CHAPTER 22

Internet Applications—Electronic Mail and Network Management

One of the most exciting aspects of birds' lives is how they interact with others during such social activities as defending territories, courting mates, nesting, raising young, and flocking. Birds' level of sociability changes with the seasons; they may be gregarious at certain times of year yet highly territorial at others. Some of the most fascinating behavior occurs in spring and summer when birds are engaged in breeding. During a social interaction, an individual is coordinating its activities with those of another. This inevitably requires communication.

—*Secret Lives of Common Birds*, Marie Read

KEY TOPICS

- The most widely used protocol for the transmission of electronic mail is SMTP. SMTP assumes that the content of the message is a simple text block. The recent MIME standard expands SMTP to support transmission of multimedia information.
- The most important standardized scheme for supporting network management applications is the Simple Network Management Protocol (SNMP). The original version of SNMP is available on a wide array of products and is widely used. SNMPv2 contains a number of functional enhancements to SNMP and is supplanting it. SNMPv3 provides security features that are added on to SNMPv2.

All of the protocols and functions described in Part Five are geared toward one objective: the support of distributed applications that involve the interaction of multiple independent systems. In the OSI model, such applications occupy the application layer and are directly supported by the presentation layer. In the TCP/IP suite, such applications typically rely on TCP or UDP for support.

In this chapter, we examine two applications that give the reader a feel for the range and diversity of applications supported by a communications architecture. The chapter begins with electronic mail, with the SMTP and MIME standards as examples; SMTP provides a basic e-mail service, while MIME adds multimedia capability to SMTP. The chapter then discusses network management, a support-type application, designed to assure the effective monitoring and control of a distributed system. The specific protocol that is examined is the Simple Network Management Protocol (SNMP), which is designed to operate in both the TCP/IP and OSI environments.

Refer to Figure 2.5 to see the position within the TCP/IP suite of the protocols discussed in this chapter.

22.1 ELECTRONIC MAIL—SMTP AND MIME

The most heavily used application in virtually any distributed system is electronic mail. The Simple Mail Transfer Protocol (SMTP) has always been the workhorse of the TCP/IP suite. However, SMTP has traditionally been limited to the delivery of simple text messages. In recent years, there has been a demand for the capability to deliver mail containing various types of data, including voice, images, and video clips. To satisfy this requirement, a new electronic mail standard, which builds on SMTP, has been defined: the Multi-Purpose Internet Mail Extension (MIME). In this section, we first examine SMTP, and then look at MIME.

Simple Mail Transfer Protocol (SMTP)

SMTP is the standard protocol for transferring mail between hosts in the TCP/IP suite; it is defined in RFC 821.

Although messages transferred by SMTP usually follow the format defined in RFC 822, described later, SMTP is not concerned with the format or content of messages themselves, with two exceptions. This concept is often expressed by saying that SMTP uses information written on the *envelope* of the mail (message header), but does not look at the contents (message body) of the envelope. The two exceptions are as follows:

1. SMTP standardizes the message character set as 7-bit ASCII.
2. SMTP adds log information to the start of the delivered message that indicates the path the message took.

Basic Electronic Mail Operation Figure 22.1 illustrates the overall flow of mail in a typical system. Although much of this activity is outside the scope of SMTP, the figure illustrates the context within which SMTP typically operates.

To begin, mail is created by a user agent program in response to user input. Each created message consists of a header that includes the recipient's e-mail address and other information, and a body containing the message to be sent. These messages are then queued in some fashion and provided as input to an SMTP Sender program, which is typically an always-present server program on the host.

Although the structure of the outgoing mail queue will differ depending on the host's operating system, each queued message conceptually has two parts:

1. The message text, consisting of
   - The RFC 822 header: This constitutes the message envelope and includes an indication of the intended recipient or recipients.
   - The body of the message, composed by the user.
2. A list of mail destinations.

The list of mail destinations for the message is derived by the user agent from the 822 message header. In some cases, the destination or destinations are literally specified in the message header. In other cases, the user agent may need to expand mailing list names, remove duplicates, and replace mnemonic names with actual

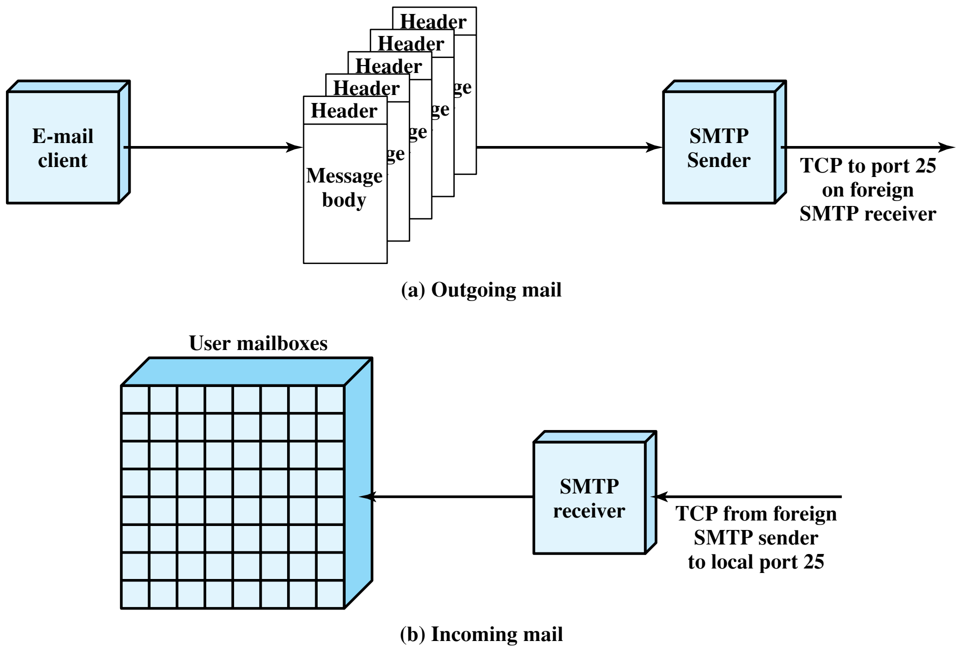

Figure 22.1 SMTP Mail Flow

mailbox names. If any blind carbon copies (BCCs) are indicated, the user agent needs to prepare messages that conform to this requirement. The basic idea is that the multiple formats and styles preferred by humans in the user interface are replaced by a standardized list suitable for the SMTP send program.

The **SMTP sender** takes messages from the outgoing mail queue and transmits them to the proper destination host via SMTP transactions over one or more TCP connections to port 25 on the target hosts. A host may have multiple SMTP senders active simultaneously if it has a large volume of outgoing mail, and should also have the capability of creating SMTP receivers on demand so that mail from one host cannot delay mail from another.

Whenever the SMTP sender completes delivery of a particular message to one or more users on a specific host, it deletes the corresponding destinations from that message's destination list. When all destinations for a particular message are processed, the message is deleted from the queue. In processing a queue, the SMTP sender can perform a variety of optimizations. If a particular message is sent to multiple users on a single host, the message text need be sent only once. If multiple messages are ready to send to the same host, the SMTP sender can open a TCP connection, transfer the multiple messages, and then close the connection rather than opening and closing a connection for each message.

The SMTP sender must deal with a variety of errors. The destination host may be unreachable, out of operation, or the TCP connection may fail while mail is being transferred. The sender can requeue the mail for later delivery but give up after some period rather than keep the message in the queue indefinitely. A common error is a faulty destination address, which can occur due to user input error or because the intended destination user has a new address on a different host. The SMTP sender must either redirect the message if possible or return an error notification to the message's originator.

The **SMTP protocol** is used to transfer a message from the SMTP sender to the SMTP receiver over a TCP connection. SMTP attempts to provide reliable operation but does not guarantee to recover from lost messages. SMTP does not return an end-to-end acknowledgment to a message's originator to indicate that a message is successfully delivered to the message's recipient. Also, SNMP does not guarantee to return error indications. However, the SMTP-based mail system is generally considered reliable.

The **SMTP receiver** accepts each arriving message and either places it in the appropriate user mailbox or copies it to the local outgoing mail queue if forwarding is required. The SMTP receiver must be able to verify local mail destinations and deal with errors, including transmission errors and lack of storage capacity.

The SMTP sender is responsible for a message up to the point where the SMTP receiver indicates that the transfer is complete; however, this simply means that the message has arrived at the SMTP receiver, not that the message has been delivered to and retrieved by the intended final recipient. The SMTP receiver's error-handling responsibilities are generally limited to giving up on TCP connections that fail or are inactive for very long periods. Thus, the sender has most of the error recovery responsibility. Errors during completion indication may cause duplicate, but not lost, messages.

In most cases, messages go directly from the mail originator's machine to the destination machine over a single TCP connection. However, mail will occasionally go through intermediate machines via an SMTP forwarding capability, in which case the message must traverse a series of TCP connections between source and destination. One way for this to happen is for the sender to specify a route to the destination in the form of a sequence of servers. A more common event is forwarding required because a user has moved.

It is important to note that the SMTP protocol is limited to the conversation that takes place between the SMTP sender and the SMTP receiver. SMTP's main function is the transfer of messages, although there are some ancillary functions dealing with mail destination verification and handling. The rest of the mail-handling apparatus depicted in Figure 22.1 is beyond the scope of SMTP and may differ from one system to another.

We now turn to a discussion of the main elements of SMTP.

SMTP Overview The operation of SMTP consists of a series of commands and responses exchanged between the SMTP sender and receiver. The initiative is with the SMTP sender, who establishes the TCP connection. Once the connection is established, the SMTP sender sends commands over the connection to the receiver. Each command generates exactly one reply from the SMTP receiver.

Table 22.1 SMTP Commands

Name	Command Form	Description
HELO	HELO <SP> <domain> <CRLF>	Send identification
MAIL	MAIL <SP> FROM:<reverse-path> <CRLF>	Identifies originator of mail
RCPT	RCPT <SP> TO:<forward-path> <CRLF>	Identifies recipient of mail
DATA	DATA <CRLF>	Transfer message text
RSET	RSET <CRLF>	Abort current mail transaction
NOOP	NOOP <CRLF>	No operation
QUIT	QUIT <CRLF>	Close TCP connection
SEND	SEND <SP> FROM:<reverse-path> <CRLF>	Send mail to terminal
SOML	SOML <SP> FROM:<reverse-path> <CRLF>	Send mail to terminal if possible; otherwise to mailbox
SAML	SAML <SP> FROM:<reverse-path> <CRLF>	Send mail to terminal and mailbox
VRFY	VRFY <SP> <string> <CRLF>	Confirm user name
EXPN	EXPN <SP> <string> <CRLF>	Return membership of mailing list
HELP	HELP [<SP> <string>] <CRLF>	Send system-specific documentation
TURN	TURN <CRLF>	Reverse role of sender and receiver

<CRLF> = carriage return, line feed
<SP> = space
Square brackets denote optional elements.
Shaded commands are optional in a conformant SMTP implementation.

Table 22.1 lists the **SMTP commands**. Each command consists of a single line of text, beginning with a four-letter command code followed in some cases by an argument field. Most replies are a single-line, although multiple-line replies are possible. The table indicates those commands that all receivers must be able to recognize. The other commands are optional and may be ignored by the receiver.

SMTP replies are listed in Table 22.2. Each reply begins with a three-digit code and may be followed by additional information. The leading digit indicates the category of the reply:

- **Positive Completion reply:** The requested action has been successfully completed. A new request may be initiated.
- **Positive Intermediate reply:** The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The sender-SMTP should send another command specifying this information. This reply is used in command sequence groups.
- **Transient Negative Completion reply:** The command was not accepted and the requested action did not occur. However, the error condition is temporary and the action may be requested again.
- **Permanent Negative Completion reply:** The command was not accepted and the requested action did not occur.

Table 22.2 SMTP Replies

Code	Description
	Positive Completion Reply
211	System status, or system help reply
214	Help message (Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user)
220	<domain> Service ready
221	<domain> Service closing transmission channel
250	Requested mail action okay, completed
251	User not local; will forward to <forward-path>
	Positive Intermediate Reply
354	Start mail input; end with <CRLF>.<CRLF>
	Transient Negative Completion Reply
421	<domain> Service not available, losing transmission channel (This may be a reply to any command if the service knows it must shut down)
450	Requested mail action not taken: mailbox unavailable (e.g., mailbox busy)
451	Requested action aborted: local error in processing
452	Requested action not taken: insufficient system storage
	Permanent Negative Completion Reply
500	Syntax error, command unrecognized (This may include errors such as command line too long)
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
550	Requested action not taken: mailbox unavailable (e.g., mailbox not found, no access)
551	User not local; please try <forward-path>
552	Requested mail action aborted: exceeded storage allocation
553	Requested action not taken: mailbox name not allowed (e.g., mailbox syntax incorrect)
554	Transaction failed

Basic SMTP operation occurs in three phases: connection setup, exchange of one or more command-response pairs, and connection termination. We examine each phase in turn.

Connection Setup An SMTP sender will attempt to set up a TCP connection with a target host when it has one or more mail messages to deliver to that host. The sequence is quite simple:

1. The sender opens a TCP connection with the receiver.
2. Once the connection is established, the receiver identifies itself with "220 Service Ready."

3. The sender identifies itself with the HELO command.
4. The receiver accepts the sender's identification with "250 OK."

If the mail service on the destination is unavailable, the destination host returns a "421 Service Not Available" reply in step 2 and the process is terminated.

Mail Transfer Once a connection has been established, the SMTP sender may send one or more messages to the SMTP receiver. There are three logical phases to the transfer of a message:

1. A MAIL command identifies the originator of the message.
2. One or more RCPT commands identify the recipients for this message.
3. A DATA command transfers the message text.

The **MAIL command** gives the reverse path, which can be used to report errors. If the receiver is prepared to accept messages from this originator, it returns a "250 OK" reply. Otherwise the receiver returns a reply indicating failure to execute the command (codes 451, 452, 552) or an error in the command (codes 421, 500, 501).

The **RCPT command** identifies an individual recipient of the mail data; multiple recipients are specified by multiple use of this command. A separate reply is returned for each RCPT command, with one of the following possibilities:

1. The receiver accepts the destination with a 250 reply; this indicates that the designated mailbox is on the receiver's system.
2. The destination will require forwarding and the receiver will forward (251).
3. The destination requires forwarding but the receiver will not forward; the sender must resend to the forwarding address (551).
4. A mailbox does not exist for this recipient at this host (550).
5. The destination is rejected due to some other failure to execute (codes 450, 451, 452, 552, 553) or an error in the command (codes 421, 500, 501, 503).

The advantage of using a separate RCPT phase is that the sender will not send the message until it is assured that the receiver is prepared to receive the message for at least one recipient, thereby avoiding the overhead of sending an entire message only to learn that the destination is unknown. Once the SMTP receiver has agreed to receive the mail message for at least one recipient, the SMTP sender uses the **DATA command** to initiate the transfer of the message. If the SMTP receiver is still prepared to receive the message, it returns a 354 message; otherwise the receiver returns a reply indicating failure to execute the command (codes 451, 554) or an error in the command (codes 421, 500, 501, 503). If the 354 reply is returned, the SMTP sender proceeds to send the message over the TCP connection as a sequence of ASCII lines. The end of the message is indicated by a line containing only a period. The SMTP receiver responds with a 250 OK reply if the message is accepted or with the appropriate error code (451, 452, 552, 554).

An example, taken from RFC 821, illustrates the process:

```
S: MAIL FROM:<Smith@Alpha.ARPA>
R: 250 OK

S: RCPT TO:<Jones@Beta.ARPA>
R: 250 OK

S: RCPT TO:<Green@Beta.ARPA>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.ARPA>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

The SMTP sender is transmitting mail that originates with the user Smith@Alpha.ARPA. The message is addressed to three users on machine Beta.ARPA, namely, Jones, Green, and Brown. The SMTP receiver indicates that it has mailboxes for Jones and Brown but does not have information on Green. Because at least one of the intended recipients has been verified, the sender proceeds to send the text message.

Connection Closing The SMTP sender closes the connection in two steps. First, the sender sends a QUIT command and waits for a reply. The second step is to initiate a TCP close operation for the TCP connection. The receiver initiates its TCP close after sending its reply to the QUIT command.

RFC 822 RFC 822 defines a format for text messages that are sent using electronic mail. The SMTP standard adopts RFC 822 as the format for use in constructing messages for transmission via SMTP. In the RFC 822 context, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. The contents comprise the object to be delivered to the recipient. The RFC 822 standard applies only to the contents. However, the content standard includes a set of header fields that may be used by the mail system to create the envelope, and the standard is intended to facilitate the acquisition of such information by programs.

An RFC 822 message consists of a sequence of lines of text and uses a general "memo" framework. That is, a message consists of some number of header lines, which follow a rigid format, followed by a body portion consisting of arbitrary text.

A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are From, To, Subject, and Date. Here is an example message:

```
Date: Tue, 16 Jan 1996 10:37:17 (EST)
From: "William Stallings" <ws@host.com>
Subject: The Syntax in RFC 822
To: Smith@ Other -Host.com
Cc: Jones@ Yet-Another-Host.com

Hello. This section begins the actual message body, which is delimited
from the message heading by a blank line.
```

Another field that is commonly found in RFC 822 headers is Message-ID. This field contains a unique identifier associated with this message.

Multipurpose Internet Mail Extensions (MIME)

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP and RFC 822 for electronic mail. [RODR02] lists the following limitations of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a de facto standard.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail messages over a certain size.
4. SMTP gateways that translate between the character codes ASCII and EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include
   - Deletion, addition, or reordering of carriage return and linefeed
   - Truncating or wrapping lines longer than 76 characters
   - Removal of trailing white space (tab and space characters)
   - Padding of lines in a message to the same length
   - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

Overview The MIME specification includes the following elements:

1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

- **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to the RFCs.
- **Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to present the data to the user or otherwise deal with the data in an appropriate manner.
- **Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
- **Content-ID:** Used to uniquely identify MIME entities in multiple contexts.
- **Content-Description:** A plaintext description of the object with the body; this is useful when the object is not displayable (e.g., audio data).

Any or all of these fields may appear in a normal RFC 822 header. A compliant implementation must support the MIME-Version, Content-Type, and Content-Transfer-Encoding fields; the Content-ID and Content-Description fields are optional and may be ignored by the recipient implementation.

MIME Content Types The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 22.3 lists the MIME content types. There are seven different major types of content and a total of 14 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

For the **text type** of body, no special software is required to get the full meaning of the text, aside from support of the indicated character set. The only defined subtype is plaintext, which is simply a string of ASCII characters or ISO 8859 characters. An earlier version of the MIME specification included a *richtext* subtype, which allows greater formatting flexibility. It is expected that this subtype will reappear in a later RFC.

The **multipart type** indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter, called boundary, that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header.

Table 22.3 MIME Content Types

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN μ-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript
	octet-stream	General binary data consisting of 8-bit bytes.

Here is a simple example of a multipart message, containing two parts both consisting of simple text:

```
From: John Smith <js@company.com>
To:  Ned Jones <ned@soft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary=”simple boundary”

This is the preamble.  It is to be ignored, though it is a handy place for mail
composers to include an explanatory note to non-MIME conformant
readers.
—simple boundary

This is implicitly typed plain ASCII text. It does NOT end with a linebreak.
—simple boundary
Content-type: text/plain; charset=us-ascii

This is explicitly typed plain ASCII text. It DOES end with a linebreak.
—simple boundary—

This is the epilogue.  It is also to be ignored.
```

There are four subtypes of the multipart type, all of which have the same overall syntax. The **multipart/mixed subtype** is used when there are multiple independent body parts that need to be bundled in a particular order. For the **multipart/parallel subtype**, the order of the parts is not significant. If the recipient's system is appropriate, the multiple parts can be presented in parallel. For example, a picture or text part could be accompanied by a voice commentary that is played while the picture or text is displayed.

For the **multipart/alternative subtype**, the various parts are different representations of the same information. The following is an example:

```
From: John Smith <js@company.com>
To:  Ned Jones <ned@soft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary=”boundary42”

—boundary42

Content-Type: text/plain; charset=us-ascii

  ...plaintext version of message goes here....
—boundary42
Content-Type: text/richtext

  .... RFC 1341 richtext version of same message goes here ...
—boundary42—
```

In this subtype, the body parts are ordered in terms of increasing preference. For this example, if the recipient system is capable of displaying the message in the richtext format, this is done; otherwise, the plaintext format is used.

The **multipart/digest subtype** is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e-mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The **message type** provides a number of important capabilities in MIME. The **message/rfc822 subtype** indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but any MIME message.

The **message/partial subtype** enables fragmentation of a large message into a number of parts, which must be reassembled at the destination.

For this subtype, three parameters are specified in the Content-Type: Message/Partial field:

- **id:** A value that is common to each fragment of the same message, so that the fragments can be identified at the recipient for reassembly, but unique across different messages.
- **number:** A sequence number that indicates the position of this fragment in the original message. The first fragment is numbered 1, the second 2, and so on.
- **total:** The total number of parts. The last fragment is identified by having the same value for the *number* and *total* parameters.

The **message/external-body subtype** indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for the encapsulated message.

The Content-Type field in the outer header must include an access-type parameter, which has one of the following values:

- **FTP:** The message body is accessible as a file using the file transfer protocol (FTP). For this access type, the following additional parameters are mandatory: name, the name of the file; and site, the domain name of the host where the file resides. Optional parameters are directory, the directory in which the file is located; and mode, which indicates how FTP should retrieve the file (e.g., ASCII, image). Before the file transfer can take place, the user will need to provide a user id and password. These are not transmitted with the message for security reasons.
- **TFTP:** The message body is accessible as a file using the trivial file transfer protocol (TFTP). The same parameters as for FTP are used, and the user id and password must also be supplied.
- **Anon-FTP:** Identical to FTP, except that the user is not asked to supply a user id and password. The parameter name supplies the name of the file.
- **local-file:** The message body is accessible as a file on the recipient's machine.
- **AFS:** The message body is accessible as a file via the global AFS (Andrew File System). The parameter name supplies the name of the file.
- **mail-server:** The message body is accessible by sending an e-mail message to a mail server. A *server* parameter must be included that gives the e-mail address of the server. The body of the original message, known as the phantom body, should contain the exact command to be sent to the mail server.

The **image type** indicates that the body contains a displayable image. The subtype, jpeg or gif, specifies the image format. In the future, more subtypes will be added to this list.

The **video type** indicates that the body contains a time-varying picture image, possibly with color and coordinated sound. The only subtype so far specified is mpeg.

The **audio type** indicates that the body contains audio data. The only subtype, basic, conforms to an ISDN service known as "64-kbps, 8-kHz Structured, Usable for Speech Information," with a digitized speech algorithm referred to as μ-law PCM (pulse code modulation). This general type is the typical way of transmitting speech signals over a digital network. The term μ-law refers to the specific encoding technique; it is the standard technique used in North America and Japan. A competing system, known as A-law, is standard in Europe.

The **application type** refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application. The **application/octet-stream subtype** indicates general binary data in a sequence of octets. RFC 2045 recommends that the receiving implementation should offer to put the data in a file or use the data as input to a program.

The **application/Postscript subtype** indicates the use of Adobe Postscript.

MIME Transfer Encodings The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values, as listed in Table 22.4. However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. This could be a vendor-specific or application-specific scheme. The two actual encoding schemes defined are quoted-printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable, and one that is safe for all types of data in a way that is reasonably compact.

The **quoted-printable** transfer encoding is useful when the data consist largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces

Table 22.4 MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

reversible (soft) line breaks to limit message lines to 76 characters. The encoding rules are as follows:

1. **General 8-bit representation:** This rule is to be used when none of the other rules apply. Any character is represented by an equal sign followed by a two-digit hexadecimal representation of the octet's value. For example, the ASCII form feed, which has an 8-bit value of decimal 12, is represented by "=0C."
2. **Literal representation:** Any character in the range decimal 33 ("!") through decimal 126 ("~"), except decimal 61 ("="), is represented as that ASCII character.
3. **White space:** Octets with the values 9 and 32 may be represented as ASCII tab and space characters, respectively, except at the end of a line. Any white space (tab or blank) at the end of a line must be represented by rule 1. On decoding, any trailing white space on a line is deleted. This eliminates any white space added by intermediate transport agents.
4. **Line breaks:** Any line break, regardless of its initial representation, is represented by the RFC 822 line break, which is a carriage-return/line-feed combination.
5. **Soft line breaks:** If an encoded line would be longer than 76 characters (excluding <CRLF>), a soft line break must be inserted at or before character position 75. A soft line break consists of the hexadecimal sequence 3D0D0A, which is the ASCII code for an equal sign followed by carriage return, line feed.

The **base64 transfer encoding**, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs. This technique maps arbitrary binary input into printable character output. The form of encoding has the following relevant characteristics:

1. The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set. Thus, the characters themselves can be encoded into whatever form is needed by a specific system. For example, the character "E" is represented in an ASCII-based system as hexadecimal 45 and in an EBCDIC-based system as hexadecimal C5.
2. The character set consists of 65 printable characters, one of which is used for padding. With $2^6 = 64$ available characters, each character can be used to represent 6 bits of input.
3. No control characters are included in the set. Thus, a message encoded in radix 64 can traverse mail handling systems that scan the data stream for control characters.
4. The hyphen character ("-") is not used. This character has significance in the RFC 822 format and should therefore be avoided.

Table 22.5 shows the mapping of 6-bit input values to characters. The character set consists of the alphanumeric characters plus "+" and "/". The "=" character is used as the padding character.

Figure 22.2 illustrates the simple mapping scheme. Binary input is processed in blocks of 3 octets, or 24 bits. Each set of 6 bits in the 24-bit block is mapped into a character. In the figure, the characters are shown encoded as 8-bit quantities. In this typical case, each 24-bit input is expanded to 32 bits of output.

Table 22.5 Radix-64 Encoding

6-Bit Value	Character Encoding	6-Bit Value	Character Encoding	6-Bit Value	Character Encoding	6-Bit Value	Character Encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

For example, consider the 24-bit raw text sequence 00100011 01011100 10010001, which can be expressed in hexadecimal as 235C91. We arrange this input in blocks of 6 bits:

001000 110101 110010 010001

The extracted 6-bit decimal values are 8, 53, 50, and 17. Looking these up in Table 22.5 yields the radix-64 encoding as the following characters: I1yR. If these characters are stored in 8-bit ASCII format with parity bit set to zero, we have

01001001 00110001 01111001 01010010

In hexadecimal, this is 49317952. To summarize,

Input Data	
Binary representation	00100011 01011100 10010001
Hexadecimal representation	235C91
Radix-64 Encoding of Input Data	
Character representation	I1yR
ASCII code (8 bit, zero parity)	01001001 00110001 01111001 01010010
Hexadecimal representation	49317952

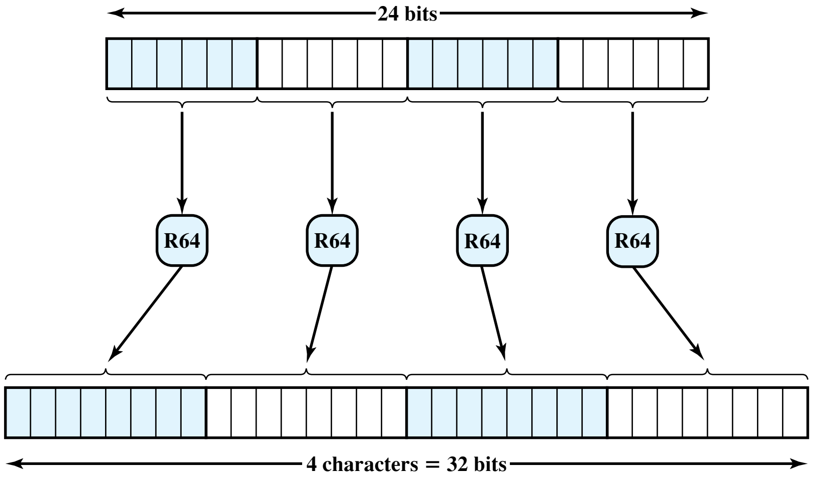

Figure 22.2 Printable Encoding of Binary Data into Radix-64 Format

22.2 NETWORK MANAGEMENT—SNMP

Networks and distributed processing systems are of critical and growing importance in business, government, and other organizations. Within a given organization, the trend is toward larger, more complex networks supporting more applications and more users. As these networks grow in scale, two facts become painfully evident:

- The network and its associated resources and distributed applications become indispensable to the organization.
- More things can go wrong, disabling the network or a portion of the network or degrading performance to an unacceptable level.

A large, reliable network cannot be put together and managed by human effort alone. The complexity of such a system dictates the use of automated network management tools. The urgency of the need for such tools is increased, and the difficulty of supplying such tools is also increased, if the network includes equipment from multiple vendors. In response, standards that deal with network management have been developed, covering services, protocols, and management information base.

This section begins with an introduction to the overall concepts of standardized network management. The remainder of the section is devoted to a discussion of SNMP, the most widely used network management standard.

Network Management Systems

A network management system is a collection of tools for network monitoring and control that is integrated in the following senses:

- A single operator interface with a powerful but user-friendly set of commands for performing most or all network management tasks.

- A minimal amount of additional equipment. That is, most of the hardware and software required for network management is incorporated into the existing user equipment.

A network management system consists of incremental hardware and software additions implemented among existing network components. The software used in accomplishing the network management tasks resides in the host computers and communications processors (e.g., networks switches, routers). A network management system is designed to view the entire network as a unified architecture, with addresses and labels assigned to each point and the specific attributes of each element and link known to the system. The active elements of the network provide regular feedback of status information to the network control center.

Simple Network Management Protocol Version 1 (SNMPv1)

SNMP was developed for use as a network management tool for networks and internetworks operating TCP/IP. It has since been expanded for use in all types of networking environments. The term *simple network management protocol (SNMP)* is actually used to refer to a collection of specifications for network management that include the protocol itself, the definition of a database, and associated concepts.

Basic Concepts The model of network management that is used for SNMP includes the following key elements:

- Management station, or manager
- Agent
- Management information base
- Network management protocol

The **management station** is typically a standalone device, but may be a capability implemented on a shared system. In either case, the management station serves as the interface for the human network manager into the network management system. The management station will have, at minimum,

- A set of management applications for data analysis, fault recovery, and so on
- An interface by which the network manager may monitor and control the network
- The capability of translating the network manager's requirements into the actual monitoring and control of remote elements in the network
- A database of network management information extracted from the databases of all the managed entities in the network

Only the last two elements are the subject of SNMP standardization.

The other active element in the network management system is the **management agent**. Key platforms, such as hosts, bridges, routers, and hubs, may be equipped with agent software so that they may be managed from a management station. The agent responds to requests for information from a management station, responds to requests for actions from the management station, and may asynchronously provide the management station with important but unsolicited information.

To manage resources in the network, each resource is represented as an object. An object is, essentially, a data variable that represents one aspect of the managed agent. The collection of objects is referred to as a **management information base** (MIB). The MIB functions as a collection of access points at the agent for the management station. These objects are standardized across systems of a particular class (e.g., bridges all support the same management objects). A management station performs the monitoring function by retrieving the value of MIB objects. A management station can cause an action to take place at an agent or can change the configuration settings of an agent by modifying the value of specific variables.

The management station and agents are linked by a **network management protocol**. The protocol used for the management of TCP/IP networks is the Simple Network Management Protocol (SNMP). An enhanced version of SNMP, known as SNMPv2, is intended for both TCP/IP- and OSI-based networks. Each of these protocols includes the following key capabilities:

- **Get:** Enables the management station to retrieve the value of objects at the agent
- **Set:** Enables the management station to set the value of objects at the agent
- **Notify:** Enables an agent to send unsolicited notifications to the management station of significant events

In a traditional centralized network management scheme, one host in the configuration has the role of a network management station; there may be one or two other management stations in a backup role. The remainder of the devices on the network contain agent software and a MIB, to allow monitoring and control from the management station. As networks grow in size and traffic load, such a centralized system is unworkable. Too much burden is placed on the management station, and there is too much traffic, with reports from every single agent having to wend their way across the entire network to headquarters. In such circumstances, a decentralized, distributed approach works best (e.g., Figure 22.3). In a decentralized network management scheme, there may be multiple top-level management stations, which might be referred to as management servers. Each such server might directly manage a portion of the total pool of agents. However, for many of the agents, the management server delegates responsibility to an intermediate manager. The intermediate manager plays the role of manager to monitor and control the agents under its responsibility. It also plays an agent role to provide information and accept control from a higher-level management server. This type of architecture spreads the processing burden and reduces total network traffic.

Network Management Protocol Architecture SNMP is an application-level protocol that is part of the TCP/IP protocol suite. It is intended to operate over the user datagram protocol (UDP). Figure 22.4 suggests the typical configuration of protocols for SNMPv1. For a standalone management station, a manager process controls access to a central MIB at the management station and provides an interface to the network manager. The manager process achieves network management

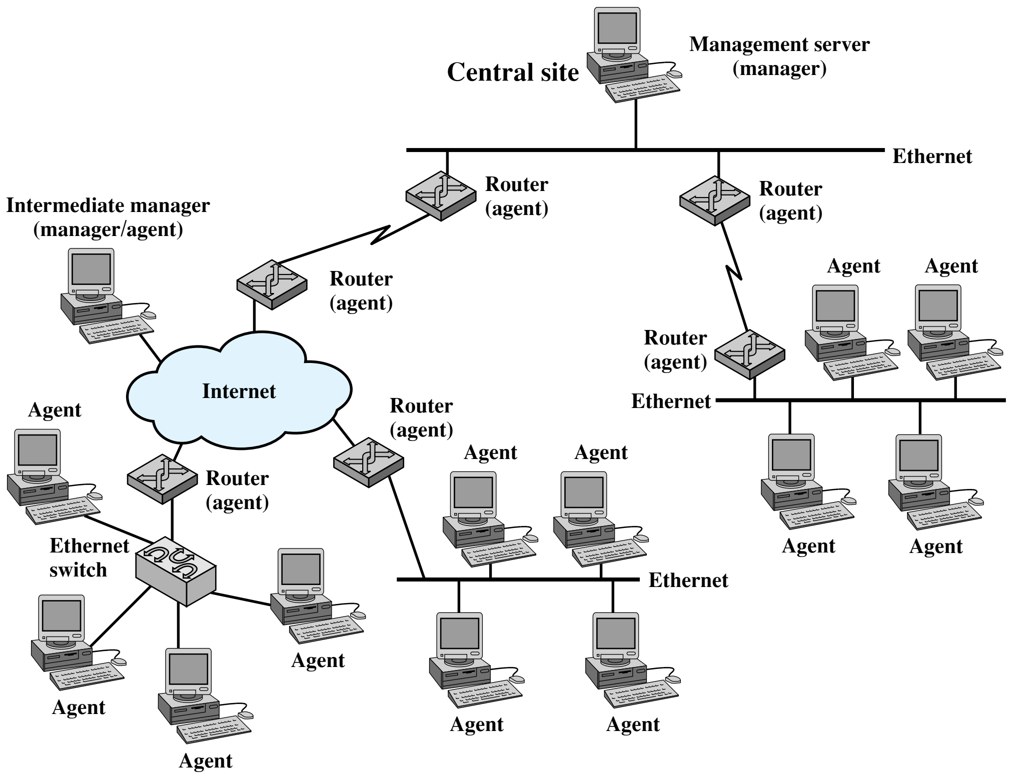

Figure 22.3 Example Distributed Network Management Configuration

by using SNMP, which is implemented on top of UDP, IP, and the relevant network-dependent protocols (e.g., Ethernet, ATM, frame relay).

Each agent must also implement SNMP, UDP, and IP. In addition, there is an agent process that interprets the SNMP messages and controls the agent's MIB. For an agent device that supports other applications, such as FTP, TCP as well as UDP is required. In Figure 22.4, the shaded portions depict the operational environment: that which is to be managed. The unshaded portions provide support to the network management function.

Figure 22.5 provides a somewhat closer look at the protocol context of SNMP. From a management station, three types of SNMP messages are issued on behalf of a management applications: GetRequest, GetNextRequest, and SetRequest. The first two are two variations of the get function. All three messages are acknowledged by the agent in the form of a GetResponse message, which is passed up to the management application. In addition, an agent may issue a trap message in response to an event that affects the MIB and the underlying managed resources. Management requests are sent to UDP port 161, while the agent sends traps to UDP port 162.

Because SNMP relies on UDP, which is a connectionless protocol, SNMP is itself connectionless. No ongoing connections are maintained between a management station and its agents. Instead, each exchange is a separate transaction between a management station and an agent.

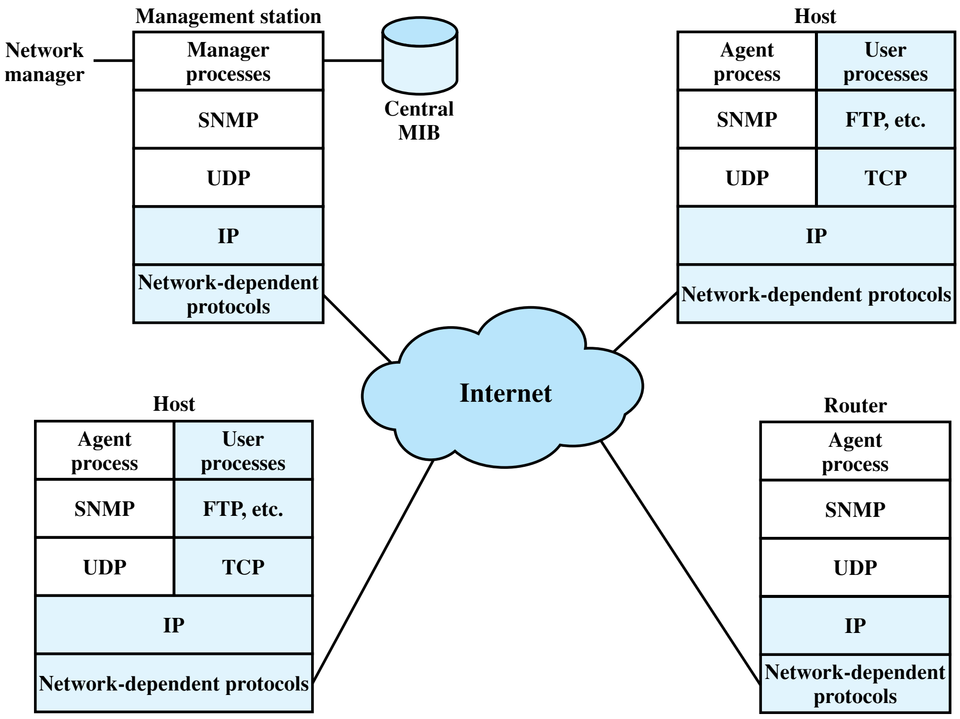

Figure 22.4 SNMPv1 Configuration

Simple Network Management Protocol Version 2 (SNMPv2)

In August of 1988, the specification for SNMP was issued and rapidly became the dominant network management standard. A number of vendors offer standalone network management workstations based on SNMP, and most vendors of bridges, routers, workstations, and PCs offer SNMP agent packages that allow their products to be managed by an SNMP management station.

As the name suggests, SNMP is a simple tool for network management. It defines a limited, easily implemented MIB of scalar variables and two-dimensional tables, and it defines a streamlined protocol to enable a manager to get and set MIB variables and to enable an agent to issue unsolicited notifications, called *traps*. This simplicity is the strength of SNMP. SNMP is easily implemented and consumes modest processor and network resources. Also, the structure of the protocol and the MIB are sufficiently straightforward that it is not difficult to achieve interoperability among management stations and agent software from a mix of vendors.

With its widespread use, the deficiencies of SNMP became increasingly apparent; these include both functional deficiencies and a lack of a security facility. As a result, an enhanced version, known as SNMPv2, was issued (RFCs 1901, 1905 through 1909, and 2578 through 2580). SNMPv2 has quickly gained support, and a number of vendors announced products within months of the issuance of the standard.

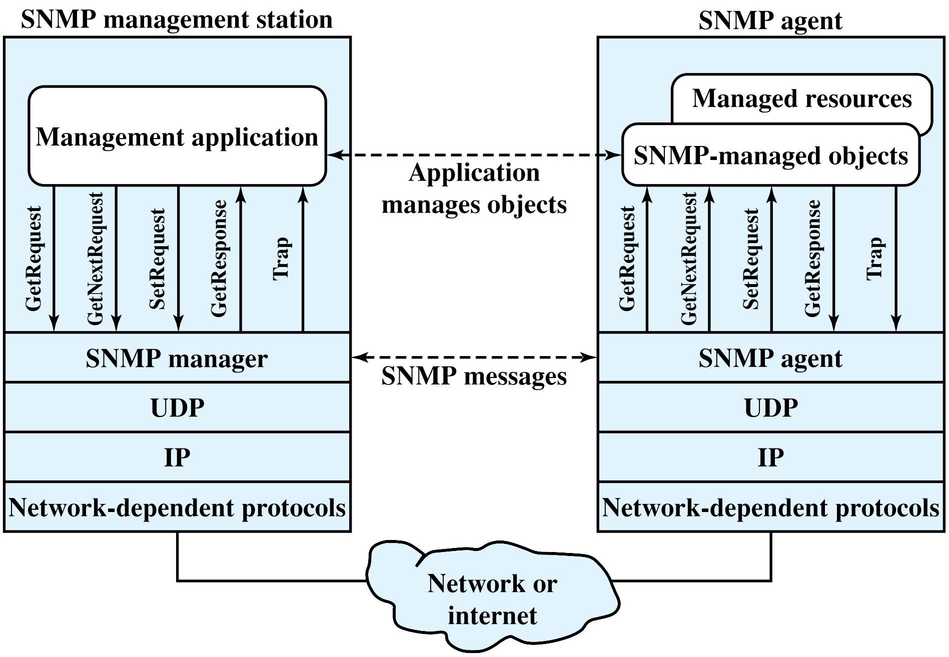

Figure 22.5 The Role of SNMPv1

The Elements of SNMPv2

As with SNMPv1, SNMPv2 provides a framework on which network management applications can be built. Those applications, such as fault management, performance monitoring, accounting, and so on, are outside the scope of the standard.

SNMPv2 provides the infrastructure for network management. Figure 22.6 is an example of a configuration that illustrates that infrastructure.

The essence of SNMPv2 is a protocol that is used to exchange management information. Each "player" in the network management system maintains a local database of information relevant to network management, known as the MIB. The SNMPv2 standard defines the structure of this information and the allowable data types; this definition is known as the structure of management information (SMI). We can think of this as the language for defining management information. The standard also supplies a number of MIBs that are generally useful for network management.[1] In addition, new MIBs may be defined by vendors and user groups.

At least one system in the configuration must be responsible for network management. It is here that any network management applications are hosted. There may be more than one of these management stations, to provide redundancy or simply to split up the duties in a large network. Most other systems act in the role of agent. An agent collects information locally and stores it for later access by a

[1]There is a slight fuzziness about the term *MIB*. In its singular form, the term *MIB* can be used to refer to the entire database of management information at a manager or an agent. It can also be used in singular or plural form to refer to a specific defined collection of management information that is part of an overall MIB. Thus, the SNMPv2 standard includes the definition of several MIBs and incorporates, by reference, MIBs defined in SNMPv1.

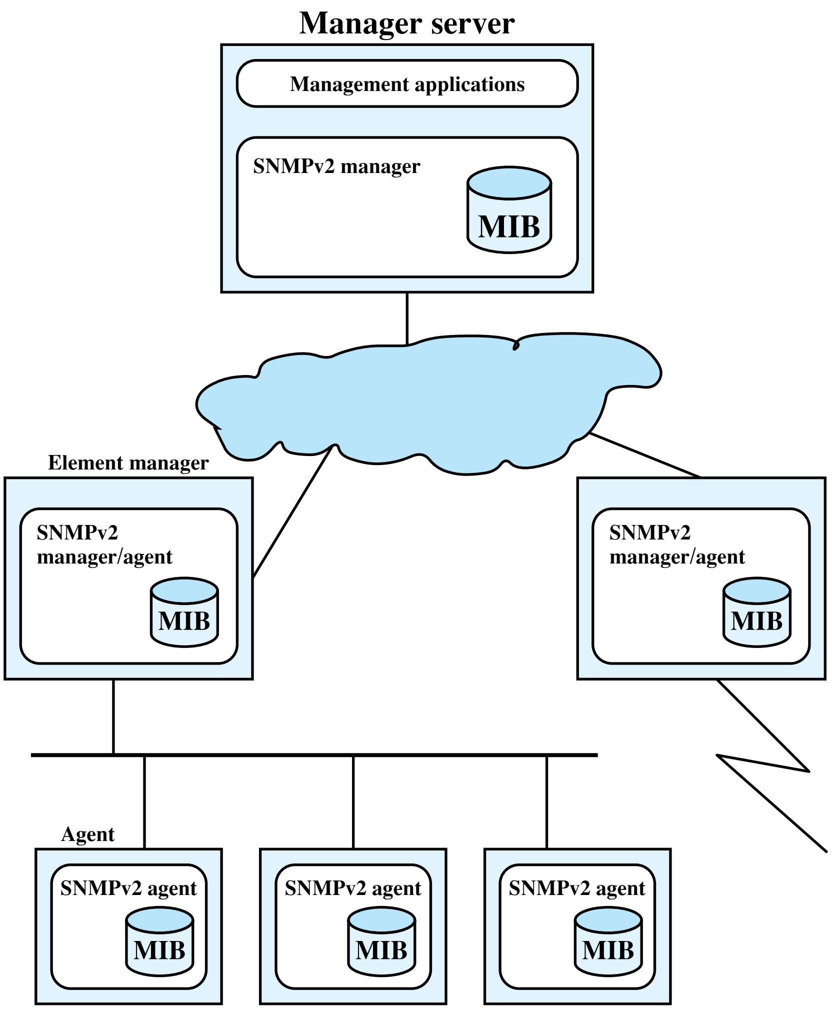

Figure 22.6 SNMPv2-Managed Configuration

manager. The information includes data about the system itself and may also include traffic information for the network or networks to which the agent attaches.

SNMPv2 supports either a highly centralized network management strategy or a distributed one. In the latter case, some systems operate both in the role of manager and of agent. In its agent role, such a system will accept commands from a superior management system. Some of those commands relate to the local MIB at the agent. Other commands require the agent to act as a proxy for remote devices. In this case, the proxy agent assumes the role of manager to access information at a remote agent, and then assumes the role of an agent to pass that information on to a superior manager.

All of these exchanges take place using the SNMPv2 protocol, which is a simple request/response type of protocol. Typically, SNMPv2 is implemented on top of the user datagram protocol (UDP), which is part of the TCP/IP suite. Because

SNMPv2 exchanges are in the nature of discrete request-response pairs, an ongoing reliable connection is not required.

Structure of Management Information The SMI defines the general framework within which a MIB can be defined and constructed. The SMI identifies the data types that can be used in the MIB, and how resources within the MIB are represented and named. The philosophy behind SMI is to encourage simplicity and extensibility within the MIB. Thus, the MIB can store only simple data types: scalars and two-dimensional arrays of scalars, called tables. The SMI does not support the creation or retrieval of complex data structures. This philosophy is in contrast to that used with OSI systems management, which provides for complex data structures and retrieval modes to support greater functionality. SMI avoids complex data types and structures to simplify the task of implementation and to enhance interoperability. MIBs will inevitably contain vendor-created data types and, unless tight restrictions are placed on the definition of such data types, interoperability will suffer.

There are three key elements in the SMI specification. At the lowest level, the SMI specifies the data types that may be stored. Then the SMI specifies a formal technique for defining objects and tables of objects. Finally, the SMI provides a scheme for associating a unique identifier with each actual object in a system, so that data at an agent can be referenced by a manager.

Table 22.6 shows the data types that are allowed by the SMI. This is a fairly restricted set of types. For example, real numbers are not supported. However, it is rich enough to support most network management requirements.

Protocol Operation The heart of the SNMPv2 framework is the protocol itself. The protocol provides a straightforward, basic mechanism for the exchange of management information between manager and agent.

Table 22.6 Allowable Data Types in SNMPv2

Data Type	Description
INTEGER	Integers in the range of -2^{31} to $2^{31} - 1$.
UInteger32	Integers in the range of 0 to $2^{32} - 1$.
Counter32	A nonnegative integer that may be incremented modulo 2^{32}.
Counter64	A nonnegative integer that may be incremented modulo 2^{64}.
Gauge32	A nonnegative integer that may increase or decrease, but shall not exceed a maximum value. The maximum value can not be greater than $2^{32} - 1$.
TimeTicks	A nonnegative integer that represents the time, modulo 2^{32}, in hundredths of a second.
OCTET STRING	Octet strings for arbitrary binary or textual data; may be limited to 255 octets.
IpAddress	A 32-bit internet address.
Opaque	An arbitrary bit field.
BIT STRING	An enumeration of named bits.
OBJECT IDENTIFIER	Administratively assigned name to object or other standardized element. Value is a sequence of up to 128 nonnegative integers.

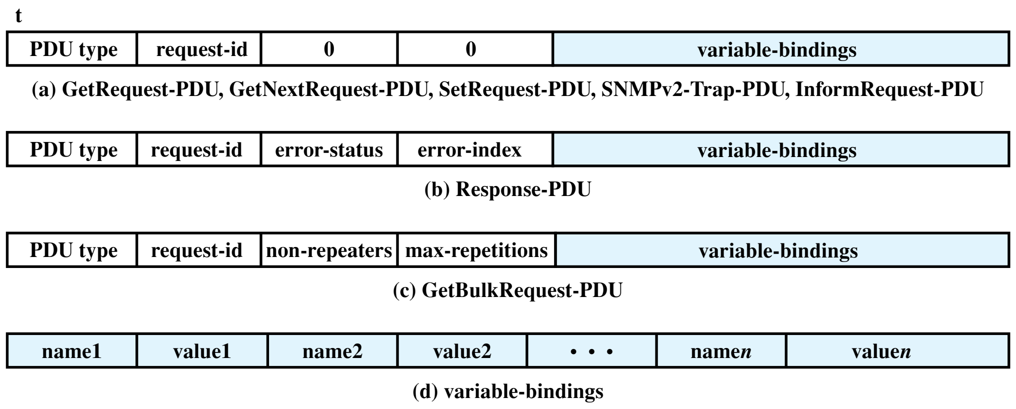

Figure 22.7 SNMPv2 PDU Format

The basic unit of exchange is the message, which consists of an outer message wrapper and an inner protocol data unit (PDU). The outer message header deals with security and is discussed later in this section.

Seven types of PDUs may be carried in an SNMP message. The general formats for these are illustrated informally in Figure 22.7. Several fields are common to a number of PDUs. The request-id field is an integer assigned such that each outstanding request can be uniquely identified. This enables a manager to correlate incoming responses with outstanding requests. It also enables an agent to cope with duplicate PDUs generated by an unreliable transport service. The variable-bindings field contains a list of object identifiers; depending on the PDU, the list may also include a value for each object.

The GetRequest-PDU, issued by a manager, includes a list of one or more object names for which values are requested. If the get operation is successful, then the responding agent will send a Response-PDU. The variable-bindings list will contain the identifier and value of all retrieved objects. For any variables that are not in the relevant MIB view, its identifier and an error code are returned in the variable-bindings list. Thus, SNMPv2 permits partial responses to a GetRequest, which is a significant improvement over SNMP. In SNMP, if one or more of the variables in a GetRequest is not supported, the agent returns an error message with a status of noSuchName. To cope with such an error, the SNMP manager must either return no values to the requesting application, or it must include an algorithm that responds to an error by removing the missing variables, resending the request, and then sending a partial result to the application.

The GetNextRequest-PDU also is issued by a manager and includes a list of one or more objects. In this case, for each object named in the variable-bindings field, a value is to be returned for the object that is next in lexicographic order, which is equivalent to saying next in the MIB in terms of its position in the tree structure of object identifiers. As with the GetRequest-PDU, the agent will return values for as many variables as possible. One of the strengths of the GetNextRequest-PDU is that it enables a manager entity to discover the structure of a MIB view dynamically. This is useful if the manager does not know a priori the set of objects that are supported by an agent or that are in a particular MIB view.

One of the major enhancements provided in SNMPv2 is the GetBulkRequest PDU. The purpose of this PDU is to minimize the number of protocol exchanges required to retrieve a large amount of management information. The GetBulkRequest PDU allows an SNMPv2 manager to request that the response be as large as possible given the constraints on message size.

The SetRequest-PDU is issued by a manager to request that the values of one or more objects be altered. The receiving SNMPv2 entity responds with a Response-PDU containing the same request-id. The SetRequest operation is atomic: Either all of the variables are updated or none are. If the responding entity is able to set values for all of the variables listed in the incoming variable-bindings list, then the Response-PDU includes the variable-bindings field, with a value supplied for each variable. If at least one of the variable values cannot be supplied, then no values are returned, and no values are updated. In the latter case, the error-status code indicates the reason for the failure, and the error-index field indicates the variable in the variable-bindings list that caused the failure.

The SNMPv2-Trap-PDU is generated and transmitted by an SNMPv2 entity acting in an agent role when an unusual event occurs. It is used to provide the management station with an asynchronous notification of some significant event. The variable-bindings list is used to contain the information associated with the trap message. Unlike the GetRequest, GetNextRequest, GetBulkRequest, SetRequest, and InformRequest-PDUs, the SNMPv2-Trap-PDU does not elicit a response from the receiving entity; it is an unconfirmed message.

The InformRequest-PDU is sent by an SNMPv2 entity acting in a manager role, on behalf of an application, to another SNMPv2 entity acting in a manager role, to provide management information to an application using the latter entity. As with the SNMPv2-Trap-PDU, the variable-bindings field is used to convey the associated information. The manager receiving an InformRequest acknowledges receipt with a Response-PDU.

For both the SNMPv2-Trap and the InformRequest, various conditions can be defined that indicate when the notification is generated; the information to be sent is also specified.

Simple Network Management Protocol Version 3 (SNMPv3)

Many of the functional deficiencies of SNMP were addressed in SNMPv2. To correct the security deficiencies of SNMPv1/v2, SNMPv3 was issued as a set of Proposed Standards in January 1998 (currently RFCs 2570 through 2575). This set of documents does not provide a complete SNMP capability but rather defines an overall SNMP architecture and a set of security capabilities. These are intended to be used with the existing SNMPv2.

SNMPv3 provides three important services: authentication, privacy, and access control. The first two are part of the User-Based Security model (USM), and the last is defined in the View-Based Access Control Model (VACM). Security services are governed by the identity of the user requesting the service; this identity is expressed as a principal, which may be an individual or an application or a group of individuals or applications.

The authentication mechanism in USM assures that a received message was transmitted by the principal whose identifier appears as the source in the message header. This mechanism also assures that the message has not been altered in transit and has not been artificially delayed or replayed. The sending principal provides authentication by including a message authentication code with the SNMP message it is sending. This code is a function of the contents of the message, the identity of the sending and receiving parties, the time of transmission, and a secret key that should be known only to sender and receiver. The secret key must be set up outside of USM as a configuration function. That is, the configuration manager or network manager is responsible for distributing secret keys to be loaded into the databases of the various SNMP managers and agents. This can be done manually or using some form of secure data transfer outside of USM. When the receiving principal gets the message, it uses the same secret key to calculate the message authentication code once again. If the receiver's version of the code matches the value appended to the incoming message, then the receiver knows that the message can only have originated from the authorized manager and that the message was not altered in transit. The shared secret key between sending and receiving parties must be preconfigured. The actual authentication code used is known as HMAC, which is an Internet-standard authentication mechanism.

The privacy facility of USM enables managers and agents to encrypt messages. Again, manager principal and agent principal must share a secret key. In this case, if the two are configured to use the privacy facility, all traffic between them is encrypted using the Data Encryption Standard (DES). The sending principal encrypts the message using the DES algorithm and its secret key and sends the message to the receiving principal, which decrypts it using the DES algorithm and the same secret key.

The access control facility makes it possible to configure agents to provide different levels of access to the agent's MIB to different managers. An agent principal can restrict access to its MIB for a particular manager principal in two ways. First, it can restrict access to a certain portion of its MIB. For example, an agent may restrict most manager parties to viewing performance-related statistics and only allow a single designated manager principal to view and update configuration parameters. Second, the agent can limit the operations that a manager can use on that portion of the MIB. For example, a particular manager principal could be limited to read-only access to a portion of an agent's MIB. The access control policy to be used by an agent for each manager must be preconfigured and essentially consists of a table that detail the access privileges of the various authorized managers.

22.3 RECOMMENDED READING AND WEB SITES

[KHAR98] provides an overview of SMTP. [ROSE98] provides a book-length treatment of electronic mail, including some coverage of SMTP and MIME. [STAL99] provides a comprehensive and detailed examination of SNMP, SNMPv2, and SNMPv3; the book also provides an overview of network management technology. One of the few textbooks on the subject of network management is [SUBR00].

KHAR98c Khare, R. “The Spec’s in the Mail.” *IEEE Internet Computing*, September/October 1998.

ROSE98 Rose, M., and Strom, D. *Internet Messaging: From the Desktop to the Enterprise.* Upper Saddle River, NJ: Prentice Hall, 1998.

STAL99 Stallings, W. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Reading, MA: Addison-Wesley, 1999.

SUBR00 Subranamian, M. *Network Management: Principles and Practice.* Reading, MA: Addison-Wesley, 2000.

Recommended Web sites:

- **SMTP/MIME RFCS:** A complete list, maintained by IETF.
- **Simple Web Site:** Maintained by the University of Twente. It is a good source of information on SNMP, including pointers to many public-domain implementations and lists of books and articles.

22.4 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

agent	management station	radix-64 encoding
base64	Multipurpose Internet Mail Extension (MIME)	Simple Mail Transfer Protocol (SMTP)
electronic mail	network management protocol	Simple Network Management Protocol (SNMP)
management information base (MIB)	network management system	

Review Questions

22.1 What is the difference between the RFC 821 and RFC 822?
22.2 What are the SMTP and MIME standards?
22.3 What is the difference between a MIME content type and a MIME transfer encoding?
22.4 Briefly explain radix-64 encoding.
22.5 What is a network management system?
22.6 List and briefly define the key elements of SNMP.
22.7 What functions are provided by SNMP?
22.8 What are the differences among SNMPv1, SNMPv2, and SNMPv3?

Problems

22.1 Electronic mail systems differ in the manner in which multiple recipients are handled. In some systems, the originating user agent or mail sender makes all the necessary copies and these are sent out independently. An alternative approach is to determine the route for each destination first. Then a single message is sent out on a common portion of the route and copies are only made when the routes diverge; this process is referred to as mail-bagging. Discuss the relative advantages and disadvantages of the two methods.

22.2 Excluding the connection establishment and termination, what is the minimum number of network round trips to send a small email message using SMTP?

22.3 Explain the differences between the intended use for the quoted-printable and Base64 encodings

22.4 Suppose you need to send one message to three different users: user1@example.com, user2@example.com, and user3@example.com. Is there any difference between sending one separate message per user and sending only one message with multiple (three) recipients? Explain.

22.5 We've seen that the character sequence "<CR><LF>.<CR><LF>" indicates the end of mail data to a SMTP-server. What happens if the mail data itself contains that character sequence?

22.6 Users are free to define and use additional header fields other than the ones defined in RFC 822. Such header fields must begin with the string "X-". Why?

22.7 Suppose you find some technical problems with the mail account user@example.com. Who should you try to contact in order to solve them?

22.8 Although TCP is a full-duplex protocol, SMTP uses TCP in a half-duplex fashion. The client sends a command and then stops and waits for the reply. How can this half-duplex operation fool the TCP slow start mechanism when the network is running near capacity?

22.9 The original (version 1) specification of SNMP has the following definition of a new type:

Gauge ::= [APPLICATION 2] IMPLICIT INTEGER (0..4294967295)

The standard includes the following explanation of the semantics of this type:

> This application-wide type represents a non-negative integer, which may increase or decrease, but which latches at a maximum value. This standard specifies a maximum value of $2^{32} - 1$ (4294967295 decimal) for gauges.

Unfortunately, the word *latch* is not defined, and this has resulted in two different interpretations. The SNMPv2 standard cleared up the ambiguity with the following definition:

> The value of a Gauge has its maximum value whenever the information being modeled is greater than or equal to that maximum value; if the information being modeled subsequently decreases below the maximum value, the Gauge also decreases.

a. What is the alternative interpretation?
b. Discuss the pros and cons of the two interpretations.

22.10 Because SNMP uses two different port numbers (UDP ports 161 and 162), a single system can easily run both a manager and an agent. What would happen if the same port number were used for both?