CHAPTER 9

PRACTICE SET

Questions

Q9-1. Fault and performance are areas of management defined by OSI; personnel is not.

Q9-3. This is related to hardware reconfiguration, which is part of reconfiguration, which is in turn part of configuration management.

Q9-5. Reactive fault management is responsible for handling faults in a timely manner. Proactive fault management is responsible for preventing some faults from occurring.

Q9-7. Internal traffic measures the number of packets circulated inside the network; external traffic measures the number of packets sent to or received from outside.

Q9-9. A router or a switch cannot be used as a manager station; only a host can be a manager station.

Q9-11. The name is encoded as 1.3.6 (see Figure 9.6).

Q9-13.

a. Simple	**b. Simple**	**c. Simple**
d. Sequence of	**e. Sequence**	**f. Sequence of**

Q9-15. SMI only sets the rules for naming objects, distinguishes between simple data types and shows how to combine them to make structured data types, and specifies how to encode objects and the values to be stored in those objects. MIB, on the other hand, defines the objects to be managed in SNMP using the rules set by SMI.

Q9-17. The identifiers of the variables are *x*.1, *x*.2, and *x*.3.

Q9-19. SNMP can only reference an entity which is a leaf in the MIB tree. Non-leaf entities in the leaf are not variables to be accessed.

Q9-21. We can say that tables in the MIB are column-oriented. This means that each column in a table is represented by a leaf in the MIB tree. In this case, we have three leaves related to the table. The rows in a table are not represented as

leaves in the MIB tree because the number of rows are not fixed from one object to another. Consider the case of a forwarding (routing) table. A router may have a forwarding table with three rows; another table may have a forwarding table with eight rows.

Q9-23.

a. A GetRequest PDU is sent from a client (manager) to a server (agent).

b. A Response PDU is sent from a server (agent) to a client (manager).

c. A Trap PDU is sent from a server (agent) to a client (manager).

Problems

P9-1. The identifiers are (*x*.1) and (*x*.2). The variable identifier of an object does not have any relation to the type of the variable. Each simple variable of an object is a leaf on the MIB tree, as shown below:

P9-3. Since MIB considers each column of a tree as a variable, we have four leaves on the tree. The identifiers for the two variables are (*x*.1) and (*x*.2). The identifier for the table is (*x*.3), which is not a leaf. The identifier for the table entry is (*x*.3.1). Each column has an identifier on the leaf. The identifier for the first column is (*x*.3.1.1); the identifier for the second column is (*x*.3.1.2).

P9-5. A simple variable has only one instance, which is not on the MIB tree. The instance can be referred to by adding a zero to the identifier of the variable. In this case, the identifiers of the variables are (*x*.1) and (*x*.2). The instances can be referred to as (*x*.1.**0**) and (*x*.2.**0**).

P9-7. The identifier of the table is (1.3.6.1.2.4.21). The identifier of the table entry is (1.3.6.1.2.4.21.1). The identifier of the second column is (1.3.6.1.2.4.21.1.2). Each instance must add the corresponding index, which is the destination IP addresses:

First instance:	**column 2, row 1:**	**1.3.6.1.2.4.21.1.2.201.14.67.0**
Second instance:	**column 2, row 2:**	**1.3.6.1.2.4.21.1.2.123.16.0.0**
Third instance:	**column 2, row 3:**	**1.3.6.1.2.4.21.1.2.11.0.0.0**
Fourth instance:	**column 2, row 4:**	**1.3.6.1.2.4.21.1.2.0.0.0.0**

P9-9. Using TLV (tag, length, and value), we have:

Tag	Length	Value
04	0C	48 65 6C 6C 6F 20 57 6F 72 6C 64 2E

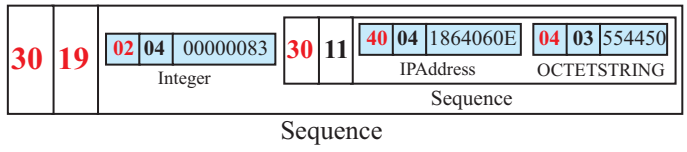
The length of the string (including space and period) is 12 or 0C in hexadecimal. The value of the string is based on ASCII (see Appendix A). The encoding in compact form is **040C48656C6C6F20576F726C642E**.

P9-11. Using TLV (tag, length, and value), we have:

Tag	Length	Value
06	08	01 03 06 01 02 01 07 01

Each digit in the identifier is translated into two hexadecimal digits. The encoding in compact form is **06080103060102010701**.

P9-13. First we need to encode the inner sequence and then the outer one, as shown below:



The total length is (13 + 6 = 19) bytes. The whole data type in compact format is

3019020400000083301140041846060E040355335O

P9-15. Since the code starts with the tag 30, it is a sequence. We need to split the code carefully to find each component.

T	L	T	L	V	T	L	V
30	0C	02	04	00 00 09 98	40	04	0A 05 03 0E

The first byte is the tag for a sequence and the next byte is the length of the sequence (12 bytes). The next byte defines a data type (an integer). The next byte is the length of the integer (4 bytes) and the next four bytes define the value (2456). The last tag defines an IP address with the value 10.5.3.14. In summary, the code defines a sequence of an integer and an IP address. The following shows the whole data type using ASN.1:

```
SEQUENCE
{
        INTEGER 2456
        IP Address 10.5.3.14
}
```

P9-17. A VarBind is a sequence of two data items: the corresponding object identifier and the value of the variable.

a. In the GetRequest message the sequence is the instance of the object identifier (the last 00 defines the instance). The last two bytes defines the identifier of the null object (05) and the null value (00).

T	L	T	L	V	T	L
30	0D	06	0B	0103060102010704 00	05	00

b. In the Response message the sequence is the instance of the object identifier (the last 00 defines the instance). The last six bytes defines a counter of length 4 with the value 15.

T	L	T	L	V	T	L	V
30	11	06	0F	0103060102010704 00	41	04	0000000F

P9-19. A GetRequest PDU is a structure of five elements, as shown in Figure 9.18. It can be defined as shown below. Note that VarBindList is itself a structure that needs to be defined.

```
GetRequest PDU ::= SEQUENCE
{
        PDUType Tag
        RequestID Integer32
        ErrorStatus INTEGER (0..18)
        ErrorIndex  INTEGER
        VariableBinding VarBindList
}
```

P9-21. A VarBindList is a list of VarBinds. Note that VarBind is itself a structure that needs to be defined.

```
VarBindList ::= SEQUENCE OF
{
        VarBindList VarBind
        ...
}
```