CHAPTER 28

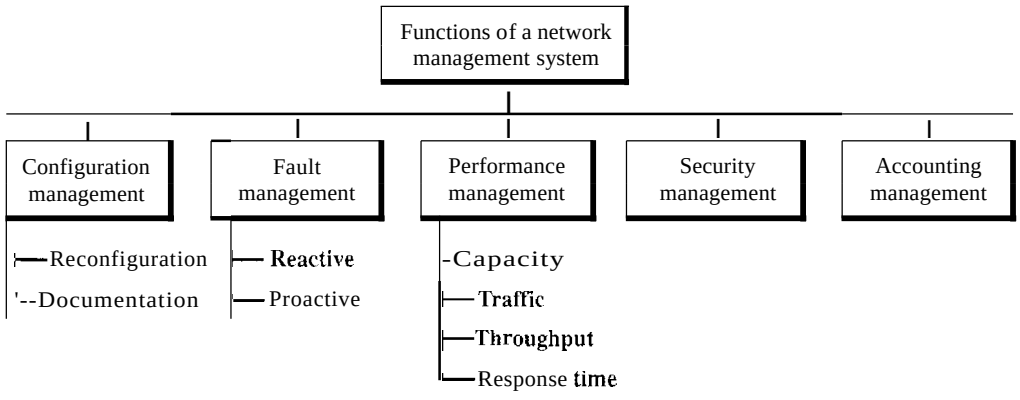
Network Management: SNMP

We can define **network management** as monitoring, testing, configuring, and troubleshooting network components to meet a set of requirements defined by an organization. These requirements include the smooth, efficient operation of the network that provides the predefined quality of service for users. To accomplish this task, a network management system uses hardware, software, and humans. In this chapter, first we briefly discuss the functions of a network management system. Then we concentrate on the most common management system, the Simple Network Management Protocol (SNMP).

28.1 NETWORK MANAGEMENT SYSTEM

We can say that the functions performed by a network management system can be divided into five broad categories: configuration management, fault management, performance management, security management, and accounting management, as shown in Figure 28.1.

Figure 28.1 *Functions of a network management system*



Configuration Management

A large network is usually made up of hundreds of entities that are physically or logically connected to one another. These entities have an initial configuration when the network is set up, but can change with time. Desktop computers may be replaced by others; application software may be updated to a newer version; and users may move from one group to another. The **configuration management** system must know, at any time, the status of each entity and its relation to other entities. Configuration management can be divided into two subsystems: reconfiguration and documentation.

Reconfiguration

Reconfiguration, which means adjusting the network components and features, can be a daily occurrence in a large network. There are three types of reconfiguration: hardware reconfiguration, software reconfiguration, and user-account reconfiguration.

Hardware reconfiguration covers all changes to the hardware. For example, a desktop computer may need to be replaced. A router may need to be moved to another part of the network. A subnetwork may be added or removed from the network. All these need the time and attention of network management. In a large network, there must be specialized personnel trained for quick and efficient hardware reconfiguration. Unfortunately, this type of reconfiguration cannot be automated and must be manually handled case by case.

Software reconfiguration covers all changes to the software. For example, new software may need to be installed on servers or clients. An operating system may need updating. Fortunately, most software reconfiguration can be automated. For example, updating an application on some or all clients can be electronically downloaded from the server.

User-account reconfiguration is not simply adding or deleting users on a system. You must also consider the user privileges, both as an individual and as a member of a group. For example, a user may have read and write permission with regard to some files, but only read permission with regard to other files. User-account reconfiguration can be, to some extent, automated. For example, in a college or university, at the beginning of each quarter or semester, new students are added to the system. The students are normally grouped according to the courses they take or the majors they pursue.

Documentation

The original network configuration and each subsequent change must be recorded meticulously. This means that there must be documentation for hardware, software, and user accounts.

Hardware documentation normally involves two sets of documents: maps and specifications. Maps track each piece of hardware and its connection to the network. There can be one general map that shows the logical relationship between each subnetwork. There can also be a second general map that shows the physical location of each subnetwork. For each subnetwork, then, there is one or more maps that show all pieces of equipment. The maps use some kind of standardization to be easily read and understood by current and future personnel. Maps are not enough per se. Each piece of hardware also needs to be documented. There must be a set of specifications for each piece

of hardware connected to the network. These specifications must include information such as hardware type, serial number, vendor (address and phone number), time of purchase, and warranty information.

All software must also be documented. Software documentation includes information such as the software type, the version, the time installed, and the license agreement.

Most operating systems have a utility that allows the documentation of user accounts and their privileges. The management must make sure that the files with this information are updated and secured. Some operating systems record access privileges in two documents; one shows all files and access types for each user; the other shows the list of users that have a particular access to a file.

Fault Management

Complex networks today are made up of hundreds and sometimes thousands of components. Proper operation of the network depends on the proper operation of each component individually and in relation to each other. **Fault management** is the area of network management that handles this issue.

An effective fault management system has two subsystems: reactive fault management and proactive fault management.

Reactive Fault Management

A reactive fault management system is responsible for detecting, isolating, correcting, and recording faults. It handles short-term solutions to faults.

The first step taken by a reactive fault management system is to detect the exact location of the fault. A fault is defined as an abnormal condition in the system. When a fault occurs, either the system stops working properly or the system creates excessive errors. A good example of a fault is a damaged communication medium. This fault may interrupt communication or produce excessive errors.

The next step taken by a reactive fault management system is to isolate the fault. A fault, if isolated, usually affects only a few users. After isolation, the affected users are immediately notified and given an estimated time of correction.

The third step is to correct the fault. This may involve replacing or repairing the faulty component(s).

After the fault is corrected, it must be documented. The record should show the exact location of the fault, the possible cause, the action or actions taken to correct the fault, the cost, and time it took for each step. Documentation is extremely important for several reasons:

- The problem may recur. Documentation can help the present or future administrator or technician solve a similar problem.
- The frequency of the same kind of failure is an indication of a major problem in the system. If a fault happens frequently in one component, it should be replaced with a similar one, or the whole system should be changed to avoid the use of that type of component.
- The statistic is helpful to another part of network management, performance management.

Proactive Fault Management

Proactive fault management tries to prevent faults from occurring. Although this is not always possible, some types of failures can be predicted and prevented. For example, if a manufacturer specifies a lifetime for a component or a part of a component, it is a good strategy to replace it before that time. As another example, if a fault happens frequently at one particular point of a network, it is wise to carefully reconfigure the network to prevent the fault from happening again.

Performance Management

Performance management, which is closely related to fault management, tries to monitor and control the network to ensure that it is running as efficiently as possible. Performance management tries to quantify performance by using some measurable quantity such as capacity, traffic, throughput, or response time.

Capacity

One factor that must be monitored by a performance management system is the capacity of the network. Every network has a limited capacity, and the performance management system must ensure that it is not used above this capacity. For example, if a LAN is designed for 100 stations at an average data rate of 2 Mbps, it will not operate properly if 200 stations are connected to the network. The data rate will decrease and blocking may occur.

Traffic

Traffic can be measured in two ways: internally and externally. Internal traffic is measured by the number of packets (or bytes) traveling inside the network. External traffic is measured by the exchange of packets (or bytes) outside the network. During peak hours, when the system is heavily used, blocking may occur if there is excessive traffic.

Throughput

We can measure the throughput of an individual device (such as a router) or a part of the network. Performance management monitors the throughput to make sure that it is not reduced to unacceptable levels.

Response Time

Response time is normally measured from the time a user requests a service to the time the service is granted. Other factors such as capacity and traffic can affect the response time. Performance management monitors the average response time and the peak-hour response time. Any increase in response time is a very serious condition as it is an indication that the network is working above its capacity.

Security Management

Security management is responsible for controlling access to the network based on the predefined policy. We discuss security and in particular network security in Chapters 31 and 32.

Accounting Management

Accounting management is the control of users' access to network resources through charges. Under accounting management, individual users, departments, divisions, or even projects are charged for the services they receive from the network. Charging does not necessarily mean cash transfer; it may mean debiting the departments or divisions for budgeting purposes. Today, organizations use an accounting management system for the following reasons:

- It prevents users from monopolizing limited network resources.
- It prevents users from using the system inefficiently.
- Network managers can do short- and long-term planning based on the demand for network use.

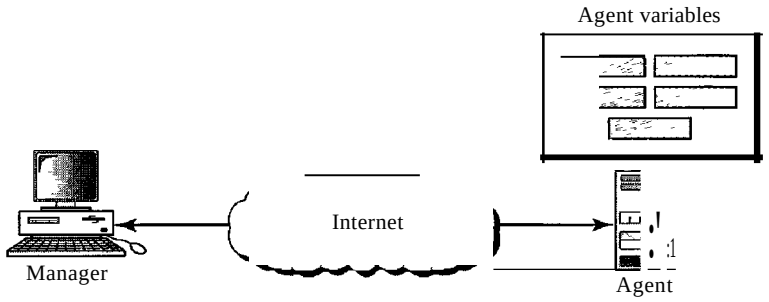
28.2 SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

The Simple Network Management Protocol (SNMP) is a framework for managing devices in an internet using the TCPIIP protocol suite. It provides a set of fundamental operations for monitoring and maintaining an internet.

Concept

SNMP uses the concept of manager and agent. That is, a manager, usually a host, controls and monitors a set of agents, usually routers (see Figure 28.2).

Figure 28.2 *SNMP concept*



SNMP is an application-level protocol in which a few manager stations control a set of agents. The protocol is designed at the application level so that it can monitor devices made by different manufacturers and installed on different physical networks. In other words, SNMP frees management tasks from both the physical characteristics of the managed devices and the underlying networking technology. It can be used in a heterogeneous internet made of different LANs and WANs connected by routers made by different manufacturers.

Managers and Agents

A management station, called a manager, is a host that runs the SNMP client program. A managed station, called an agent, is a router (or a host) that runs the SNMP server program. Management is achieved through simple interaction between a manager and an agent.

The agent keeps performance information in a database. The manager has access to the values in the database. For example, a router can store in appropriate variables the number of packets received and forwarded. The manager can fetch and compare the values of these two variables to see if the router is congested or not.

The manager can also make the router perform certain actions. For example, a router periodically checks the value of a reboot counter to see when it should reboot itself. It reboots itself, for example, if the value of the counter is 0. The manager can use this feature to reboot the agent remotely at any time. It simply sends a packet to force a 0 value in the counter.

Agents can also contribute to the management process. The server program running on the agent can check the environment, and if it notices something unusual, it can send a warning message, called a trap, to the manager.

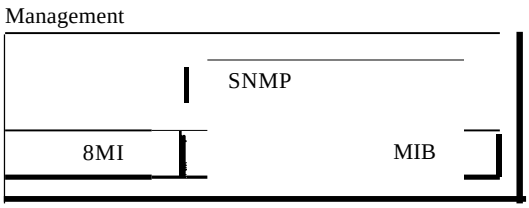
In other words, management with SNMP is based on three basic ideas:

1. A manager checks an agent by requesting information that reflects the behavior of the agent.
2. A manager forces an agent to perform a task by resetting values in the agent database.
3. An agent contributes to the management process by warning the manager of an unusual situation.

Management Components

To do management tasks, SNMP uses two other protocols: Structure of Management Information (SMI) and Management Information Base (MIB). In other words, management on the Internet is done through the cooperation of the three protocols SNMP, SMI, and MIB, as shown in Figure 28.3.

Figure 28.3 *Components of network management on the Internet*



Let us elaborate on the interactions between these protocols.

Role of SNMP

SNMP has some very specific roles in network management. It defines the format of the packet to be sent from a manager to an agent and vice versa. It also interprets the

result and creates statistics (often with the help of other management software). The packets exchanged contain the object (variable) names and their status (values). SNMP is responsible for reading and changing these values.

SNMP defines the format of packets exchanged between a manager and an agent. It reads and changes the status (values) of objects (variables) in 8NMP packets.

Role of SMI

To use SNMP, we need rules. We need rules for naming objects. This is particularly important because the objects in SNMP form a hierarchical structure (an object may have a parent object and some children objects). Part of a name can be inherited from the parent. We also need rules to define the type of the objects. What types of objects are handled by SNMP? Can SNMP handle simple types or structured types? How many simple types are available? What are the sizes of these types? What is the range of these types? In addition, how are each of these types encoded?

We need these universal rules because we do not know the architecture of the computers that send, receive, or store these values. The sender may be a powerful computer in which an integer is stored as 8-byte data; the receiver may be a small computer that stores an integer as 4-byte data.

SMI is a protocol that defines these rules. However, we must understand that SMI only defines the rules; it does not define how many objects are managed in an entity or which object uses which type. SMI is a collection of general rules to name objects and to list their types. The association of an object with the type is not done by SMI.

8MI defines the general rules for naming objects, defining object types (including range and length), and showing how to encode objects and values.

8M1 does not define the number of objects an entity should manage or name the objects to be managed or define the association between the objects and their values.

Role of MIB

We hope it is clear that we need another protocol. For each entity to be managed, this protocol must define the number of objects, name them according to the rules defined by SMI, and associate a type to each named object. This protocol is MIB. MIB creates a set of objects defined for each entity similar to a database (mostly metadata in a database, names and types without values).

MIB creates a collection of named objects, their types, and their relationships to each other in an entity to be managed.

An Analogy

Before discussing each of these protocols in greater detail, we give an analogy. The three network management components are similar to what we need when we write a program in a computer language to solve a problem.

Before we write a program, the syntax of the language (such as C or Java) must be predefined. The language also defines the structure of variables (simple, structured, pointer, and so on) and how the variables must be named. For example, a variable name must be 1 to *N* characters in length and start with a letter followed by alphanumeric characters. The language also defines the type of data to be used (integer, float, char, etc.). In programming the rules are defined by the language. In network management the rules are defined by SMI.

Most computer languages require that variables be declared in each specific program. The declaration names each variable and defines the predefined type. For example, if a program has two variables (an integer named *counter* and an array named *grades* of type char), they must be declared at the beginning of the program:

```
int counter;
char grades [40];
```

Note that the declarations name the variables (counter and grades) and define the type of each variable. Because the types are predefined in the language, the program knows the range and size of each variable.

MIB does this task in network management. MIB names each object and defines the type of the objects. Because the type is defined by SMI, SNMP knows the range and size.

After declaration in programming, the program needs to write statements to store values in the variables and change them if needed. SNMP does this task in network management. SNMP stores, changes, and interprets the values of objects already declared by MIB according to the rules defined by SMI.

We can compare the task of network management to the task of writing a program.

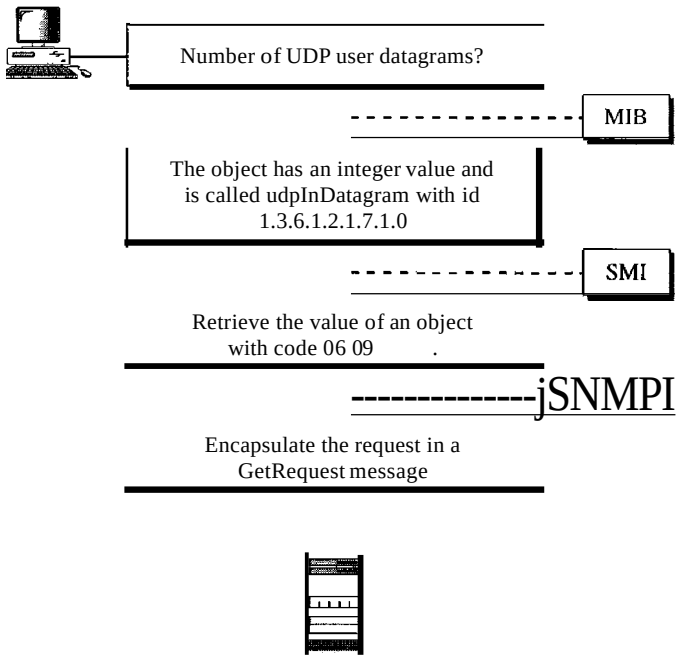
- Both tasks need rules. In network management this is handled by SMI.
- Both tasks need variable declarations. In network management this is handled by MIB.
- Both tasks have actions performed by statements. In network management this is handled by SNMP.

An Overview

Before discussing each component in detail, we show how each is involved in a simple scenario. This is an overview that will be developed later at the end of the chapter. A manager station (SNMP client) wants to send a message to an agent station (SNMP server) to find the number of UDP user datagrams received by the agent. Figure 28.4 shows an overview of steps involved.

MIB is responsible for finding the object that holds the number of the UDP user datagrams received. SMI, with the help of another embedded protocol, is responsible for encoding the name of the object. SNMP is responsible for creating a message, called a GetRequest message, and encapsulating the encoded message. Of course, things are more complicated than this simple overview, but we first need more details of each protocol.

Figure 28.4 *Management overview*



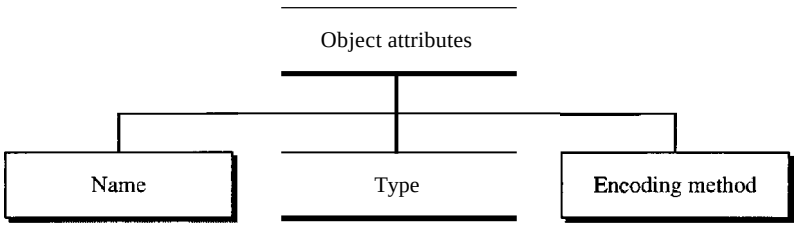
Structure of Management Information

The Structure of Management Information, version 2 (SMIv2) is a component for network management. Its functions are

1. To name objects
2. To define the type of data that can be stored in an object
3. To show how to encode data for transmission over the network

SMI is a guideline for SNMP. It emphasizes three attributes to handle an object: name, data type, and encoding method (see Figure 28.5).

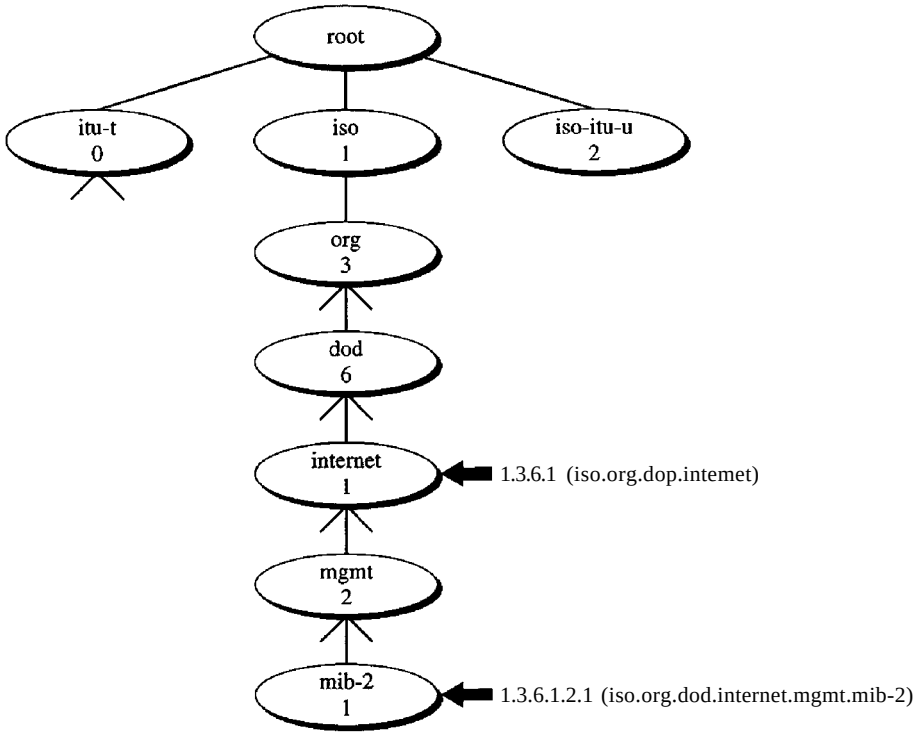
Figure 28.5 *Object attributes*



Name

SMI requires that each managed object (such as a router, a variable in a router, a value) have a unique name. To name objects globally, SMI uses an object identifier, which is a hierarchical identifier based on a tree structure (see Figure 28.6).

Figure 28.6 *Object identifier*



The tree structure starts with an unnamed root. Each object can be defined by using a sequence of integers separated by dots. The tree structure can also define an object by using a sequence of textual names separated by dots. The integer-dot representation is used in SNMP. The name-dot notation is used by people. For example, the following shows the same object in two different notations:

iso.org.dod.internet.mgmt.mib-2 ➡ 1.3.6.1.2.1

The objects that are used in SNMP are located under the *mib-2* object, so their identifiers always start with 1.3.6.1.2.1.

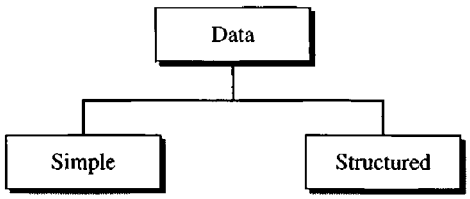
AU objects managed by SNMP are given an object identifier.
The object identifier always starts with 1.3.6.1.2.1.

Type

The second attribute of an object is the type of data stored in it. To define the data type, SMI uses fundamental Abstract Syntax Notation 1 (ASN.l) definitions and adds some new definitions. In other words, SMI is both a subset and a superset of ASN.1.

SMI has two broad categories of data type: *simple* and *structured.* We first define the simple types and then show how the structured types can be constructed from the simple ones (see Figure 28.7).

Figure 28.7 *Data type*



Simple Type The simple data types are atomic data types. Some of them are taken directly from ASN.l; others are added by SMI. The most important ones are given in Table 28.1. The first five are from ASN.l; the next seven are defined by SMI.

Table 28.1 *Data types*

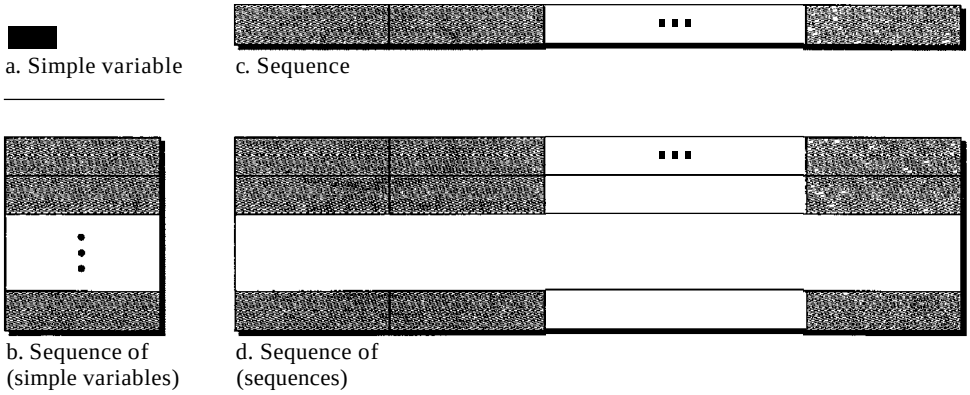
Type	*Size*	*Description*
INTEGER	4 bytes	An integer with a value between -2^{31} and $2^{31} - 1$
Integer32	4 bytes	Same as INTEGER
Unsigned32	4 bytes	Unsigned with a value between 0 and $2^{32} - 1$
OCTET STRING	Variable	Byte string up to 65,535 bytes long
OBJECT IDENTIFIER	Variable	An object identifier
IPAddress	4 bytes	An IP address made of four integers
Counter32	4 bytes	An integer whose value can be incremented from 0 to 2^{32}; when it reaches its maximum value, it wraps back to 0.
Counter64	8 bytes	64-bit counter
Gauge32	4 bytes	Same as Counter32, but when it reaches its maximum value, it does not wrap; it remains there until it is reset
TimeTicks	4 bytes	A counting value that records time in $\frac{1}{100}$ s
BITS		A string of bits
Opaque	Variable	Uninterpreted string

Structured Type By combining simple and structured data types, we can make new structured data types. SMI defines two structured data types: *sequence* and *sequence of*

- Sequence. A *sequence* data type is a combination of simple data types, not necessarily of the same type. It is analogous to the concept of a *struct* or a *record* used in programming languages such as C.
- Sequence of. A *sequence of* data type is a combination of simple data types all of the same type or a combination of sequence data types all of the same type. It is analogous to the concept of an *array* used in programming languages such as C.

Figure 28.8 shows a conceptual view of data types.

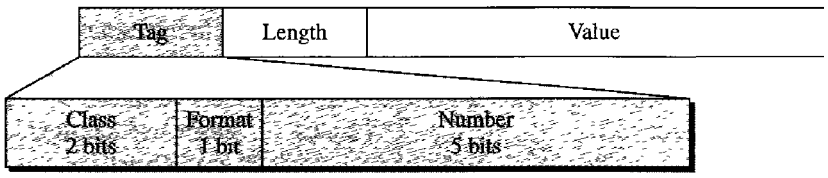
Figure 28.8 *Conceptual data types*



Encoding Method

SMI uses another standard, Basic Encoding Rules (BER), to encode data to be transmitted over the network. BER specifies that each piece of data be encoded in triplet format: tag, length, and value, as illustrated in Figure 28.9.

Figure 28.9 *Encoding format*



- Tag. The tag is a I-byte field that defines the type of data. It is composed of three subfields: *class* (2 *bits),fonnat* (1 bit), and *number* (5 bits). The class subfield defines the scope of the data. Four classes are defined: universal (00), applicationwide (01), context-specific (10), and private (11). The universal data types are those taken from ASN.l (INTEGER, OCTET STRING, and ObjectIdentifier). The applicationwide data types are those added by SMI (IPAddress, Counter, Gauge, and TimeTicks). The five context-specific data types have meanings that may change from one protocol to another. The private data types are vendor-specific.

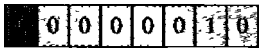
  The format subfield indicates whether the data are simple (0) or structured (1). The number subfield further divides simple or structured data into subgroups. For example, in the universal class, with simple format, INTEGER has a value of 2, OCTET STRING has a value of 4, and so on. Table 28.2 shows the data types we use in this chapter and their tags in binary and hexadecimal numbers.

Table 28.2 *Codes for data types*

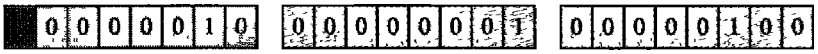
Data Type	*Class*	*Format*	*Number*	*Tag (Binary)*	*Tag (Hex)*
INTEGER	00	0	00010	00000010	02
OCTET STRING	00	0	00100	00000100	04
OBJECT IDENTIFIER	00	0	00110	00000110	06
NULL	00	0	00101	00000101	05
Sequence, sequence of	00	1	10000	00110000	30
IPAddress	01	0	00000	01000000	40
Counter	01	0	00001	01000001	41
Gauge	01	0	00010	01000010	42
TimeTicks	01	0	00011	01000011	43
Opaque	01	0	00100	01000100	44

- Length. The length field is 1 or more bytes. If it is 1 byte, the most significant bit must be 0. The other 7 bits define the length of the data. If it is more than 1 byte, the most significant bit of the first byte must be 1. The other 7 bits of the first byte define the number of bytes needed to define the length. See Figure 28.10 for a depiction of the length field.

Figure 28.10 *Length format*



a. The colored part defines the length (2).



b. The shaded part dermes the length of the length (2 bytes);
the colored bytes define the length (260 bytes).

- Value. The value field codes the value of the data according to the rules defined in BER.

To show how these three fields-tag, length, and value-can define objects, we give some examples.

Example 28.1

Figure 28.11 shows how to define INTEGER 14.

Example 28.2

Figure 28.12 shows how to define the OCTET STRING "HI."

Figure 28.11 *Example* 28.1, *INTEGER 14*

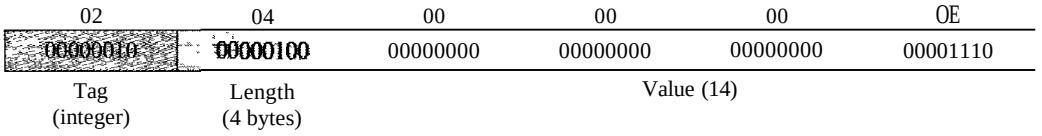
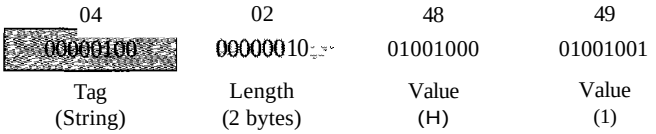


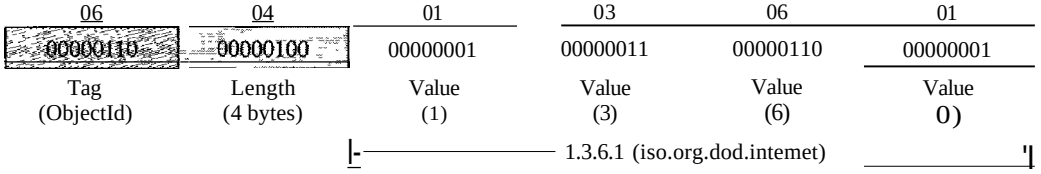
Figure 28.12 *Example* 28.2, *OCTET STRING "HI"*



Example 28.3

Figure 28.13 shows how to define ObjectIdentifier 1.3.6.1 (iso.org.dod.internet).

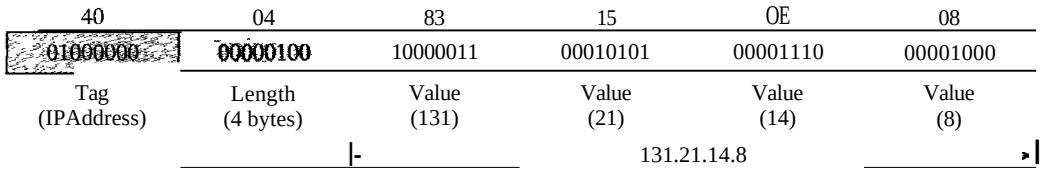
Figure 28.13 *Example* 28.3, *ObjectIdentifier 1.3.6.1*



Example 28.4

Figure 28.14 shows how to define IPAddress 131.21.14.8.

Figure 28.14 *Example* 28.4, *IPAddress 131.21.14.8*

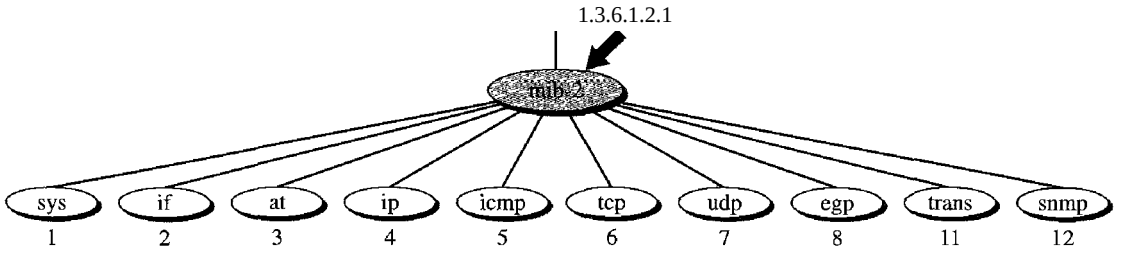


Management Information Base (MIB)

The Management Information Base, version 2 (MIB2) is the second component used in network management. Each agent has its own MIB2, which is a collection of all the objects that the manager can manage. The objects in MIB2 are categorized under 10

different groups: system, interface, address translation, ip, icmp, tcp, udp, egp, transmission, and snmp. These groups are under the mib-2 object in the object identifier tree (see Figure 28.15). Each group has defined variables and/or tables.

Figure 28.15 *mib-2*



The following is a brief description of some of the objects:

- sys This object *(system)* defines general information about the node (system), such as the name, location, and lifetime.
- if This object *(inteiface)* defines information about all the interfaces of the node including interface number, physical address, and IP address.
- at This object *(address translation)* defines the information about the ARP table.
- ip This object defines information related to IP, such as the routing table and the IP address.
- icmp This object defines information related to ICMP, such as the number of packets sent and received and total errors created.
- tcp This object defines general information related to TCP, such as the connection table, time-out value, number of ports, and number of packets sent and received.
- udp This object defines general information related to UDP, such as the number of ports and number of packets sent and received.
- snmp This object defines general information related to SNMP itself.

Accessing MIB Variables

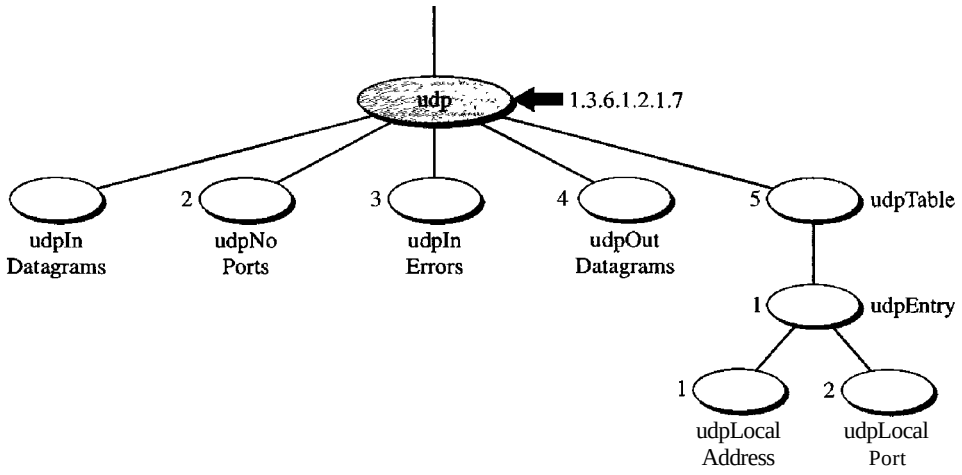
To show how to access different variables, we use the udp group as an example. There are four simple variables in the udp group and one sequence of (table of) records. Figure 28.16 shows the variables and the table.

We will show how to access each entity.

Simple Variables To access any of the simple variables, we use the id of the group (1.3.6.1.2.1.7) followed by the id of the variable. The following shows how to access each variable.

udpInDatagrams → 1.3,6.1.2.1.7.1
udpNoPorts → 1.3.6.1.2.1.7.2
udplnErrors → 1.3.6.1.2.1.7.3
udpOutDatagrams → 1.3.6.1.2.1.7.4

Figure 28.16 *udp group*



However, these object identifiers define the variable, not the instance (contents). To show the instance or the contents of each variable, we must add an instance suffix. The instance suffix for a simple variable is simply a 0. In other words, to show an instance of the above variables, we use the following:

udpInDatagrams.O	1.3.6.1.2.1.7.1.0
udpNoPorts.O	1.3.6.1.2.1.7.2.0
udpInErrors.0	1.3.6.1.2.1.7.3.0
udpOutDatagrams.0	1.3.6.1.2.1.7.4.0

Tables To identify a table, we first use the table id. The udp group has only one table (with id 5) as illustrated in Figure 28.17.

So to access the table, we use the following:

udpTable ➡ 1.3.6.1.2.1.7.5

However, the table is not at the leaf level in the tree structure. We cannot access the table; we define the entry (sequence) in the table (with id of 1), as follows:

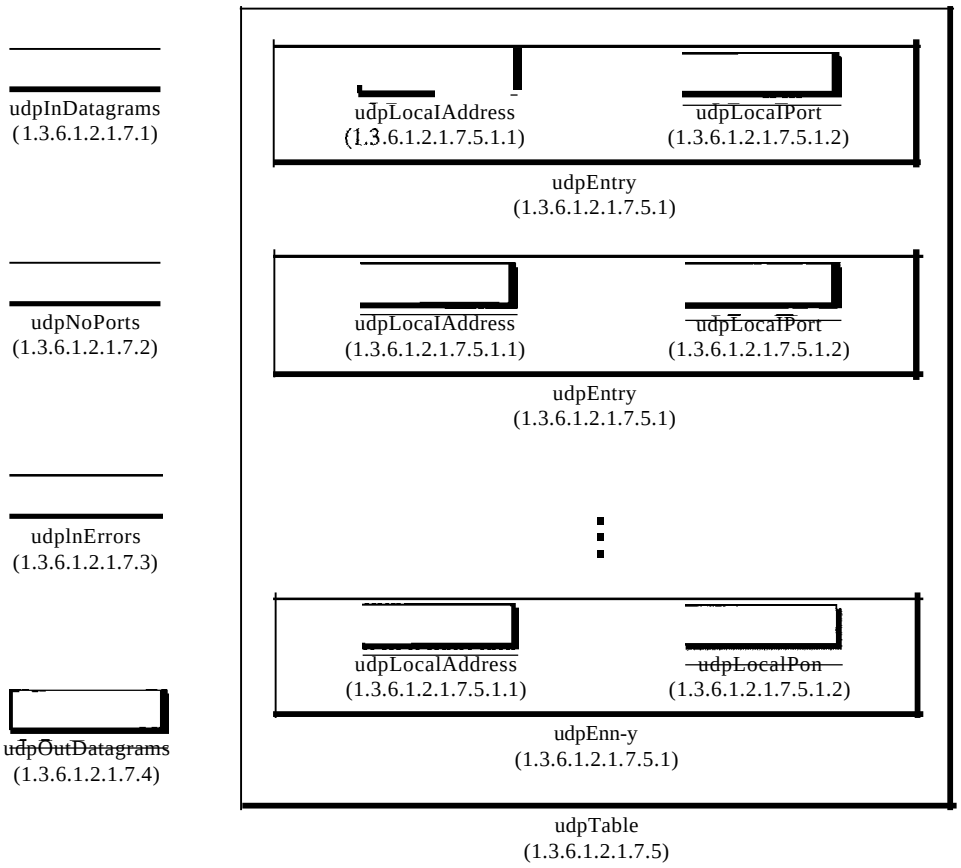
udpEntry ➡ 1.3.6.1.2.1.7.5.1

This entry is also not a leaf and we cannot access it. We need to define each entity (field) in the entry.

udpLocalAddress	1.3.6.1.2.1.7.5.1.1
udpLocalPort	1.3.6.1.2.1.7.5.1.2

These two variables are at the leaf of the tree. Although we can access their instances, we need to define *which* instance. At any moment, the table can have several values for

Figure 28.17 *udp variables and tables*



each local address/local port pair. To access a specific instance (row) of the table, we add the index to the above ids. In MIB, the indexes of arrays are not integers (like most programming languages). The indexes are based on the value of one or more fields in the entries. In our example, the udpTable is indexed based on both the local address and the local port number. For example, Figure 28.18 shows a table with four rows and values for each field. The index of each row is a combination of two values.

To access the instance of the local address for the first row, we use the identifier augmented with the instance index:

udpLocalAddress.181.23.45.14.23 ➡ 1.3.6.1.2.7.5.1.1.181.23.45.14.23

Note that not all tables are indexed in the same way. Some tables are indexed by using the value of one field, others by using the value of two fields, and so on.

Lexicographic Ordering

One interesting point about the MIB variables is that the object identifiers (including the instance identifiers) follow in lexicographic order. Tables are ordered according to column-row rules, which means one should go column by column. In each column, one should go from the top to the bottom, as shown in Figure 28.19.

Figure 28.18 *Indexes for udpTable*

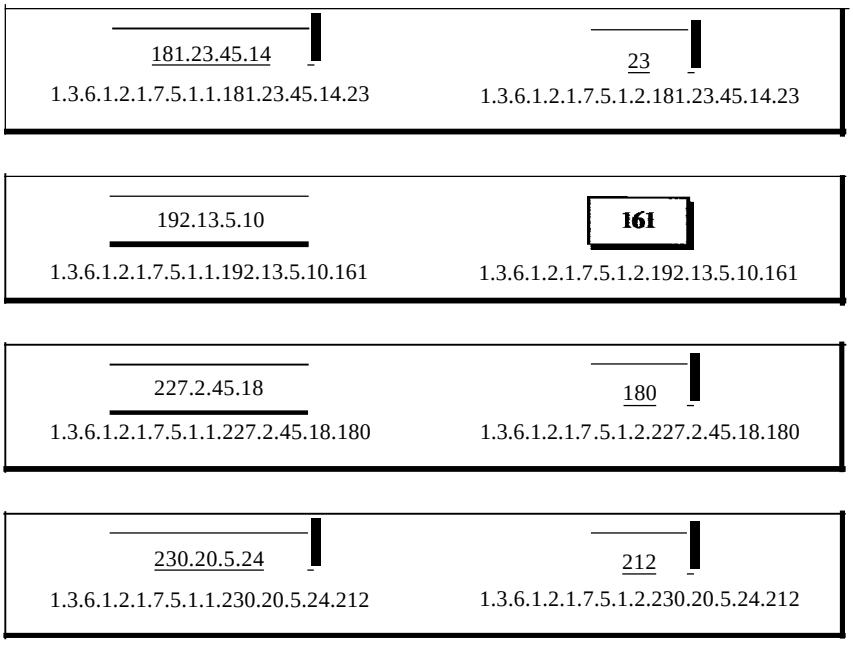
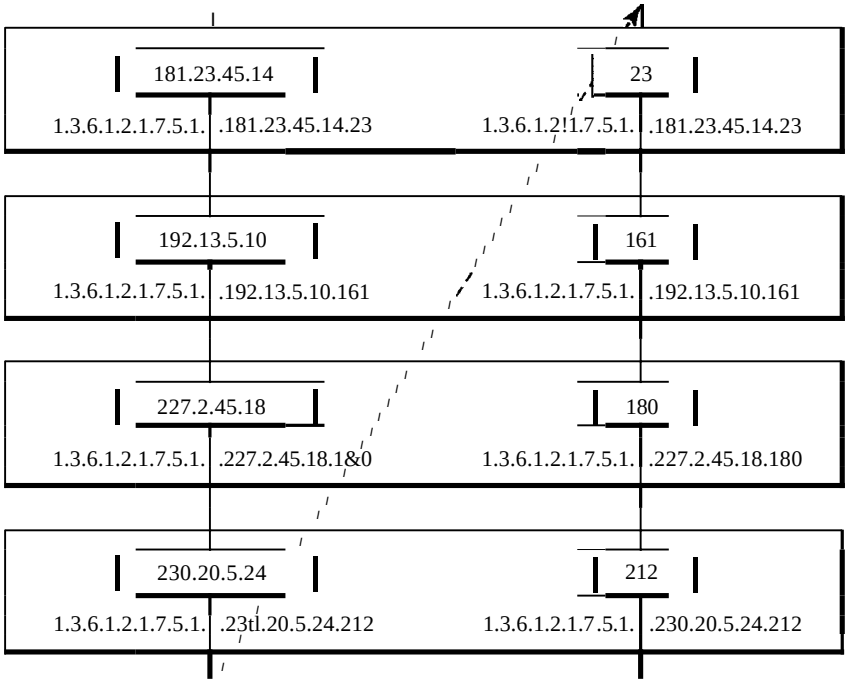


Figure 28.19 *Lexicographic ordering*



The lexicographic ordering enables a manager to access a set of variables one after another by defining the first variable, as we will see in the GetNextRequest command in the next section.

SNMP

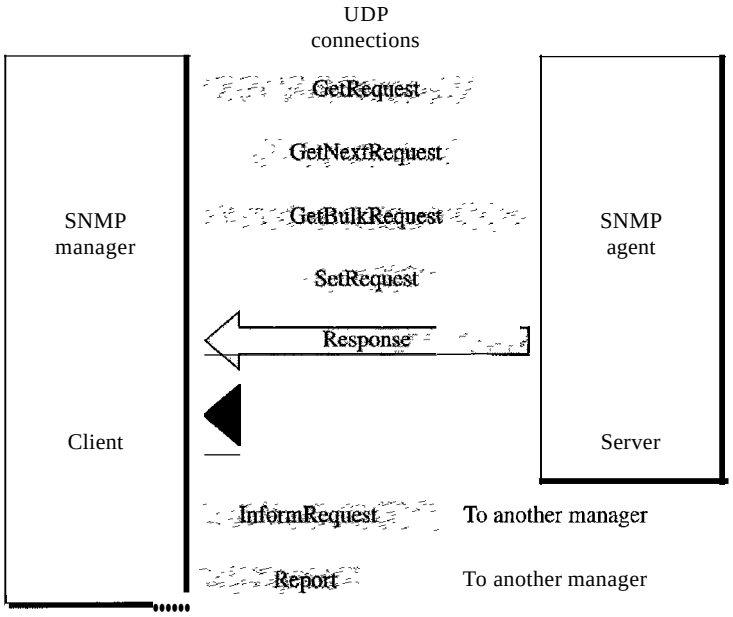
SNMP uses both SMI and MIB in Internet network management. It is an application program that allows

1. A manager to retrieve the value of an object defined in an agent
2. A manager to store a value in an object defined in an agent
3. An agent to send an alarm message about an abnormal situation to the manager

PDUs

SNMPv3 defines eight types of packets (or PDUs): GetRequest, GetNextRequest, GetBulkRequest, SetRequest, Response, Trap, InformRequest, and Report (see Figure 28.20).

Figure 28.20 *SNMP PDUs*



GetRequest The GetRequest PDU is sent from the manager (client) to the agent (server) to retrieve the value of a variable or a set of variables.

GetNextRequest The GetNextRequest PDU is sent from the manager to the agent to retrieve the value of a variable. The retrieved value is the value of the object following the defined Objectid in the PDD. It is mostly used to retrieve the values of the entries in a table. If the manager does not know the indexes of the entries, it cannot retrieve the values. However, it can use GetNextRequest and define the ObjectId of the table. Because the first entry has the ObjectId immediately after the ObjectId of the table, the value of the first entry is returned. The manager can use this ObjectId to get the value of the next one, and so on.

GetBulkRequest The GetBulkRequest POD is sent from the manager to the agent to retrieve a large amount of data. It can be used instead of multiple GetRequest and GetNextRequest PODs.

SetRequest The SetRequest PDD is sent from the manager to the agent to set (store) a value in a variable.

Response The Response PDD is sent from an agent to a manager in response to GetRequest or GetNextRequest. It contains the value(s) of the variable(s) requested by the manager.

Trap The Trap (also called SNMPv2 Trap to distinguish it from SNMPv1 Trap) POD is sent from the agent to the manager to report an event. For example, if the agent is rebooted, it infonns the manager and reports the time of rebooting.

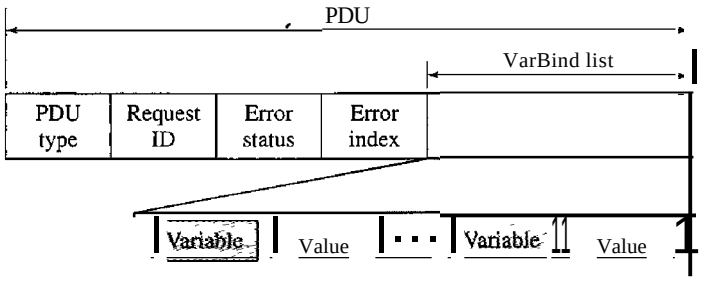
InformRequest The InfonnRequest POD is sent from one manager to another remote manager to get the value of some variables from agents under the control of the remote manager. The remote manager responds with a Response POD.

Report The Report POD is designed to report some types of errors between managers. It is not yet in use.

Format

The fonnat for the eight SNMP PODs is shown in Figure 28.21. The GetBulkRequest POD differs from the others in two areas, as shown in the figure.

Figure 28.21 *SNMP PDU format*



Differences:

1. Error status and error index values are zeros for all request messages except GetBulkRequest.
2. Error status field is replaced by nonrepeater field and error index field is replaced by max-repetitions field in GetBulkRequest.

The fields are listed below:

- **PDU** type. This field defines the type of the POD (see Table 28.4).
- Request ID. This field is a sequence number used by the manager in a Request POD and repeated by the agent in a response. It is used to match a request to a response.

- Error status. This is an integer that is used only in Response PDUs to show the types of errors reported by the agent. Its value is 0 in Request PDUs. Table 28.3 lists the types of errors that can occur.

Table 28.3 *Types of errors*

Status	*Name*	*Meaning*
0	noError	No error
1	tooBig	Response too big to fit in one message
2	noSuchName	Variable does not exist
3	badValue	The value to be stored is invalid
4	readOnly	The value cannot be modified
5	genErr	Other errors

- Nonrepeaters. This field is used only in GetBulkRequest and replaces the error status field, which is empty in Request PDUs.
- Error index. The error index is an offset that tells the manager which variable caused the error.
- **Max-repetition.** This field is also used only in GetBulkRequest and replaces the error index field, which is empty in Request PDUs.
- VarBind list. This is a set of variables with the corresponding values the manager wants to retrieve or set. The values are null in GetRequest and GetNextRequest. In a Trap PDU, it shows the variables and values related to a specific PDU.

Messages

SNMP does not send only a PDU, it embeds the PDU in a message. A message in SNMPv3 is made of four elements: version, header, security parameters, and data (which include the encoded PDU), as shown in Figure 28.22.

Because the length of these elements is different from message to message, SNMP uses BER to encode each element. Remember that BER uses the tag and the length to define a value. The *version* defines the current version (3). The *header* contains values for message identification, maximum message size (the maximum size of the reply), message flag (one octet of data type OCTET STRING where each bit defines security type, such as privacy or authentication, or other information), and a message security model (defining the security protocol). The message *security parameter* is used to create a message digest (see Chapter 31). The data contain the PDU. If the data are encrypted, there is information about the encrypting engine (the manager program that did the encryption) and the encrypting context (the type of encryption) followed by the encrypted PDU. If the data are not encrypted, the data consist of just the PDU.

To define the type of PDU, SNMP uses a tag. The class is context-sensitive (10), the format is structured (1), and the numbers are 0, 1, 2, 3, 5, 6, 7, and 8 (see Table 28.4). Note that SNMPv1 defined A4 for Trap, which is obsolete today.

Figure 28.22 *SNMP message*

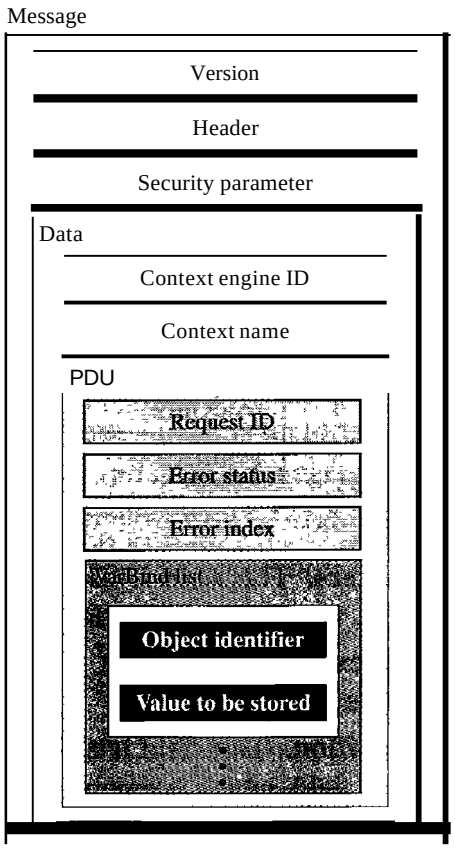


Table 28.4 *Codes for SNMP messages*

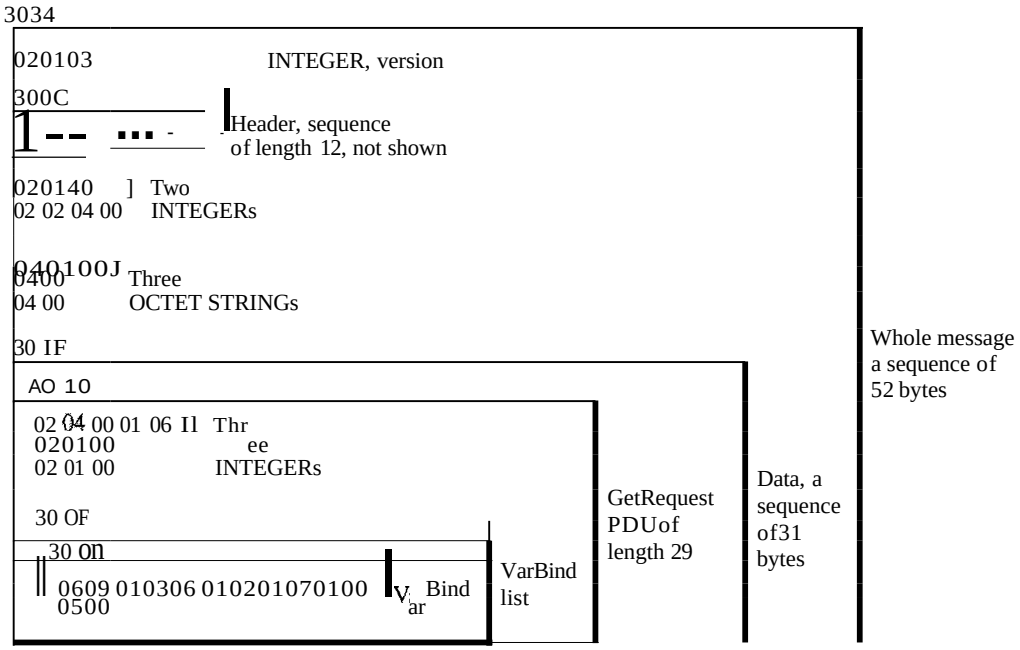
Data	*Class*	*Format*	*Number*	*Whole Tag (Binary)*	*Whole Tag (Hex)*
GetRequest	10	1	00000	**10100000**	**AO**
GetNextRequest	10	1	00001	**10100001**	**Al**
Response	10	1	00010	**10100010**	**A2**
SetRequest	10	1	00011	**10100011**	**A3**
GetBulkRequest	10	1	00101	**10100101**	**AS**
InformRequest	10	1	00110	**10100110**	**A6**
Trap (SNMPv2)	10	1	00111	**10100111**	**A7**
Report	10	1	01000	**10101000**	**A8**

***Example** 28,5*

In this example, a manager station (SNMP client) uses the GetRequest message to retrieve the number of UDP datagrams that a router has received.

There is only one VarBind entity. The corresponding MID variable related to this information is udpInDatagrams with the object identifier 1.3.6.1.2.1.7.1.0. The manager wants to retrieve a value (not to store a value), so the value defines a null entity. Figure 28.23 shows the conceptual

Figure 28.23 *Example 28.5*



view of the packet and the hierarchical nature of sequences. We have used white and colored boxes for the sequences and a gray one for the PDU.

The VarBind list has only one VarBind. The variable is of type 06 and length 09. The value is of type 05 and length 00. The whole VarBind is a sequence of length OD (13). The VarBind list is also a sequence of length OF (15). The GetRequest PDU is of length ID (29).

Now we have three OCTET STRINGs related to the security parameter, security model, and flags. Then we have two integers defining maximum size (1024) and message ID (64). The header is a sequence of length 12, which we left blank for simplicity. There is one integer, version (version 3). The whole message is a sequence of 52 bytes.

Figure 28.24 shows the actual message sent by the manager station (client) to the agent (server).

UDPPorts

SNMP uses the services of UDP on two well-known ports, 161 and 162. The well-known port 161 is used by the server (agent), and the well-known port 162 is used by the client (manager).

The agent (server) issues a passive open on port 161. It then waits for a connection from a manager (client). A manager (client) issues an active open, using an ephemeral port, The request messages are sent from the client to the server, using the ephemeral port as the source port and the well-known port 161 as the destination port. The response messages are sent from the server to the client, using the well-known port 161 as the source port and the ephemeral port as the destination port.

The manager (client) issues a passive open on port 162. It then waits for a connection from an agent (server). Whenever it has a Trap message to send, an agent (server) issues an active open, using an ephemeral port. This connection is only one-way, from the server to the client (see Figure 28.25).

Figure 28.24 *GetRequest message*

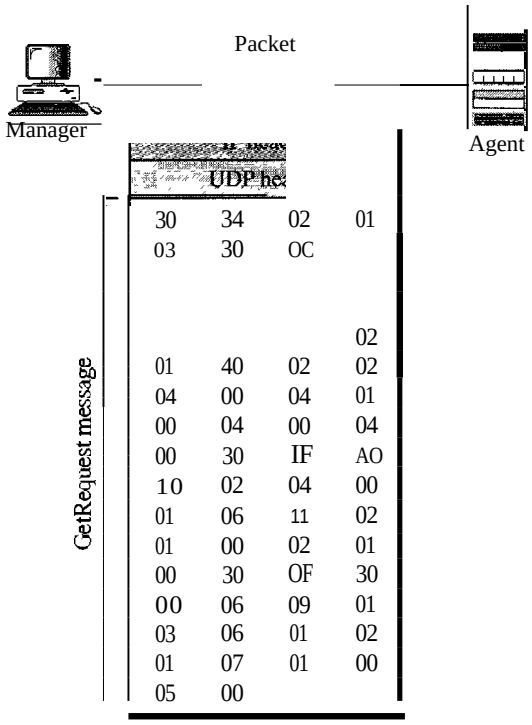
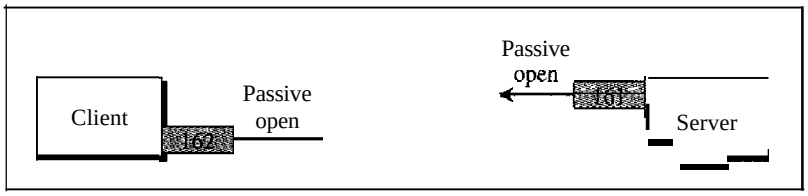
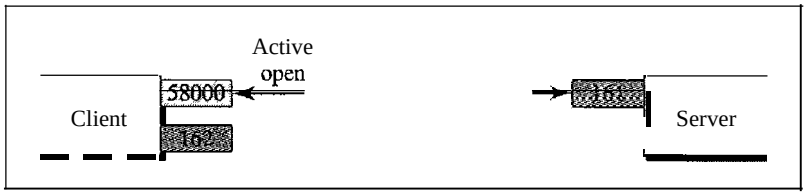


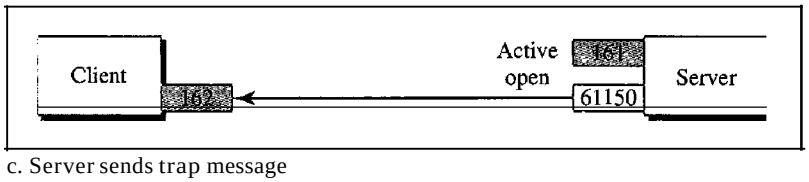
Figure 28.25 *Port numbers for SNMP*



a. Passive open by both client and server



b. Exchange of request and response messages



c. Server sends trap message

The client/server mechanism in SNMP is different from other protocols. Here both the client and the server use well-known ports. In addition, both the client and the server are running infinitely. The reason is that request messages are initiated by a manager (client), but Trap messages are initiated by an agent (server).

Security

The main difference between SNMPv3 and SNMPv2 is the enhanced security. SNMPv3 provides two types of security: general and specific. SNMPv3 provides message authentication, privacy, and manager authorization. We discuss these three aspects in Chapter 31. In addition, SNMPv3 allows a manager to remotely change the security configuration, which means that the manager does not have to be physically present at the manager station.

28.3 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and sites. The items in brackets [...] refer to the reference list at the end of the text.

Books

SNMP is discussed in [MS01], Chapter 25 of [Ste94], Section 22.3 of [Sta04], and Chapter 39 of [Com04]. Network management is discussed in [Sub01].

Sites

The following sites are related to topics discussed in this chapter.

- www.ietf.org/rfc.html Information about RFCs

RFCs

The following RFCs are related to SNMP, MIB, and SMI:

1065, 1067,1098, 1155,1157, 1212, 1213, 1229, 1231, 1243, 1284, 1351, 1352,1354, 1389, 1398, 1414, 1441, 1442, 1443, 1444,1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1461, 1472, 1474, 1537, 1623,1643,1650, 1657, 1665, 1666, 1696, 1697, 1724, 1742,1743, 1748, 1749

28.4 KEY TERMS

Abstract Syntax Notation 1 (ASN.l)
accounting management
agent
Basic Encoding Rules (BER)
configuration management
fault management
hardware documentation
lexicographic ordering
Management Information Base (MIB)
manager
network management
object identifier
performance management
security management

simple data type
Simple Network Management Protocol (SNMP)
structured data type
Structure of Management Information (SMI)
trap

28.5 SUMMARY

- The five areas comprising network management are configuration management, fault management, performance management, accounting management, and security management.
- Configuration management is concerned with the physical or logical changes of network entities. It includes the reconfiguration and documentation of hardware, software, and user accounts.
- Fault management is concerned with the proper operation of each network component. It can be reactive or proactive.
- Performance management is concerned with the monitoring and control of the network to ensure the network runs as efficiently as possible. It is quantified by measuring the capacity, traffic, throughput, and response time.
- Security management is concerned with controlling access to the network.
- Accounting management is concerned with the control of user access to network resources through charges.
- Simple Network Management Protocol (SNMP) is a framework for managing devices in an internet using the TCP/IP protocol suite.
- A manager, usually a host, controls and monitors a set of agents, usually routers.
- The manager is a host that runs the SNMP client program.
- The agent is a router or host that runs the SNMP server program.
- SNMP frees management tasks from both the physical characteristics of the managed devices and the underlying networking technology.
- SNMP uses the services of two other protocols: Structure of Management Information (SMI) and Management Information Base (MIB).
- SMI names objects, defines the type of data that can be stored in an object, and encodes the data.
- SMI objects are named according to a hierarchical tree structure.
- SMI data types are defined according to Abstract Syntax Notation 1 (ASN.l).
- SMI uses Basic Encoding Rules (BER) to encode data.
- MIB is a collection of groups of objects that can be managed by SNMP.
- MIB uses lexicographic ordering to manage its variables.
- SNMP functions in three ways:
  1. A manager can retrieve the value of an object defined in an agent.
  2. A manager can store a value in an object defined in an agent.
  3. An agent can send an alarm message to the manager.

- SNMP defines eight types of packets: GetRequest, GetNextRequest, SetRequest, GetBulkRequest, Trap, InformRequest, Response, and Report.
- SNMP uses the services of UDP on two well-known ports, 161 and 162.
- SNMPv3 has enhanced security features over previous versions.

28.6 PRACTICE SET

Review Questions

1. Define network management.
2. List five functions of network management.
3. Define configuration management and its purpose.
4. List two subfunctions of configuration management.
5. Define fault management and its purpose.
6. List two subfunctions of fault management.
7. Define performance management and its purpose.
8. List four measurable quantities of performance management.
9. Define security management and its purpose.
10. Define account management and its purpose.

Exercises

11. Show the encoding for INTEGER 1456.
12. Show the encoding for the OCTET STRING "Hello World."
13. Show the encoding for an arbitrary OCTET STRING of length 1000.
14. Show how the following record (sequence) is encoded.

INTEGER	OCTET STRING	IP Address
2345	"COMPUTER"	185.32.1.5

15. Show how the following record (sequence) is encoded.

Time Tick	INTEGER	Object Id
12000	14564	1.3.6.1.2.1.7

16. Show how the following array (sequence of) is encoded. Each element is an integer.

2345
1236
122
1236

17. Show how the following array of records (sequence of sequence) is encoded.

INTEGER	OCTET STRING	Counter
2345	"COMPUTER"	345
1123	"DISK"	1430
3456	"MONITOR"	2313

18. Decode the following.
    a. 02 04 01 02 14 32
    b. 30060201 11 0201 14
    c. 300904034143420202 14 14
    d. 30 OA 40 04 2351 6271 0202 14 12