacmAccessSecurityLevel*. The *viewName* representing the three views, *vacmAccessReadViewName*, *vacmAccessWriteViewName*, and *vacmAccessNotifyViewName*, are retrieved from the table. The

vacmAccessStorageType and *vacmAccessStatus* are the administrative information objects relating to the storage volatility and the row status.

The *vacmMIBViews*, subnode (5), under *vacmMIBObjects*, shown in Figure 7.19, has the subordinate nodes *vacmViewSpinLock* and *vacmViewTreeFamilyAccessTable*. The *vacmViewSpinLock* is an advisory lock that is used by SNMP command generator applications to coordinate their use of the set operation in creating or modifying views in agents. It is an optional implementation object.

The *vacmViewTreeFamilyTable* describes families of subtrees that are available within MIB views in the local SNMP agent for each context. Each row in this table describes a subtree for a *viewName* and an OBJECT IDENTIFIER. For example, if the "access allowed?" process in Figure 7.16 yields three values for *viewName*, that would result in three conceptual rows in this table. The *vacmViewTreeFamilyViewName* representing the *viewName* is one of the columnar objects and an index in the table. Two indices define a conceptual row in this table. The second is *vacmViewTreeFamilySubtree*. It is a node representing the top of the tree. For example, if the OBJECT IDENTIFIER were 1.3.6.1.2.1.1, it would represent the *system* subtree. The OBJECT IDENTIFIER for the local agent is determined by the highest OBJECT IDENTIFIER that would address all object instances in the local view.

In some situations, we may want to view different subsets of a subtree. In such cases, we can form a family of view subtrees by using a combination of two parameters. The first is the selection of the view, which is done by a family mask defined by *vacmViewTreeFamilyMask*; and the second parameter is the family type defined by *vacmViewTreeFamilyType*, shown in Figure 7.19. The family mask is a bit string that is used with the *vacmViewTreeFamilySubtree*. Using this feature, specific objects in a subtree are selected if the corresponding object identifier matches. If the corresponding bit value is 0 in the family mask, it is considered a wild card and any value of the object identifier would be

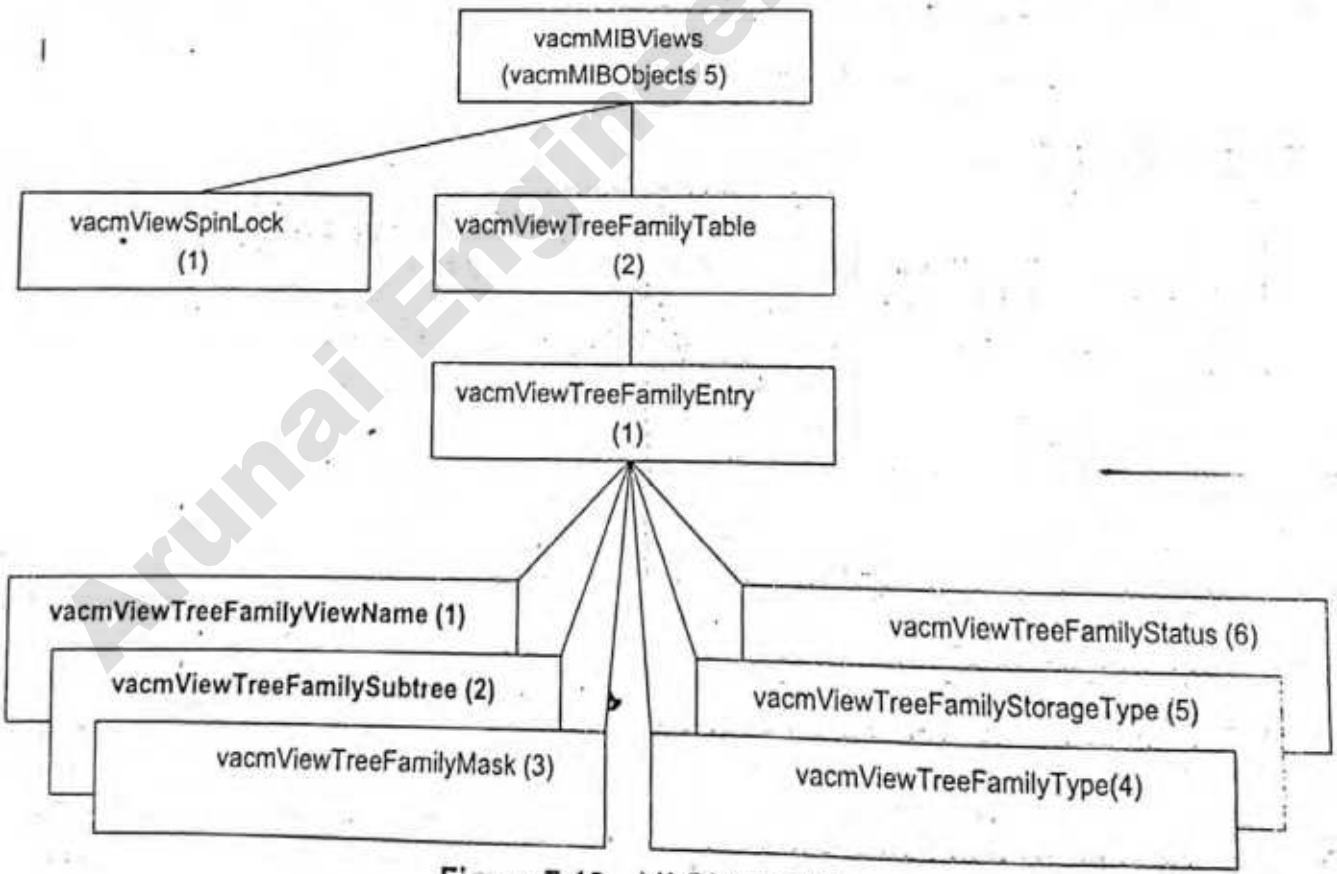


Figure 7.19 VACM MIB Views

selected. After the selection is made, if the family type value is included (1), the view is included. If it is excluded (2), the view is excluded. There is more flexibility in the views by introducing a columnar object *vacmViewTreeFamilyType* that indicates whether a particular subtree in the family of subtrees derived from *vacmViewTreeFamilySubtree* and *vacmViewTreeFamilyMask* is to be included or excluded in a context's MIB view.

As an example of the System group to be included, values for various parameters of the family entry in Figure 7.19 are:

Family view name = "system"
Family subtree = 1.3.6.1.2.1.1
Family mask = ""
Family type = 1

The zero length string, "", for mask value designates all 1s by convention.

We could extend the view by adding a second row to the table. For example, we could add an SNMP group by adding another row to the table with the family subtree 1.3.6.1.2.1.11.

Suppose we want to add a columnar object to the table. We would add the columnar object with the index added as another row. We could also add all the columnar objects of a conceptual row in a table. A useful convention for doing this is to use the definition of columnar object 0, which designates all columnar objects in a table. For example, {1.3.6.1.2.1.2.2.1.0.5} identifies all columnar objects associated with the 5th interface (corresponding to *ifIndex* value of 5) in the *ifTable*.

If more than one family name is present with the same number of subidentifiers, the lexicographic convention is followed for the predominance among them. This helps in the following way. Suppose we wanted to choose all columnar objects in the above *ifTable* example, except the *ifMtu*, which is the 4th columnar object. We would then choose {1.3.6.1.2.1.2.2.1.4.5} and the Type = 2 to exclude it. Since this is lexicographically higher than {1.3.6.1.2.1.2.2.1.0.5}, this will take precedence. Thus, the combination of the two will select all 5th row objects except *ifMtu*.

Summary

We have reviewed the latest version of SNMP, SNMPv3, in this chapter. The two major features are the specifications for a formalized SNMP architecture that addresses the three SNMP frameworks for the three versions. Two new members, dispatch and message processing modules, are defined. This would enable a network management system to handle messages from and to agents that belong to all three current versions. It would also accommodate future versions, if needed.

The second major feature is the inclusion of security. A security subsystem is defined, which addresses data integrity, data origin authentication, data confidentiality, message timeliness, and limited message replay protection. The authentication module in the security subsystem addresses the first two issues, the privacy module protects data confidentiality, and the timeliness module deals with message timeliness and limited replay protection. The security subsystem is the User-based Security Model (USM). It is derived from the traditional concept of user ID and password.

The access policies of SNMPv1 and SNMPv2 have been extended and made more flexible by the VACM. An SNMP agent handling multiple objects (contexts) can be configured to present a set of MIB views and a family of subtrees in its MIB views. These views can be matched with seven input parameters to determine access permission to the principal. They are the SM (version of SNMP), the security name (principal), the security level (dependent on the authentication and privacy parameters), the context name, the type of access needed, the object type, and the object instance.

selected. After the selection is made, if the family type value is included (1), the view is included. If it is excluded (2), the view is excluded. There is more flexibility in the views by introducing a columnar object *vacmViewTreeFamilyType* that indicates whether a particular subtree in the family of subtrees derived from *vacmViewTreeFamilySubtree* and *vacmViewTreeFamilyMask* is to be included or excluded in a context's MIB view.

As an example of the System group to be included, values for various parameters of the family entry in Figure 7.19 are:

Family view name = "system"
Family subtree = 1.3.6.1.2.1.1
Family mask = ""
Family type = 1

The zero length string, "", for mask value designates all 1s by convention.

We could extend the view by adding a second row to the table. For example, we could add an SNMP group by adding another row to the table with the family subtree 1.3.6.1.2.1.11.

Suppose we want to add a columnar object to the table. We would add the columnar object with the index added as another row. We could also add all the columnar objects of a conceptual row in a table. A useful convention for doing this is to use the definition of columnar object 0, which designates all columnar objects in a table. For example, {1.3.6.1.2.1.2.2.1.0.5} identifies all columnar objects associated with the 5th interface (corresponding to *ifIndex* value of 5) in the *ifTable*.

If more than one family name is present with the same number of subidentifiers, the lexicographic convention is followed for the predominance among them. This helps in the following way. Suppose we wanted to choose all columnar objects in the above *ifTable* example, except the *ifMtu*, which is the 4th columnar object. We would then choose {1.3.6.1.2.1.2.2.1.4.5} and the Type = 2 to exclude it. Since this is lexicographically higher than {1.3.6.1.2.1.2.2.1.0.5}, this will take precedence. Thus, the combination of the two will select all 5th row objects except *ifMtu*.

Summary

We have reviewed the latest version of SNMP, SNMPv3, in this chapter. The two major features are the specifications for a formalized SNMP architecture that addresses the three SNMP frameworks for the three versions. Two new members, dispatch and message processing modules, are defined. This would enable a network management system to handle messages from and to agents that belong to all three current versions. It would also accommodate future versions, if needed.

The second major feature is the inclusion of security. A security subsystem is defined, which addresses data integrity, data origin authentication, data confidentiality, message timeliness, and limited message replay protection. The authentication module in the security subsystem addresses the first two issues, the privacy module protects data confidentiality, and the timeliness module deals with message timeliness and limited replay protection. The security subsystem is the User-based Security Model (USM). It is derived from the traditional concept of user ID and password.

The access policies of SNMPv1 and SNMPv2 have been extended and made more flexible by the VACM. An SNMP agent handling multiple objects (contexts) can be configured to present a set of MIB views and a family of subtrees in its MIB views. These views can be matched with seven input parameters to determine access permission to the principal. They are the SM (version of SNMP), the security name (principal), the security level (dependent on the authentication and privacy parameters), the context name, the type of access needed, the object type, and the object instance.

Exercises

1. The first four octets of an SNMP engine ID in a system are set to the binary equivalent of the system's SNMP management private enterprise number as assigned by the IANA. Write the first four octets of the SNMP engine ID in hexadecimal notation for the four enterprises, (cisco, hp, 3com, and cabletron,) shown in Figure 4.14 for the following two versions:
   (a) SNMPv1
   (b) SNMPv3
2. Write the full SNMP engine ID for:
   (a) SNMPv1 for a 3Com hub with the IPv4 address 128.64.46.2 in the 6th to 9th octets followed by 0s in the rest.
   (b) SNMPv3 for the Cisco router interface with IPv6 address ::128.64.32.16.
3. Describe the SNMPv3 *scopedPDU* that the SNMP agent (router) responds to NMS with the data shown in Figure 4.2(c).
4. Figure 7.20 shows a generalized time-sequenced operation for get-request message going from a manager to an agent. Complete the primitives in Figure 7.20 explicitly identifying the application modules used.

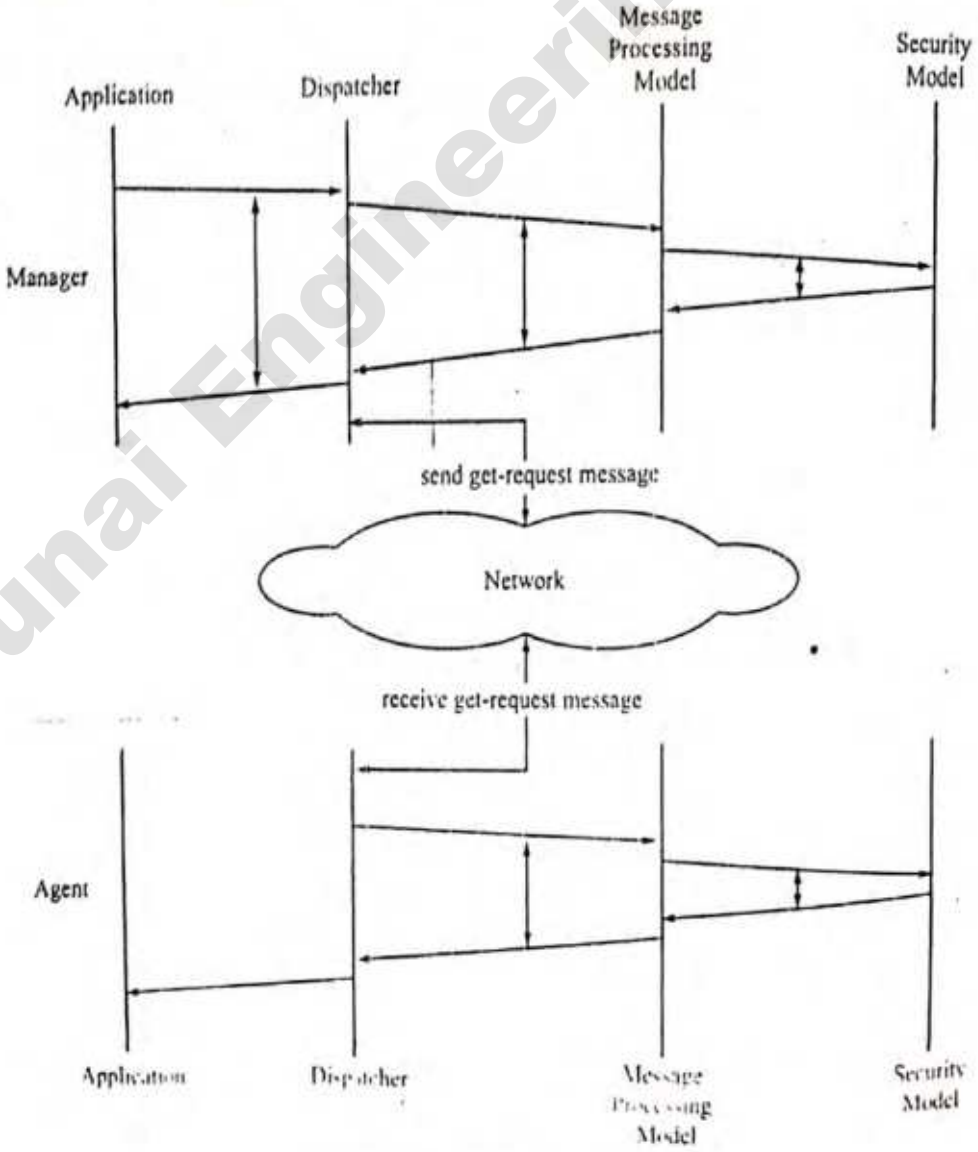
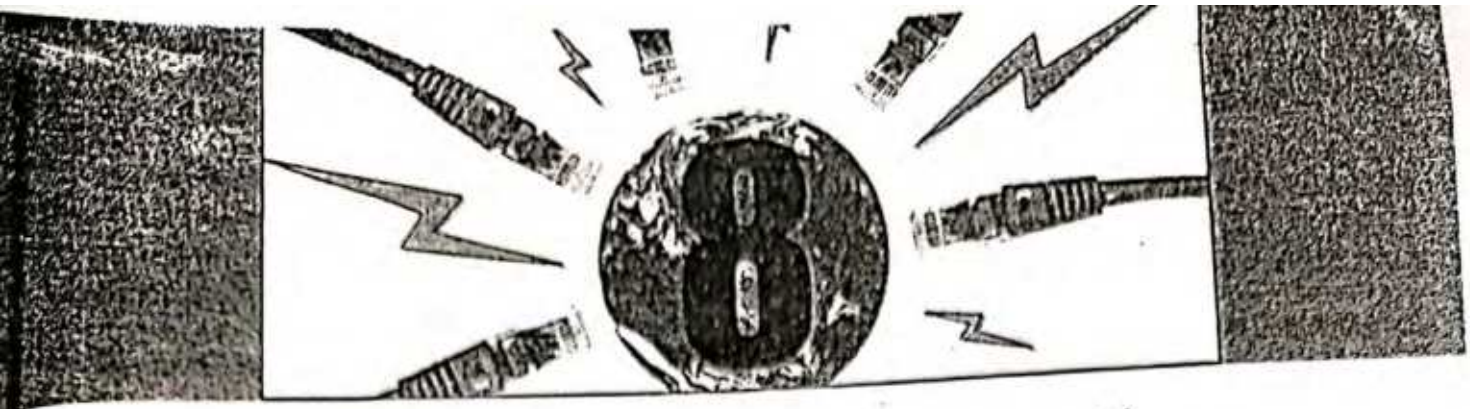


Figure 7.20 Exercise 4

5. Draw the time-sequence operation similar to that in Figure 7.20 detailing the elements of procedure for get-response message from the agent to the manager.
6. Detail the IN and OUT parameters of the *sendPdu* and *prepareOugoingMsg* primitives shown in Figure 7.4(b) by referring to RFC 2271.
7. Identify the authoritative and non-authoritative entities in Figure 7.20.
8. Define the configuration parameters for a notification generator to send traps to two network management systems *noc1* and *noc2* by filling in the objects in the *snmpTargetAddressTable*, *snmpTargetTable*, and *snmpNotifyTable*. Specifications for the two targets are given below. You may use the Appendix of RFC 2273 as a guide to answer this exercise.

	noc1	noc2
messageProcessingModel	SNMPv3	SNMPv3
securityModel	3 (USM)	3 (USM)
secuirtyName	"noc1"	"noc2"
snmpTargetParamsName	"NOAuthNoPriv-noc1"	"NOAuthNoPriv-noc1"
securityLevel	noAuthNoProv(1)	authPriv(3)
transportDomain	snmpUDPDomain	snmpUDPDomain
transportAddress	128.64.32.16:162	128.64.32.8:162
tagList	"group1"	"group2"

9. Access RFC 2274 and list and define the primitives provided by the authentication module at the sending and receiving security subsystems. Describe the services provided by the primitives.
10. Access RFC 2274 and list and define primitives provided by the privacy module at the sending and receiving security subsystems. Describe the services provided by the primitives.
11. Specify the family name, the family subtree, the family mask, and the family type in *vacmViewTreeFamilyTable* for an agent to present a view of:
    (a) the complete IP group
    (b) IP address table (*ipAddrTable*)
    (c) the row in the IP address table corresponding to the IP address 172.46.62.1
12. Write the *vacmViewTreeFamilyTable* for the three rows that present the system group in the IP address table for the row with IP address 172.46.62.1 without the *ipAdEntReasmMaxSize*.



SNMP Management: RMON

OBJECTIVES

- *Remote network monitoring: RMON*
- *RMON1: Monitoring Ethernet LAN and token-ring LAN*
- *RMON2: Monitoring upper protocol layers*
- *Generates and sends statistics close to subnetworks to central NMS*
- *RMON MIBs for RMON group objects*

The success of SNMP management resulted in the prevalence of managed network components in the computer network. SNMPv1 set the foundation for monitoring a network remotely from a centralized network operations center (NOC) and performing fault and configuration management. However, the extent to which network performance could be managed was limited. The characterization of the performance of a computer network is statistical in nature. This led to the logical step of measuring the statistics of important parameters in the network from the NOC and the development of remote monitoring (RMON) specifications.

8.1 WHAT IS REMOTE MONITORING?

We saw examples of SNMP messages going across the network between a manager and an agent in Section 5.1.4. We did this using a tool that "sniffs" every packet that is going across a local area network (LAN), opens it, and analyzes it. It is a passive operation and does nothing to the packets, which continue to proceed to their destinations. This is called monitoring or probing the network and the device that does the function is called the network monitor or the probe. Let us distinguish between the two components of a probe: (1) physical object that is connected to the transmission medium and (2) processor, which analyzes the data. If both are at the same place geographically, it is a local probe, which is how sniffers used to function. We will discuss this further in Chapter 9, when we consider management systems and tools.

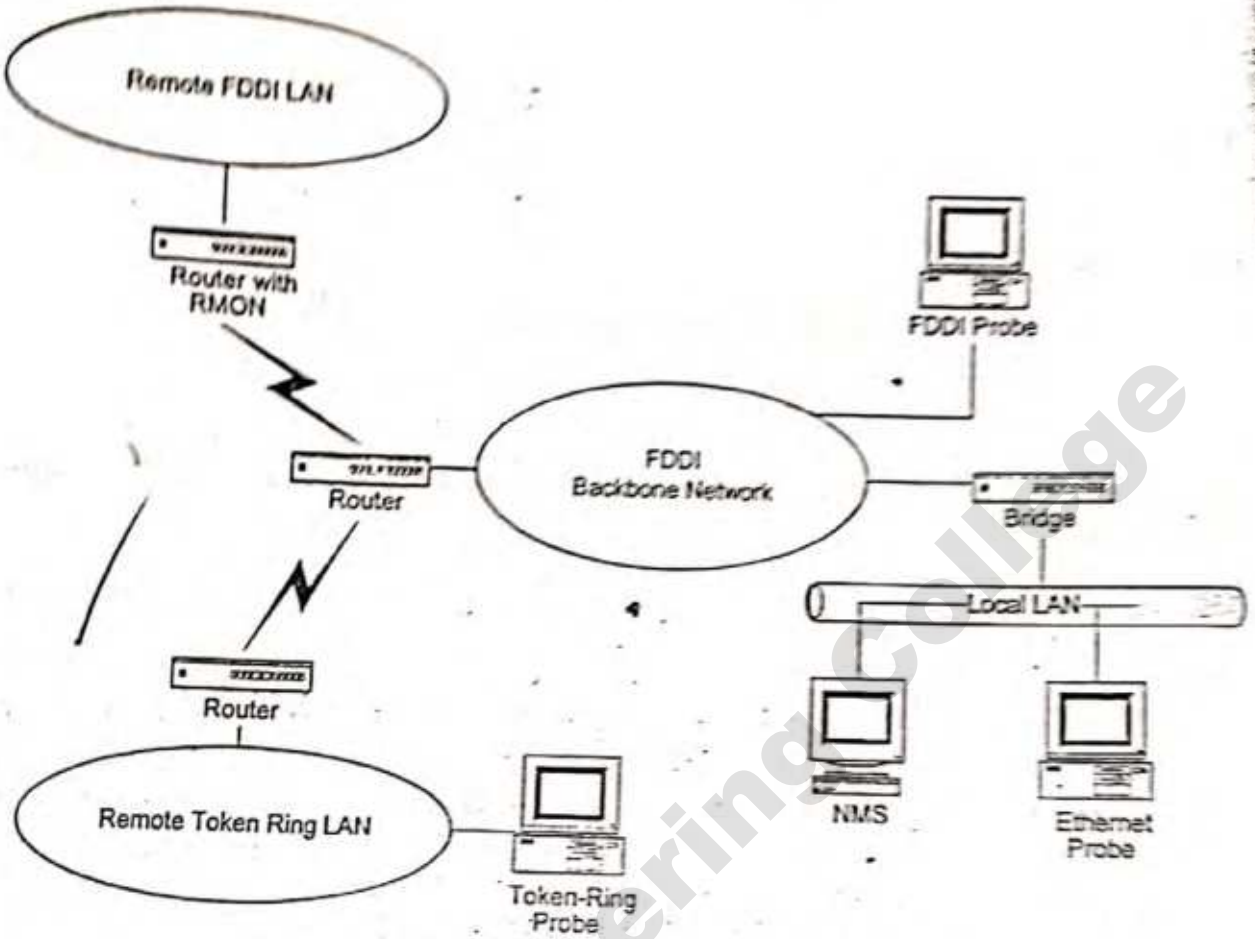


Figure 8.1 Network Configuration with RMONs

The monitored information gathered and analyzed locally can be transmitted to a remote network management station. In such a case, remotely monitoring the network using a probe is referred to as remote network monitoring or RMON. Figure 8.1 shows a fiber-distributed data interface (FDDI) backbone network with a local Ethernet LAN. There are two remote LANs, one a token-ring LAN and another, an FDDI LAN, connected to the backbone network. The network management system (NMS) is on the local Ethernet LAN. There is either an Ethernet probe or an RMON on the Ethernet LAN monitoring the local LAN. The FDDI backbone is monitored by an FDDI probe via the bridge and Ethernet LAN. A token-ring probe monitors the token-ring LAN. It communicates with the NMS via routers and the wide area network (WAN) (shown by the lightening bolt symbol of the telecommunications link). The remote FDDI is monitored by the built-in probe on the router. The FDDI probe communicates with the NMS via the WAN. All four probes that monitor the four LANs and communicate with the NMS are RMON devices.

The use of RMON devices has several advantages. First, each RMON device monitors the local network segment and does the necessary analyses. It relays the necessary information in both solicited and unsolicited fashion to the NMS. For example, RMON could be locally polling network elements in a segment. If it detects an abnormal condition, such as heavy packet loss or excessive collisions, it would send an alarm. Because the polling is local, the information is more reliable. This example of local monitoring and reporting to a remote NMS significantly reduces SNMP traffic in the network. This is especially true for the segment in which the NMS resides, as all the monitoring traffic would otherwise converge there.

The following case history illustrates another advantage. RMON reduces the necessity of agents in the network to be visible at all times to the NMS. One of the NMSs would frequently indicate that one of the hubs would show failure, but the hub recovered itself without any intervention. The performance study of the hub that the LAN was part of indicated that the LAN would frequently become overloaded with heavy traffic, and would have a significant packet loss. That included the ICMP packets that the NMS was using to poll the hub. The NMS was set to indicate a node failure if three successive ICMP packets did not receive responses. Increasing the number of packets needed to indicate a failure stopped the failure indication. This demonstrates the third advantage.

There are more chances that the monitoring packets, such as ICMP pings, may get lost in long-distance communication, especially under heavy traffic conditions. This may wrongly be interpreted by the NMS as the managed object being down. RMON pings locally and hence has less chance of losing packets, thus increasing the reliability of monitoring.

Another advantage of local monitoring using RMON is that individual segments can be monitored on a more continuous basis. This provides better statistics and greater ability for control. Thus, a fault could be diagnosed quicker by the RMON and reported to the NMS. In some situations, a failure could even be prevented by proactive management.

The overall benefits of implementing RMON technology in a network are higher network availability for users and greater productivity for administrators. A study report [CISCO/RMON] indicates increased productivity of several times for network administrators using RMON in their network.

8.2 RMON SMI AND MIB

For a network configuration system, like the one shown in Figure 8.1, to work successfully, several conditions need to be met. Network components are made by different vendors. Even the RMON devices may be from different vendors. Thus, just as in the communication of network management information, standards need to be established for common syntax and semantics for the use of RMON devices. The syntax used is ASN.1. The RMON structure of management information is similar to SMIv2 in defining object types. The Remote Network Monitoring Management Information Base (RMON MIB) defining RMON groups has been developed and defined in three stages. The original RMON MIB, now referred to as RMON1 was developed for Ethernet LAN in November 1991 RFC 1271, but was made obsolete in 1995 RFC 1757. Token-ring extensions to RMON1 were developed in September 1993 [RFC 1513]. The use of RMON1 for remote monitoring was found to be extremely beneficial. However, it addressed parameters at the OSI layer 2 level only. Hence, RMON2 [RFC 2021] was developed and released in January 1997, which addressed the parameters associated with OSI layers 3 through 7.

The RMON group is node 16 under MIB-il (mib-2 16), as shown in Figure 6.36. All the groups under the RMON group are shown in Figure 8.2. It consists of nine Ethernet RMON1 groups (rmon 1 to rmon 9), one token-ring extension group to RMON1 (rmon 10), and nine RMON2 groups (rmon 11–20) for the higher layers.

RMON1 is covered in Section 8.3 and RMON2 in Section 8.4. We will discuss the applications of RMON in Part III when we discuss applications, systems, and tools.

8.3 RMON1

RMON1 is covered by RFC 1757 for Ethernet LAN and RFC 1513. There are two data types introduced as textual conventions, and ten MIB groups (rmon 1 to rmon 10), as shown in Figure 8.2.

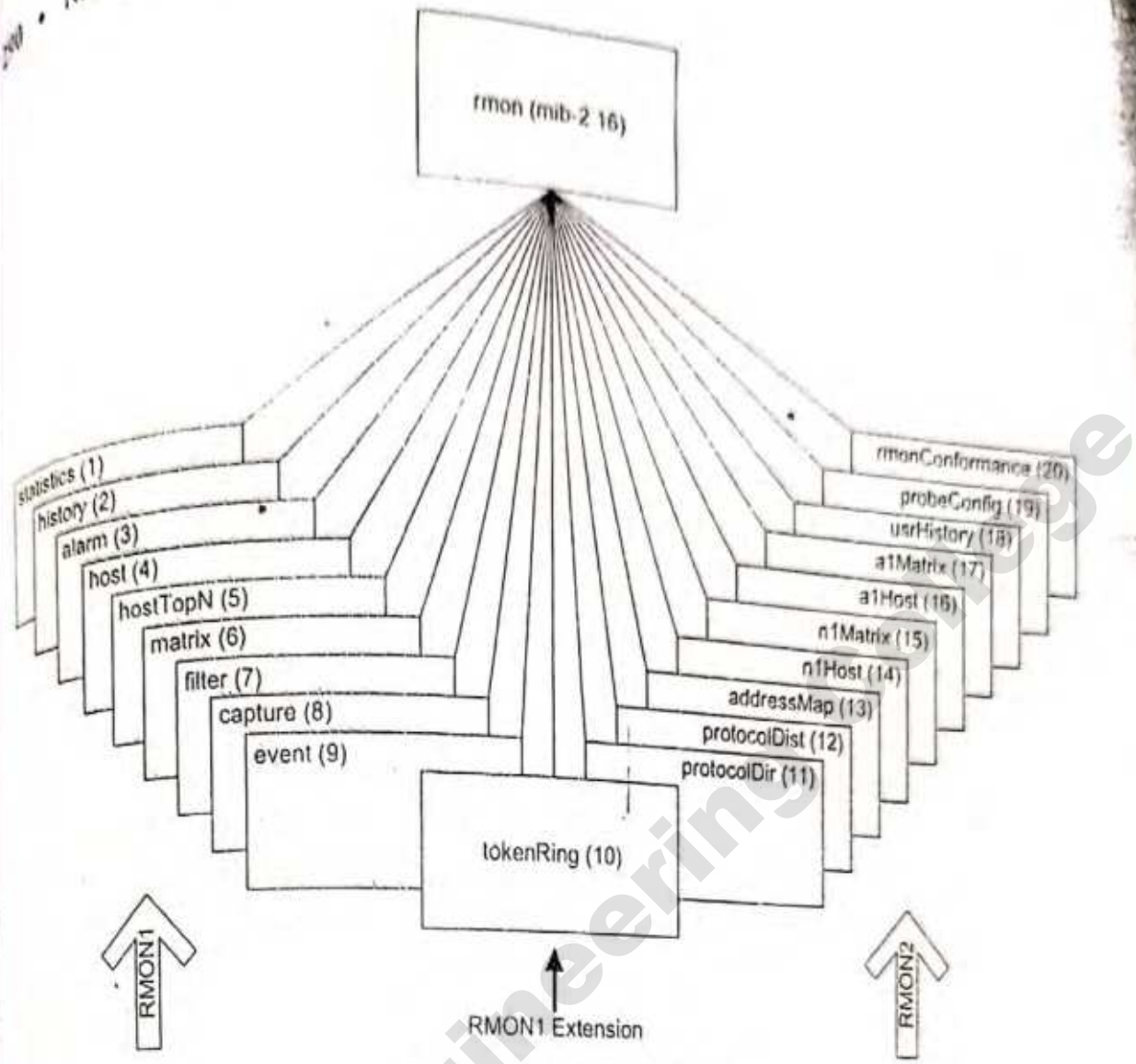


Figure 8.2 RMON Group

8.3.1 RMON1 Textual Conventions

Two new data types that are defined in RMON1 textual conventions are *OwnerString* and *EntryStatus*. Both these data types are extremely useful in the operation of RMON devices. RMON devices are used by management systems to measure and produce statistics on network elements. We will soon see that this involves setting up tables that control parameters to be monitored. Typically, there is more than one management system in the network, which could have permission to create, use, and delete control parameters in a table. Or, a human network manager in charge of network operations does such functions. For this purpose, the owner identification is made part of the control table defined by the *OwnerString* data type. The *EntryStatus* is used to resolve conflicts between management systems in manipulating control tables.

The *OwnerString* is specified in the NVT ASCII character set specified by *DisplayString*. The information content of *OwnerString* contains information about the owner: IP address, management station name, network manager's name, location, or telephone number. If the agent itself is the owner, as for example in the addition of an interface card, the *OwnerString* is set to "monitor."

In order to understand the data type, *EntryStatus*, we need to understand the concept of creation and deletion of rows in tables, which was discussed in Section 6.4.7. For a table to be shared by multiple users, a columnar object *EntryStatus*, similar to *RowStatus* in SNMPv2, is added to the table that

Table 8.1 EntryStatus Textual Convention

STATE	ENUMERATION	DESCRIPTION
valid	1	Row exists and is active. It is fully configured and operational
createRequest	2	Create a new row by creating this object
underCreation	3	Row is not fully active
invalid	4	Delete the row by disassociating the mapping of this entry

contains information on the status of the row. The *EntryStatus* data type can exist in one of four states: (1) *valid*, (2) *createRequest*, (3) *underCreation*, and (4) *invalid*. The four states of *EntryStatus* are shown in Table 8.1. Under the *valid* state condition, the instantiation or row of the table is operational and is probably measuring the number of input octets in the IF group on an interface. Any management system, which is authenticated to use the RMON device, may use this row of data. Of course, if the owner of the row decides to make it invalid, other systems lose the data. The *invalid* state is the way to delete a row. Based on implementation, the row may be immediately deleted and the resource claimed, or it may be done in a batch mode later. If the desired row of information does not already exist, the management system can create a row. The *EntryStatus* is then set to *createRequest*. The process of creation may involve more than one exchange of PDUs between the manager and the agent. In such a situation, the state of the *EntryStatus* is set to *underCreation* so that others won't use it. After the creation process is complete, it is set to the *valid* state.

8.3.2 RMON1 Groups and Functions

RMON in general, and RMON1 specifically, performs numerous functions at the data link layer. Figure 8.3 shows a pictorial representation of RMON1 groups and functions. The data-gathering modules, which are LAN probes, gather data from the remotely monitored network comprising Ethernet and token-ring LANs. The data can serve as inputs to five sets of functions. Three of those comprise monitoring of traffic statistics. The host and conversation statistics group deals with traffic data associated with the hosts, ranking of traffic for the top N hosts, and conversation between hosts. The group of statistical data associated with Ethernet LAN, namely Ethernet statistics and Ethernet history statistics, is addressed by the groups and functions in the Ethernet statistics box. The history control table controls the data to be gathered from various networks. It is also used by the token-ring statistics modules in the token-ring statistics box. Outputs of various modules are analyzed and presented in tabular and graphical forms to the user by the network manager in the NMS.

The filter group is a cascade of two filters. The packet filter filters incoming packets by performing a Boolean and/or XOR with a mask specified. This could be quite complex. The filtered packet stream is considered a channel. We can make further selections based on the channel mask. The filtered outputs may generate either alarms or events. These are reported to the network manager. The output of the data gatherer could also generate an alarm directly.

The output of the filter group could be stored in the packet capture module for further analysis by the network manager. This could be associated with a special study of the traffic pattern or troubleshooting of an abnormality in the network.

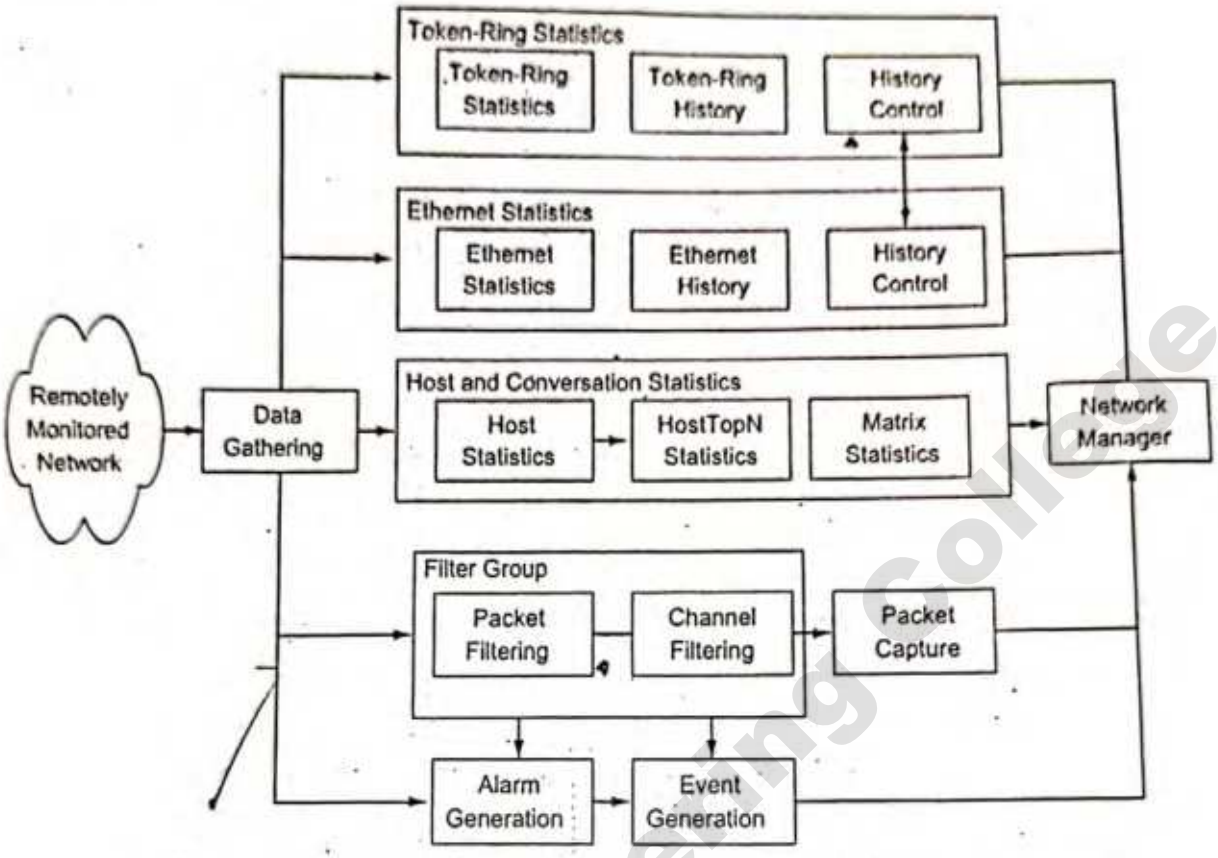


Figure 8.3 RMON1 Groups and Functions

The above functions associated with the various groups are accomplished using ten groups associated with the RMON1 MIB, as shown in Table 8.2. The first nine groups are applicable to common data and to Ethernet LAN, and the tenth group extends it to token-ring LAN. Most of the groups have one or more tables. The groups fall into three categories. The largest category is the statistics-gathering groups. These are the Statistics groups, the History groups, the Host group, the Host Top N group, and the Matrix group. The second category deals with the network event reporting functions. These are the Alarm group and the Event group. The third category deals with filtering the input packets according to selected criteria and capturing the data if desired for further analysis. These are the Filter group and the Packet Capture group. We will consider RMON1 groups and the token-ring extension to RMON1 in Sections 8.3.4 and 8.3.5, respectively.

In Table 8.2, we notice in the Tables column that some of the groups have tables with "2" as part of the name; for example, *etherStats2Table* in the Statistics group. These are additional tables created during RMON2 specifications development and are enhancements to RMON1. Hence, they are included here as part of RMON1. The enhancements to RMON1 include the standard *LastCreateTime* textual convention for all control tables and *TimeFilter* textual convention that provides capability for the filter to handle rows to be used for the index to a table. The *LastCreateTime* enhancement helps keep track of data with the changes in control. The *TimeFilter* enables an application to download only those rows that changed since a particular time. The agent returns a value only if the time mark is less than the last update time.

As an example, let us consider a *fooTable* with two rows and three columnar objects, *fooTimeMark* (with *TimeFilter* as the data type), *fooIndex*, and *foocounts*. The indices defining a row are *fooTimeMark*

Table 8.2 RMON1 MIB Groups and Tables

GROUP	OID	FUNCTION	TABLES
Statistics	rmon 1	Provides link-level statistics	–etherStatsTable –etherStats2Table –historyControlTable
History	rmon 2	Collects periodic statistical data and stores for later retrieval	–etherHistoryTable –historyControl2Table –etherHistory2Table
Alarm	rmon 3	Generates events when the data sample gathered crosses pre-established thresholds	–alarmTable
Host	rmon 4	Gathers statistical data on hosts	–hostControlTable –hostTable –hostTimeTable –hostControl2Table
Host Top N	rmon 5	Computes the top N hosts on the respective categories of statistics gathered	–hostTopNcontrolTable
Matrix	rmon 6	Gathers statistics on traffic between pairs of hosts	–matrixControlTable –matrixSDTable –matrixDSTable –matrixControl2Table
Filter	rmon 7	Performs filter function that enables capture of desired parameters	–filterTable
Packet capture	rmon 8	Provides packet capture capability to gather packets after they flow through a channel	–channelTable –filter2Table –channel2Table –buffercontrolTable
Event	rmon 9	Controls the generation of events and notifications	–captureBufferTable –eventTable
Token ring	Rmon 10	See Table 8.3	See Table 8.3

and *fooIndex*. Let the *TimeFilter* index start at 0, the last update of *fooCounter* in row #1 occur at time 3, and its value is 5. Assume the update to row #2 occurred at time 5 and the value was updated to 9. This scenario would yield the following instance of *fooCounts* in the *fooTable*:

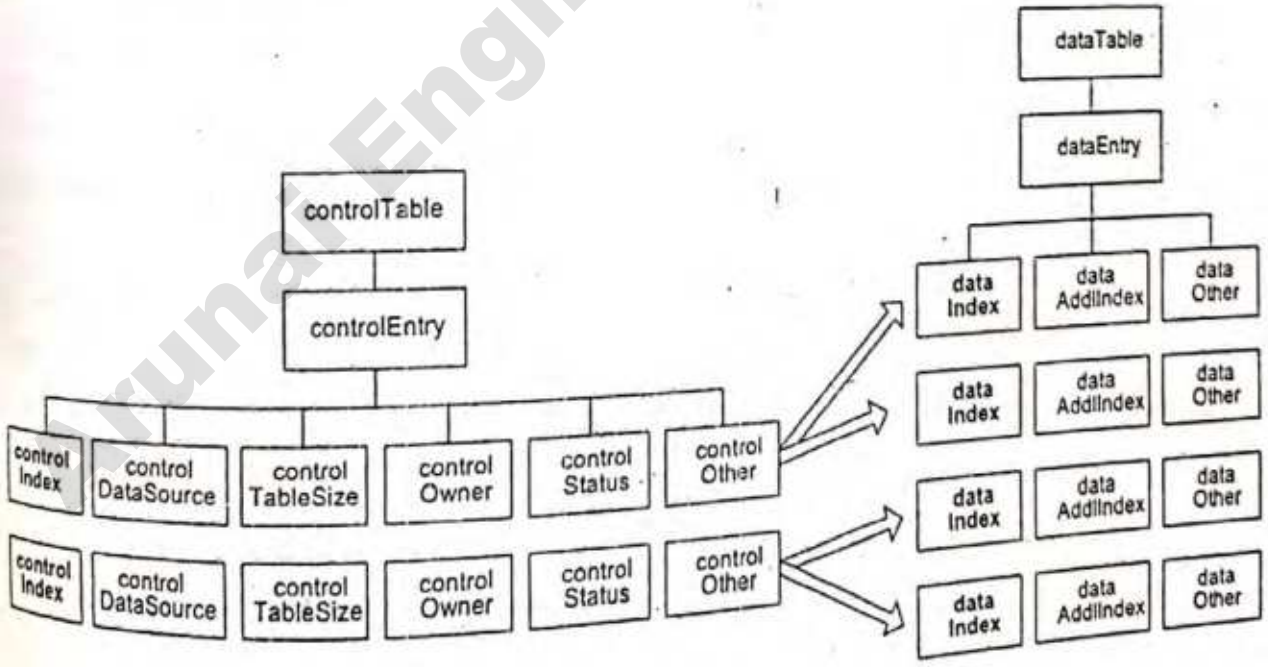
```
fooCounts.0.1   5
fooCounts.0.2   9
```

fooCounts.1.1 5
fooCounts.1.2 9
fooCounts.2.1 5
fooCounts.1.2 9
fooCounts.3.1 5
fooCounts.3.2 9
fooCounts.4.2 9 (Note that row #1 does not exist for times 4 and 5 since the last update occurred at timemark 3.)
fooCounts.5.2 9
(Both rows #1 and #2 do not exist for timemark greater than 5.)

Relationship Between Control and Data Tables

Observing the Tables column in Table 8.2, you will notice several of the groups have a data table and a control table. The data table contains rows (instances) of data. The control table defines the instances of the data rows in the data table and is settable to gather and store different instances of data. The relationship between the control table and the data table is illustrated in a generic manner in Figure 8.4. The value of the *dataIndex* in the data table is the same as the value of *controlIndex* in the control table.

Let us understand how the data table and the control table work together using the matrix group in Table 8.2. We can collect data based on source and destination addresses appearing in the packets on a given interface using the *matrixSDTable* (matrix source–destination table). The control index is an integer uniquely identifying the row in the control table. It would have a value of 1 for the first interface of a managed entity. The value of the columnar object, *controlDataSource*, identifies the source of the data that is being collected. In our example, if the interface #1 belongs to the interfaces group, then *controlDataSource* is *ifIndex.1*.



Note on Indices:
Indices marked in bold letter
Value of dataIndex same as the value of controlIndex

Figure 8.4 Relationship Between Control and Data Tables

The *controlTableSize* identifies entries associated with this data source. In our matrix source–destination table example, this would be the source–destination pair in each row of the table.

The *controlOwner* columnar object is the entity or person who created the entry. The entity could be either the agent or NMS, or a management person. The *controlStatus* is one of the entries listed in Table 8.1. The *controlOther* could be any other object.

To uniquely identify a conceptual row in the data table, we may need to specify more indices than the *dataIndex*. This is indicated as *dataAddlIndex* in Figure 8.4. In our matrix source–destination example, additional indices are source and destination address objects. The *dataOther* in the data table indicates data being collected, such as the number of packets.

8.3.4 RMON1 Common and Ethernet Groups

We have so far covered the global picture of RMON1 Ethernet MIB and how data and control tables are related to each other. Let us now address the nine RMON1 common and Ethernet groups.

Statistics Group. The statistics group contains statistics measured by the probe for each monitored Ethernet interface on a device. The *etherStatsTable* in this group has an entry for each interface. Data include statistics on packet types, size, and errors. It also provides capability to gather statistics on the collision of the Ethernet segment. The number of collisions is a best estimate, as the number of collisions detected depends on where the probe is placed on the segment.

The statistics group is used to measure live statistics on nodes and segments. Commercial NMSs include features such as dynamic presentation of various traffic patterns. The number of MIB collisions could also be used for alarm generation when it exceeds a set high threshold value.

History Group. The history group consists of two subgroups: the history control group and the history (data) group. The history control group controls the periodic statistical sampling of data from various types of networks. The control table stores configuration entries comprising interface, polling period, and other parameters. Information is stored in a media-specific table, the history table, which contains one entry for each specific sample. A short-term and a long-term interval, such as 30-second and 30-minute intervals, may be specified to obtain two different statistics. The data objects defined are dropped events, number of octets and packets, different type of errors, fragments, collisions, and utilization.

The history group is extremely useful in tracking the overall trend in the volume of traffic. Since historical data are accumulated at the data link layer, they include traffic caused by all higher-layer protocols. Short-term history statistics can also be used to troubleshoot network performance problems. For example, in one study of traffic pattern that the author participated in, short-term history statistics revealed that a significant volume of "transparent" data was contributed by servers in the network, which were functioning as "mirrors" for a public news service on the Internet. Although the service was considered to be desirable, since it was generated and consumed externally, it behaved somewhat transparently with regard to the local network traffic.

Alarm Group. The alarm group periodically takes statistical samples on specified variables in the probe and compares them with the pre-configured threshold stored in the probe. Whenever the monitored variable crosses the threshold, an event is generated. To avoid excessive generation of events on the threshold border, rising and falling thresholds are specified. This works in the following manner. Suppose an alarm event is generated when the variable crosses the falling threshold while going down in value. Another event would be generated only after the value crosses the rising threshold at least once.

The group contains an *alarm table* that has a list of entries defining the alarm parameters. The columnar objects *alarmVariable* and *alarmInterval* are used to select the variable and the sampling interval. The sampling type is either the absolute or delta value. In the former, the absolute value of the variable at the end of the previous period is stored as an alarm value. In the latter type, the absolute value at the end of a period is subtracted from the beginning of the period and the computed value is stored. These values are compared with the rising and falling thresholds to generate alarms.

An example of an absolute value would be a new interface card on test for infant mortality. The threshold of the sum of outgoing and incoming packets could be set to 1 gigaoctects and the RMON would generate an alarm/event when the threshold is reached. An example of delta type is threshold set to 10,000 packets in a 10-second interval for excessive packet loss.

Host Group. The host group contains information about the hosts on the network. It compiles the list of hosts by looking at the good packets traversing the network and extracting the source and destination MAC addresses. It maintains statistics on these hosts. There are three tables in the group: *hostControlTable*, *hostTable*, and *hostTimeTable*. The *hostControlTable* controls the interfaces on which data gathering is done. The other two tables depend on this information. The *hostTable* contains statistics about the host. The *hostTimeTable* contains the same data as the host table, but is stored in the time order in which the host entry was discovered. This helps in the fast discovery of new hosts in the system. The entries in the two data tables are synchronized with respect to the host in the *hostControlTable*. We can obtain statistics on a host using this MIB.

Host Top N Group. The host top N group performs a report-generation function for ranking the top N hosts in the category of the selected statistics. For example, we can rank-order the top ten hosts with maximum outgoing traffic. The *HostTopNControlTable* is used to initiate generation of such a report.

As an example of the type of data that can be acquired using an RMON probe, Figure 8.5 shows a chart derived using an RMON probe for the output octets of the top ten hosts in a network. The names of the hosts have been changed to generic host numbers for security reasons.

Matrix Group. The matrix group stores statistics on the conversation between pairs of hosts. An entry is created for each conversation that the probe detects. There are three tables in the group. The *matrixControlTable* controls the information to be gathered. The *matrixSDTable* keeps track of the source to destination conversations; and the *matrixDSTable* keeps data based on destination to source traffic. We can obtain a graph similar to Figure 8.5 for the conversation pairs in both directions using this group.

Filter Group. The filter group is used to filter packets to be captured based on logical expressions. The stream of data based on a logical expression is called a "channel." The group contains a filter table and a channel table. The filter table allows packets to be filtered with an arbitrary filter expression, a set of filters associated with each channel. Each filter is defined by a row in the filter table. A channel may be associated with several rows. For each channel, the input packet is validated against each filter associated with that channel and is accepted if it passes any of the tests. A row in the channel table of the filter group includes the interface ID (same as *ifIndex*) with which the channel is associated, along with acceptance criteria. The combination of the filter and channel filtering provides enormous flexibility to select packets to be captured.

Packet Capture Group. The packet capture group is a post-filter group. It captures packets from each channel based on the filter criteria of packet and channel filters in the filter group. The channel filter criteria for acceptance of the filter group output are controlled by the *bufferControlTable* and

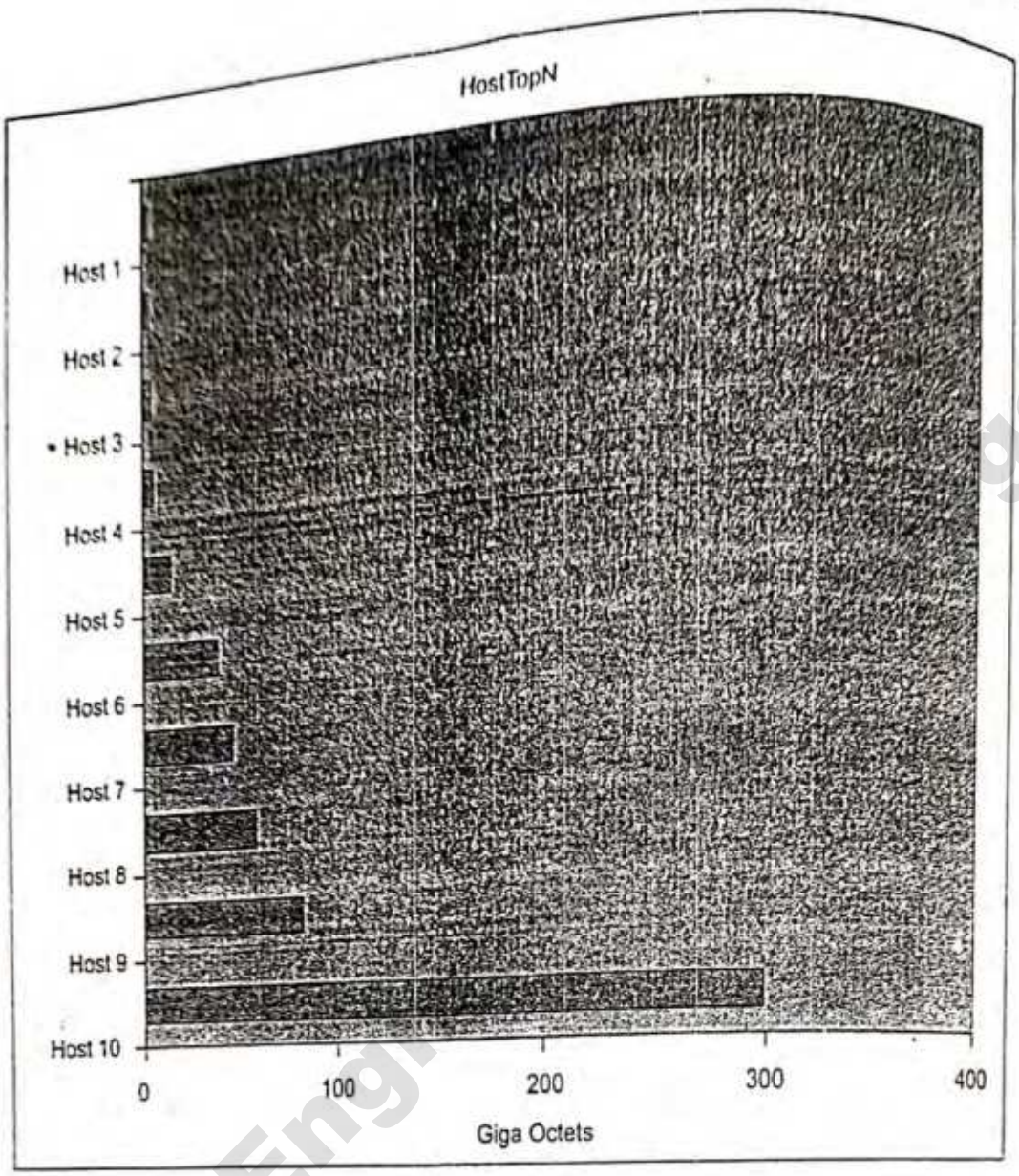


Figure 8.5 HostTop-10 Output Octets

the captured channel data in the *captureBufferTable*. Each packet captured is stored in the buffer as an instance.

Event Group. The event group controls the generation and notification of events. Both the rising alarm and the falling alarm can be specified in the *eventTable* associated with the group. Besides the transmittal of events, a log is maintained in the system.

8.3.5 RMON Token-Ring Extension Groups

As we mentioned earlier, token-ring RMON MIB is an extension to RMON1 MIB and is specified in RFC 1513. Table 8.3 presents the token-ring MIB groups and tables. There are eight groups, each with a data table and two with control tables.

There are two token-ring statistics groups, one at the MAC layer (token-ring statistics group) and a second on packets collected promiscuously (token-ring promiscuous statistics group). They both contain statistics on ring utilization and ring error statistics. The MAC-layer statistics group collects data on token-ring parameters such as token packets, errors in packets, bursts, polling, etc. The promiscuous

Table 8.3 RMON Token-Ring MIB Groups and Tables

TOKEN RING GROUP	FUNCTION	TABLES
Statistics	Current utilization and error statistics of MAC Layer	tokenRingMLStatsTable
		tokenRingMLStats2Table
Promiscuous statistics	Current utilization and error statistics of promiscuous data	tokenRingPStatsTable
		tokenRingPStats2Table
MAC-layer history	Historical utilization and error statistics of MAC layer	tokenRingMLHistoryTable
Promiscuous history	Historical utilization and error statistics of promiscuous data	tokenRingPHistoryTable
Ring station	Station statistics	ringStationControlTable
		ringStationTable
		ringStationControl2Table
Ring station order	Order of the stations	ringStationOrderTable
Ring station configuration	Active configuration of ring stations	ringStationConfigControlTable
		ringStationConfigTable
Source-routing	Utilization statistics of source routing information	sourceRoutingStatsTable
		sourceRoutingStats2Table

statistics group addresses statistics on the number of packets of various sizes and the type of packets as to data—multicast or broadcast. There are two corresponding history statistics groups—current and promiscuous. Each of the four statistics groups has one data table associated with it.

There are three groups associated with the stations on the ring. The ring station group provides statistics on each station being monitored on the ring along with its status. The data are stored in the *ringStationTable*. The rings and parameters to be monitored are controlled by the *ringStationControlTable*. The ring station order group provides the order of the station on the monitored rings and has only a data table. The ring station configuration group manages the stations on the ring.

The last group in the ring groups is the source-routing group. It is used to gather statistics on routing information in a pure source-routing environment.

8.4 RMON2

RMON1 dealt primarily with data associated with the OSI data link layer. The success and popularity of RMON1 led to the development of RMON2. RMON2 [RFC 2021] extends the monitoring capability to the upper layers, from the network layer to the application layer. The term application level is used in the SNMP RMON concept to describe a class of protocols, and not strictly the OSI layer 7 protocol. The error statistics in any layer include all errors below the layer, down to the network layer. For example, the network layer errors do not include data link layer errors, but the transport layer errors include the network layer errors.

Several of the groups and functions in RMON2 at higher layers are similar to that of the data link layer in RMON1. We will discuss the groups and their similarity here. We will cover in detail how protocol analyzer systems incorporate the higher-layer data gathered using RMON2 in Chapter 9 on NMSs and tools.

RMON2 Management Information Base

The architecture of RMON2 is the same as RMON1. RMON2 MIB is arranged into ten groups. Table 8.4 shows the RMON2 MIB groups and tables. We have already discussed enhancements to RMON1 MIB in the previous section.

The protocol directory group is an inventory of the protocols that the probe can monitor. The capability of the probe can be altered by reconfiguring the *protocolDirTable*. The protocols range from the data link control layer to the application layer. This is identified by the columnar object on the unique protocol ID. Each protocol is further subdivided based on parameters, such as fragments. The protocol identifier and protocol parameters are used as indices for the rows of the table. There is one entry in the table for each protocol. The protocols that can be used with the protocol directory have been defined in RFC 2074.

The protocol distribution group provides information on the relative traffic of different protocols either in octets or packets. It collects very basic statistics that would help a NMS manage bandwidth allocation utilized by different protocols. The *protocolDistControlTable* is configured according to the data to be collected and *protocolDistStatsTable* stores the data collected. Each row in the *protocolDistStatsTable* is indexed by the *protocolDistControlIndex* in the *protocolDistControlTable* and *protocolDirLocalIndex* in the *protocolDirTable*. The data table stores the packet and octet counts.

The address map group is similar to the address translation table binding the MAC address to the network address on each interface. It has two tables for control and data.

The network-layer host group measures traffic sent from and to each network address representing each host discovered by the probe, as the host group in RMON1 does.

The network-layer matrix group provides information on the conversation between pairs of hosts in both directions. It is very similar to the matrix tables in RMON1. The group also ranks the top N conversations. It has two control tables and three data tables.

The application layer functions are grouped into two groups, the application-layer host group and the application-layer matrix group. They both calculate traffic by protocol units and use their respective control tables in the network-layer host group and the network-layer matrix group. The application-layer matrix group can also generate a report of the top N protocol conversations.

Alarm and history group information have been combined into the user history collection group in RMON2. This function, normally done by NMSs, can be off-loaded to RMON. It has two control tables and one data table. Data objects are collected in bucket groups. Each bucket group pertains to a MIB object, and the elements in the group are the instances of the MIB object. Users can specify the data to be collected by entering data into *usrHistoryControlTable*, which will then be assembled with rows of instances in the *usrHistoryObjectTable*. Each row in the former specifies the number of buckets to be allocated for each object, and the latter contains rows of instances of the MIB object. The data are stored in *userHistoryTable*. There could be one or more instances of *userHistoryTable* associated with each *usrHistoryObjectTable*.

Table 8.4 RMON2 MIB Groups and Tables

GROUP	OID	FUNCTION	TABLES
Protocol directory	rmon 11	Inventory of protocols	protocolDirTable
Protocol distribution	rmon 12	Relative statistics on octets and packets	protocolDistControlTable
			protocolDistStatsTable
Address map	rmon 13	MAC address to network address on the interfaces	addressMapControlTable
			addressMapTable
Network-layer host	rmon 14	Traffic data from and to each host	n1HostControlTable
			n1HostTable
Network-layer matrix	rmon 15	Traffic data from each pair of hosts	n1MatrixControlTable
			n1MatrixSDTable
			n1MatrixDSTable
			n1MatrixTopNControlTable
			n1MatrixTopNTable
Application-layer host	rmon 16	Traffic data by protocol from and to each host	a1HostTable
Application-layer matrix	rmon 17	Traffic data by protocol between pairs of hosts	a1MatrixSDTable
			a1MatrixDSTable
			a1MatrixTopNControlTable
			a1MatrixTopNTable
User history collection	rmon 18	User-specified historical data on alarms and statistics	usrHistoryControlTable
			usrHistoryObjectTable
			usrHistoryTable
Probe configuration	rmon 19	Configuration of probe parameters	serialConfigTable
			netConfigTable
			trapDestTable
			serialConnectionTable
RMON conformance	rmon 20	RMON2 MIB compliances and compliance groups	See Section 8.4.2

The probe configuration group provides the facility to configure the probe. The data can be accessed using a modem connection. The pertinent data are stored in the *serialConfigTable* and *serialConnectionTable*. The *netConfigTable* contains the network configuration parameters, and the *trapDestTable* defines the destination addresses for the traps.

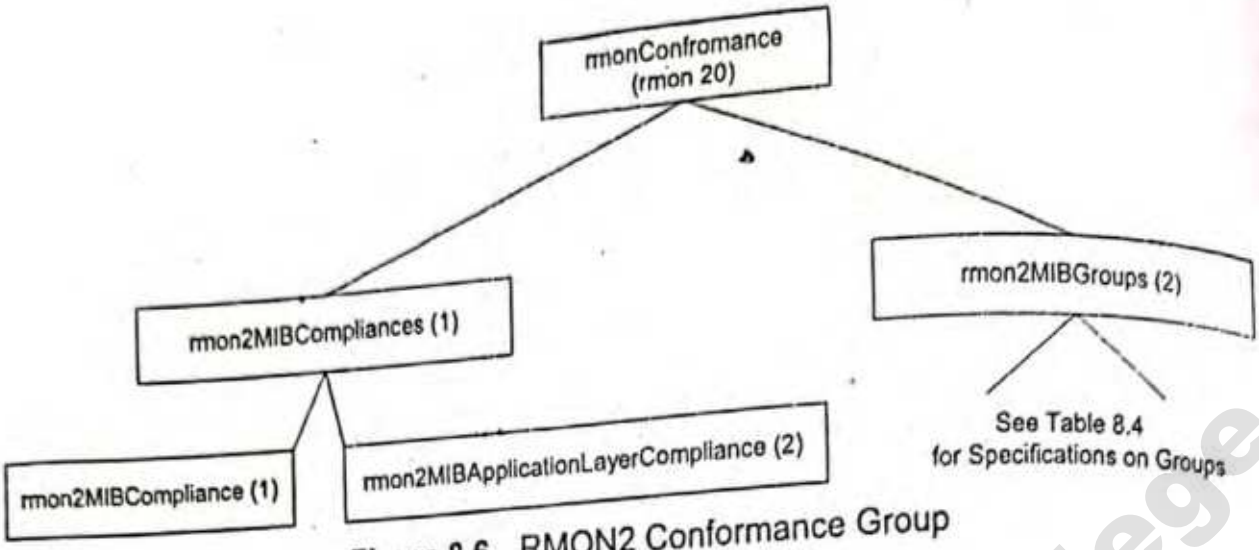


Figure 8.6 RMON2 Conformance Group

8.4.2 RMON2 Conformance Specifications

Conformance specifications were not specified in RMON1. They have been added in RMON2. As shown in Figure 8.6, the RMON2 conformance group consists of two subgroups, rmon2MIBCompliances and rmon2MIBGroups. The compliance requirements are separated into basic RMON2 MIB compliance and application layer RMON2 MIB compliance. Each compliance module defines the mandatory and optional groups. Vendors are required to implement the mandatory groups for compliance; optional groups may be used by vendors to specify additional capabilities.

There are 13 groups in rmon2MIBGroups. They are listed in Table 8.5 along with the mandatory (M) and optional (O) requirements for the basic- and application-level conformance to RMON2. The *rmon1EnhancementGroup* is mandatory for systems that implement RMON1 with RMON2. Notice that *probeConfigurationGroup* is a basic group and hence marked as mandatory, even though it is not specified as such in RFC 2021 definitions. The *rmon1EnhancementGroup* is mandatory for implementation of RMON1. The *rmon1EthernetEnhancementGroup* and *rmon1TokenRingEnhancementGroup* add enhancements to RMON1 that help management stations. The enhancements include filter entry, which provides variable-length offsets into packets and the addition of more statistical parameters.

8.5 ATM REMOTE MONITORING

We will be learning management of ATM in Chapter 9. However, there is a similarity in the use of remote probes for RMON on an ATM network. We will address the commonality and differences here. You may skip this section now, if you so choose, and return to it after you have studied ATM management.

We have thus far learned about RMON and its advantages for gathering statistics on Ethernet and token-ring LANs. RMON1 dealt with the data link layer and RMON2 with higher-level layers. IETF RMON MIBs have been extended to perform traffic monitoring and analysis for ATM networks (see af-nm-test-0080.000 in Table 9.3). Figure 8.7 shows an RMON MIB framework for the extensions, as portrayed by the ATM Forum. Switch extensions for RMON and ATM RMON define RMON objects at the "base" layer, which is the ATM sublayer level. ATM protocol IDs for RMON2 define additional objects needed at the higher-level layers [RFC 2074].

There are several differences between RMON of Ethernet and token ring and monitoring of ATM devices. Extending RMON to ATM requires design changes and new functionality. Particular attention needs to be paid to the following issues: high speed, cell vs. frames, and connection-oriented nature of