CHAPTER 27

PRACTICE SET

Questions

Q27-1. The identifiers of the variables are *x*.1, *x*.2, and *x*.3.

Q27-2. We need to check the value of the second byte, $(09)_{16}$. Since the leftmost bit in this byte is 0, the rest of the bits, $(0001001)_2$, define the size of the value field. The value field is 9 bytes long.

Q27-3. Internal traffic measures the number of packets circulated inside the network; external traffic measures the number of packets sent to or received from outside.

Q27-4. A manager station runs a client program. The program starts when a network manager needs to control the agents in the networks.

Q27-5.

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).

Q27-6. This is related to software reconfiguration, which is part of reconfiguration, which is in turn part of configuration management.

Q27-7. 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.

Q27-8. Reconfiguration and documentation are part of configuration management; encryption is part of security management.

Q27-9. This is related to hardware reconfiguration, which is part of reconfiguration, which is in turn part of configuration management.

Q27-10. The *if* object defines a set of variables in MIB that can store a set of values about the interfaces of a node (router). The manager needs to know informa-

tion about this object such as the number of interfaces, the IP address of each interface, and the link-layer (MAC) address of each interface.

Q27-11. Reactive fault management is responsible for handling faults in a timely manner. Proactive fault management is responsible for preventing some faults from occurring.

Q27-12. Each simple variable is a leaf in the MIB tree. This means that the three simple variables create a subtree with three leaves whose root is the corresponding object.

Q27-13. A router or a switch cannot be used as a manager station; only a host can be a manager station.

Q27-14. This is related to accounting management.

Q27-15. 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.

Q27-16.

a. Source port number is ephemeral; destination port number is 161.

b. Source port number is 161; destination port number is the source port number in the corresponding request message.

c. Source port number is ephemeral; destination port number is 162. Note that in this case the manager is acting as a server, but is using the well-known port number 162 instead of 161.

d. Source port number is ephemeral; destination port number is 162. Note that in this case the receiving manager is acting as a server, but is using the well-known port number 162 instead of 161.

Q27-17.

a. Simple

b. Simple

c. Simple

d. Sequence of

e. Sequence

f. Sequence of

Q27-18. In this case, proactive fault management has been ignored.

Q27-19. The name is encoded as 1.3.6.

Q27-20. This is not allowed because it is not formed according to the tree structure of Figure 27.6. It should be "ios.org.dod.internet".

Q27-21. 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.

Q27-22. A GetRequest PDU is sent to retrieve the value of a variable; a SetRequest PDU is sent to set (change) the value of a variable.

Q27-23. Fault and performance are areas of management defined by OSI; personnel is not.

Q27-24. SNMP cannot reference the entire row of a table. Each cell in a table is a separate instance of the variable defined by the table identifier. Each instance should be accessed individually.

Problems

P27-1. 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:

P27-2. An SNMP message (Figure 27.19) is a sequence of two entities. The first is a sequence; the second is a PDU, which itself needs to be defined.

```
SNMPMessage ::= SEQUENCE
{
        MessageHeader SEQUECE
        ScopedPDU PDU
}
```

P27-3. 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).

P27-4. When there is no structured data type (tag: 30), the code is a simple data type. The first two bytes are the tag, the next two bytes are the length (or the length of the length), and the rest is the value. In this case, we have

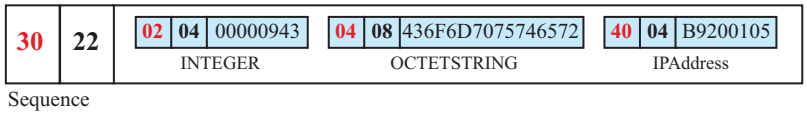
Tag	Length	Value
02	04	00 00 C7 38

Since the tag is 02, the type is an integer of value 51000.

P27-5. 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**

P27-6. We first encode the simple data types and then the sequence, as shown below:

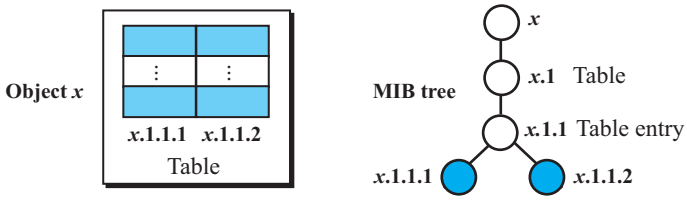


Note that the length of the sequence is larger than the sum of the lengths of the three simple data types. The total length is (6 + 10 + 6 = 22) bytes. The whole data type in compact format is

302202040000094304084367 6D7707475657240044B200105

P27-7. 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**).

P27-8. MIB considers each column of a table as a leaf on the MIB tree. Each column needs to have an identifier. However, first we need to have an identifier for the table, second we need to have an identifier for the table entry, and then we can assign identifiers to the columns. The identifier for the table is (*x*.1). The identifier for the table entry is (*x*.1.1). These two identifiers are not leaves, they are middle nodes. Each column has an identifier, which is the leaf of the tree. The identifier of the first column is (*x*.1.1.1); the identifier of the second column is (*x*.1.1.2). See below.



P27-9. A GetRequest PDU is a structure of five elements, as shown in Figure 27.18. 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
}
```

P27-10. Using TLV (tag, length, and value), we have:

Tag	Length	Value
40	04	70 38 17 4E

Each byte in the IPAddress is two hexadecimal digits. The encoding in compact form is 40047038174E.

P27-11. 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	010306010201070400	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	010306010201070400	41	04	0000000F

P27-12. A Response PDU is a structure of five elements, as shown in Figure 27.18. 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
}
```

P27-13. 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.

P27-14. Since the code starts with the tag 30, it is a sequence. However, when we split the code, we find another tag of value 30, which means there is a sequence inside another sequence.

T	L	T	L	V	T	L	T	L	V	T	L
30	0D	04	02	4E6F	30	07	06	03	01006	05	00

The first two bytes define an outer sequence of length 13 bytes. The third byte defines a string of value "No". The next two bytes define an inner sequence of seven bytes. The first data item in the inner sequence is an object identifier (tag 06) which defines the object 1.3.6. The last item in the inner sequence is a null value of length 0. The following shows the whole data type using ASN.1:

```
SEQUENCE OF
{
        OCTET STRING "No"
        SEQUENCE
        {
                OBJECT IDENTIFIER 1.3.6
                NULL
        }
}
```

P27-15. 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

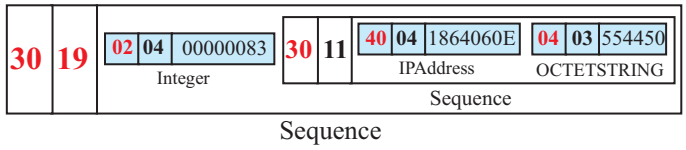
The length of the string (including space and period) is 12 or 0C in hexadecimal. The value of the string is based on ASCII (see Appendix A). The encoding in compact form is **04 0C** 48656C6C6F20576F726C642E.

P27-16. Using TLV (tag, length, and value), we have:

Tag	Length	Value
02	04	00 00 05 B0

The decimal value 1456 is 000005B0 in hexadecimal. The encoding in compact form is **02 04** 000005B0.

P27-17. 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

30190204000000833011400418460 60E0403553350

P27-18. The identifier for the variable is (*x*.1). The identifier for the first table is (*x*.2). The identifier for the first table entry is (*x*.2.1), which means that the identifiers for the two columns are (*x*.2.1.1) and (*x*.2.1.2). The identifier for the second table is (*x*.3). The identifier for the second table entry is (*x*.3.1), which means that the identifiers for the three columns are (*x*.3.1.1), (*x*.3.1.2), and (*x*.3.1.3).

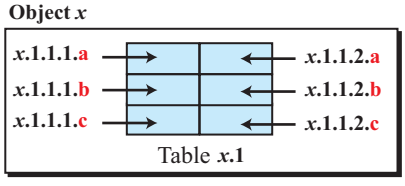
P27-19. Since the code starts with the tag 30, it is a sequence. We need to split the code carefully to find each component.

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
}
```

P27-20. A table has as many instances as the number of active cells at each moment. Each instance can be referred to by adding the index of the row to the identifier of the column. In this case, the index of each row is the content of the first column in that row (a, b, c). We first need to find the identifier for each column and then add the corresponding index for each row. The identifier of the table is (*x*.1). The identifier of the table entry is (*x*.1.1). The identifiers of the columns are (*x*.1.1.1) and (*x*.1.1.2). The following shows how each cell can be referred to.



P27-21. A VarBindList is a list of VarBinds. Note that VarBind is itself a structure that needs to be defined.

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