- characters AFKP using the public and private keys you derived in Exercise 13.25. Remember the limitation imposed by the choice of prime numbers.

Section 13.5

13.27 With the aid of a diagram, show how nonrepudiation can be obtained using the RSA algorithm:
(i) on the complete message,
(ii) on a digest of the message.
Clearly identify on your diagram the keys used and the encrypted/decrypted values at each stage.

Section 13.6

13.28 With the aid of a diagram, explain how user authentication can be carried out using a public key scheme. Include in your diagram the contents of each message and the key used to encrypt the message.

13.29 Explain the meaning and use of the following terms in relation to the authentication procedure associated with Kerberos:
(i) tickets,
(ii) ticket granting server,
(iii) nonce,
(iv) authentication server,
(v) authentication database,
(vi) authenticator.

13.30 With the aid of a diagram, identify a possible security threat that can occur with a public key system when the public key is made readily available. Describe how a certification authority can be used to overcome this threat. Include in your description a list of the fields that are present in the certificate and their use.

Internet applications

14.1 Introduction

As we showed in Figure 12.1 and explained in the accompanying text, in the TCP/IP protocol suite, given the IP address and port number of a destination application protocol/process (AP), the services provided by TCP or UDP enable two (or more) peer APs to communicate with each other in a transparent way. That is, it does not matter whether the correspondent AP(s) is(are) running in the same computer, another computer on the same network, or another computer attached to a network on the other side of the world. Also, since neither TCP nor UDP examines the content of the information being transferred, this can be a control message (PDU) associated with the application protocol, a file of characters from a selected character set, or a string of bytes output by a particular audio or video codec. Hence application protocols are concerned only with, firstly, ensuring the PDUs associated with the protocol are in the defined format and are exchanged in the specified sequence and secondly, the information/data being transferred is in an agreed transfer syntax so that it has the same meaning to each of the applications.

In this chapter, we discuss both the role and operation of a selection of the application protocols associated with the Internet. These are the simple

(electronic) mail transfer protocol (SMTP) and the related multipurpose Internet mail extensions (MIME) protocol, the file transfer protocol (FTP) and a simpler version of this (Trivial FTP), and Internet telephony. In addition, we describe two protocols which, in many instances, a user of the Internet is unaware of. The first is invoked every time we use the Internet and is called the Domain Name System (DNS). The second is concerned with the management of the various networking devices that make up the Internet and is called the **simple network management protocol** (**SNMP**). Because of its role, we shall describe the DNS protocol first.

14.2 Domain name system

As we saw in Figure 12.1 and its accompanying text, an application protocol/process (AP) communicates with a correspondent AP using the latter's IP address and port number. The IP address of the destination AP is first used to route a message – contained within one or more datagrams – across the Internet to the required host and the port number is then used within the host protocol stack to route the received message to the required destination AP. As we saw, however, in the TCP/IP protocol suite the port number of a server AP is allocated a well-known port number which is known by all the client APs that communicate with it. Hence in order to communicate with a remote AP, the source AP need only know the IP address of the host in which the AP is running. Nevertheless, even if this is represented in dotted decimal, it can require up to 12 decimal digits to be remembered, with many more for an IPv6 address. To avoid users from having to cope with such numbers, a directory service similar to that used with a PSTN is used. This is called the **Domain Name System** (**DNS**) and it enables each host computer attached to the Internet to be allocated a **symbolic name** in addition to an IP address.

There are many millions of hosts attached to the Internet each of which has a unique IP address assigned to it. Each host, therefore, must also have a unique name assigned to it and hence an efficient naming scheme is a major part of the DNS. In addition, given a symbolic name, this must be mapped into the related IP address before any communication can take place. This procedure is called name-to-address mapping and is part of the DNS. As we indicated in the introduction, this must be done every time a network application is run and hence it is essential that the procedure is carried out in an efficient way. We shall limit our discussion of the DNS to these two components. They are defined in **RFCs 1034** and **1035**.

14.2.1 Name structure and administration

All the data in the DNS constitutes what is called the **domain name space** and its contents are indexed by a name. The structure of the name space is important since it strongly influences the efficiency of both the administration of

the name space and the subsequent address resolution operation. Basically there are two approaches. One is to adopt a **flat structure** and the other a **hierarchical structure**. Although a flat structure uses the overall name space more efficiently, the resulting DNS must be administered centrally. Also, since in a large network like the Internet multiple copies of the DNS are required to speed up the name-to-address mapping operation, using a flat structure would mean that all copies of the DNS would need to be updated each time a change occurred. For these reasons, the domain name space uses a hierarchical naming structure.

In terms of the administration of the name space, the advantages of using a hierarchical structure can best be seen by considering the structure and assignment of subscriber numbers in the telephone system. At the highest level there is a country code, followed by an area code within that country, and so on. The assignment of numbers can be administered in a distributed rather than a centralized way. The assignment of country codes is administered at an international level, the assignment of area codes within each country at a national level, and so on, down to the point where the assignment of numbers within a local area can be administered within that area. This can be done knowing that as long as each higher-level number is unique within the corresponding level in the address hierarchy, the combined number will be unique within the total address space.

The adoption of a hierarchical structure also means that it is possible to partition the DNS in such a way that most name-to-address mapping operations – and other services – can be carried out locally. For example, if names are assigned according to the geographical location of hosts, then the name space can be partitioned in a similar way. Since most network transactions, and hence requests, are between hosts situated in the same local area – for example, between a community of workstations and a local server or email system – then the majority of service requests can be resolved locally and relatively few referred to another site.

As we show in Figure 14.1, the overall structure of the domain name space is represented in the form of an inverted tree with the single root at the top. The root is called the **root domain** and the top-level branch nodes in the tree, domain nodes or simply **domains**. Each domain has further branches associated with it until, at the lowest level of the tree, is a single host that is attached to the Internet. The names of the top-level domains reflect the historical development of the Internet. Initially, when the Internet spanned just the United States with a small number of international organizations linked to it, at the top level was a set of what are called **generic domains** each of which identified a particular organization to which the owner of the host belonged. These are:

com: this identifies hosts that belong to a commercial organization,

edu: an educational establishment,

gov: the US federal government,

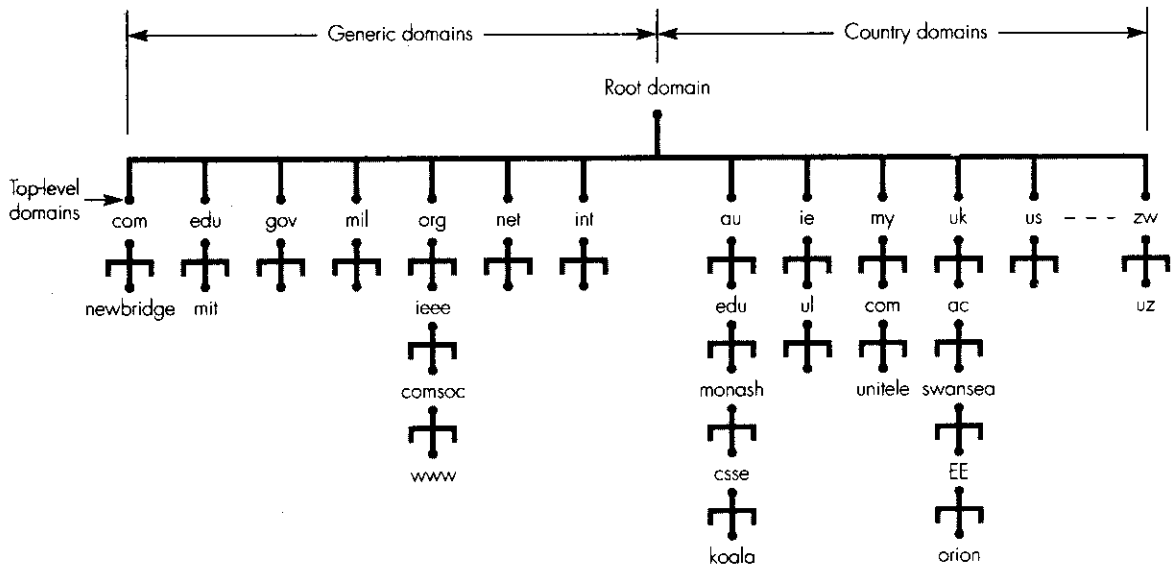


Figure 14.1 The structure of the domain name system together with some examples.

mil: the US armed forces,
org: a non-profit organization,
net: a network provider,
int: an international organization.

Later, as the Internet expanded its area of coverage, so a set of **country domains** were introduced. There is now a separate country domain for each country and these are defined in **ISO 3166**. For this reason, most hosts in the United States are identified by means of a generic domain and those outside of the US by their country domain. However, this is not always the case. Most large multinational companies, for example, often use the *com* generic domain. Also, since each domain is responsible for allocating the names of the (sub)domains that are linked to it, then some countries use different names from those used in the generic domain. For example, in the UK and a number of other countries, the domain name for *edu* is *ac* – academic community – and that for *com*, *co*. Note also that all names are case-insensitive and hence can be written in either upper-case or lower-case and still have the same meaning.

As we indicated earlier, the allocation of names is managed by the authority responsible for the domain where the name is to appear. For example, if a host located within the electrical engineering department of a new university is to be attached to the Internet, then, assuming the country is outside of the

US, first the university is assigned a name within the edu/ac domain of the country by the appropriate national authority, then the name of the department by an authority acting at a university level, and finally the name of the host by an authority within the department. The name of the host is then derived by listing the various domain names – each called a *label* – starting with the host name back to the root. These are listed from left-to-right with each label separated by a period (.) which is pronounced "dot". In this way, providing each label is unique within its own domain, then the resulting name is unique within the context of the total domain name space of the Internet. Note that the root has a null label and hence some examples are:

newbridge.com.
orion.EE.swansea. ac.uk.
unitele.com.my.

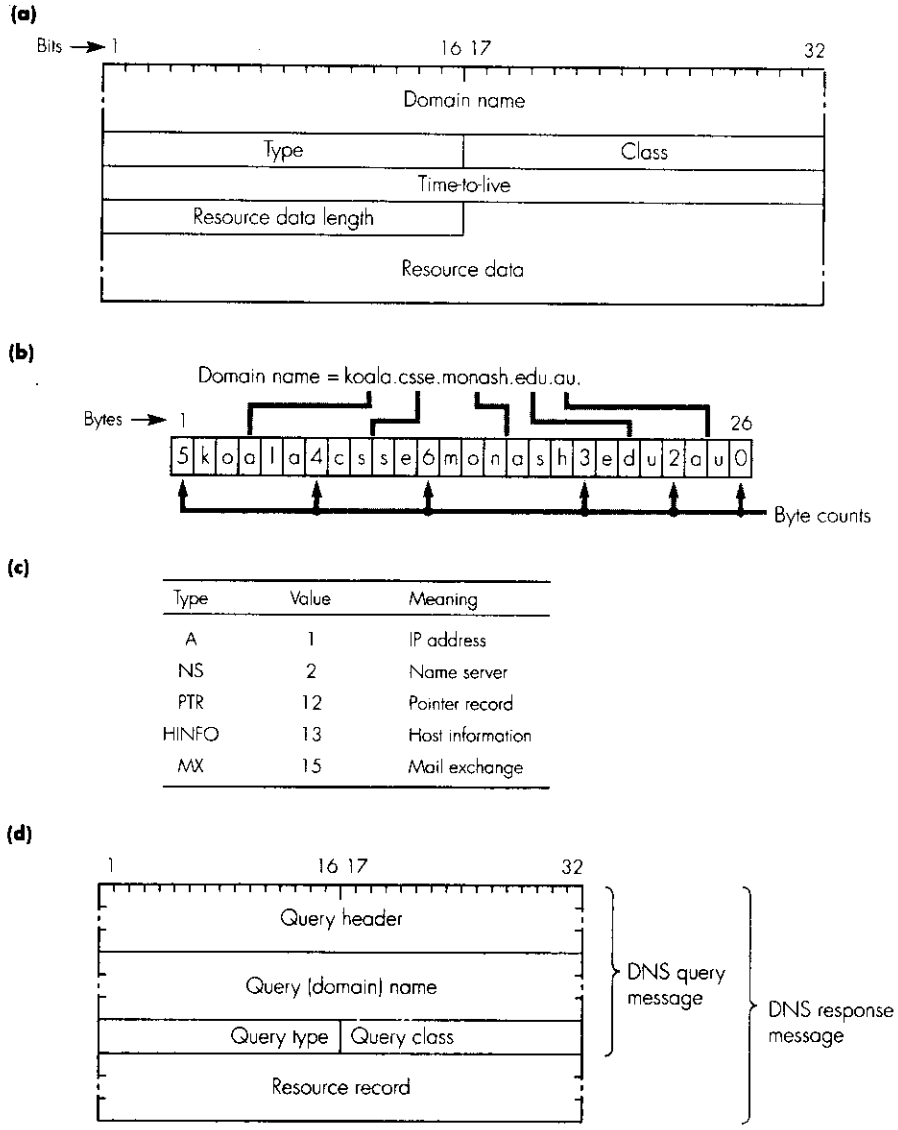
Note that since each name ends in a period, they are all examples of **absolute domain names** which are also called **fully qualified domain names** (**FQDN**). If the name does not end in a period, it is said to be incomplete or relative.

14.2.2 DNS resource records

Each domain name in the DNS name space may have information associated with it. This is stored in one or more **resource records**, each of which is indexed by the related domain name. A host name, for example, has a resource record that contains the IP address of the host. In practice there are a number of different types of record each of which has the standard format shown in Figure 14.2(a).

The *domain name* is the name of the domain to which the record relates. It consists of the string of labels that make up the domain name and is in the format shown in Figure 14.2(b). Each label is preceded by a 1-byte count that indicates the number of characters/bytes in the label. A label can be up to 63 characters long and the full domain name must be less than 256 characters. The final byte is always 0 which indicates the root.

The *type* field indicates the record type and a selection of these are listed in Figure 14.2(c). A type-A record, for example contains an IPv4 address which is stored in its 32-bit binary form. A type-NS record contains the name of the name server for this domain and is stored in the same format as the domain name. A type-PTR record contains an IP address stored in its dotted decimal form. A type-HINFO record contains the type of host and its operating system both of which are stored as an ASCII string. An MX-record contains the name of a host – an email gateway, for example, as we showed in Figure 5.12 – that is prepared to accept email for forwarding on a non-Internet (IP) site. There is also a type-AAAA record which contains an IPv6 address stored in its hexadecimal form. We shall discuss the use of some of these records in the next section.



Type	Value	Meaning
A	1	IP address
NS	2	Name server
PTR	12	Pointer record
HINFO	13	Host information
MX	15	Mail exchange

Figure 14.2 DNS resource records and queries: (a) resource record format; (b) domain name format; (c) a selection of resource record types; (d) query and response message formats.

For Internet records the *class* field is always 1 and has the mnemonic IN. The *time-to-live* field indicates the time in seconds the information contained within the record is valid. As we shall see, this is required when the IP address contained in the record has been cached. A typical value is 172 800 which is the number of seconds in 2 days.

The *resource data length* field specifies the length of the *resource data* field. As we have just indicated, the format of the latter differs for different record types and hence the number in the length field relates to the type of data present. For example, if it is an IPv4 address then the length field is 4 to indicate 4 bytes.

14.2.3 DNS query messages

The DNS database is queried using a similar list of (query) types to those used to describe the list of different resource record types. Hence there is a name-to-address resolution query – type A – and so on. A standard format is used to represent each query and this is shown in Figure 14.2(d).

The *query name field* holds the domain name – and hence resource record – to which the query relates. So for a name-to-address resolution query, for example, this contains the domain name of the host and this has the same format that we showed in Figure 14.2(b).

To initiate a query of the DNS – also called a *question* – a **DNS query message** is formed by adding a standard 12-byte header to the particular query. The **DNS response message** is then made up of the query message with one or more resource records – also called *answers* – appended to it. The 12-byte header contains a 16-bit *identification* field and a 16-bit *flags* field. The value in the identification field is assigned by the client that sent the query. It is then returned unchanged by the server in the response message and is used by the client to relate the response to a given query.

The flags field consists of a number of subfields. For example, a 1-bit field is used to indicate whether the message is a query (=0) or a response (=1). There is also a 4-bit field to indicate the type of search involved. As we shall see, this can be standard, recursive, iterative, or inverse.

14.2.4 Name servers

As we indicated earlier, the adoption of a hierarchical structure also facilitates the partitioning of the total DNS database so that most service requests – name-to-address mappings for example – can be carried out locally. To do this, the total domain name space is partitioned into a number of **zones** each of which embraces a unique portion of the total name space. Each zone is then administered by a separate authority which is also responsible for providing one or more **name servers** for the zone. Depending on its position in the hierarchy, a name server may have authority over a single zone or, if it is higher up in the hierarchy, multiple zones.

Associated with each zone is a *primary (name) server* and possibly one or more *secondary (name) servers*. The allocation of names and addresses within the zone is carried out through the primary server and it keeps this information – and hence its portion of the total database – in a block of resource records on hard disk. The resource records within its database are said there-

fore to be **authoritative records**. The records held in a secondary server are held in volatile storage and are cached versions of those held in a primary server. As we shall see, caching occurs when a primary or secondary server, on finding it does not have the resource record relating to a request, refers the request to a higher-level server. Then, on receipt of the requested IP address, the server that initiated the request retains a copy of this in its cache for a limited time period. The time is stored in the time-to-live field of the accessed resource record and, as we indicated earlier, typically, it is set to 2 days.

Some examples of (fictitious) zones are shown in Figure 14.3. As we can see, the top-level zones in the hierarchy have authority over multiple zones. The zone boundaries in the lower levels are intended to reflect the level of administrative overheads – and hence query requests – associated with the zone.

14.2.5 Service requests

All the information that is stored in each primary name server is accessible to both its secondary servers and also to any other primary server. Because of the excessive overheads that would be involved, however, each primary name server does not know how to contact – that is, does not have the IP address of – every other primary name server. Instead, each primary server knows only how to contact a set of top-level *root name servers*. There are only a small number of these and their IP addresses are stored in the configuration file of each primary server. In turn, each root server holds the name and IP address of each of the second-level servers in the hierarchy and, on receipt of a request from a primary server, the root server returns the name and IP address of the second-level server that should be used. The primary then proceeds to query this server and so on down the hierarchy until a resource record containing the required IP address is obtained. This procedure is

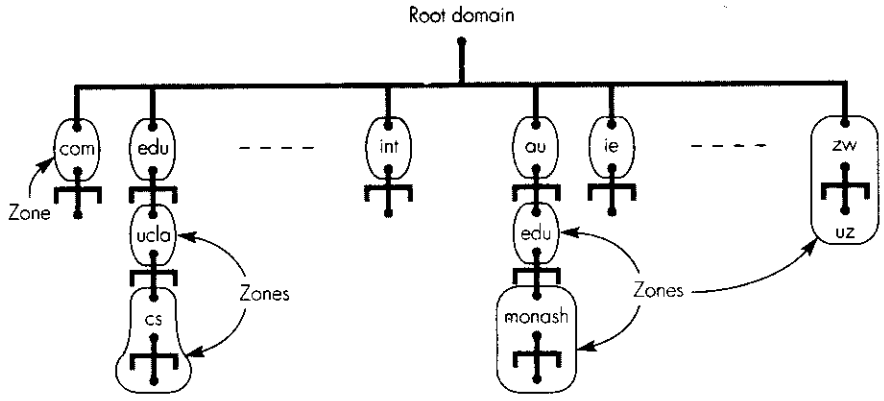


Figure 14.3 Some examples of DNS zones.

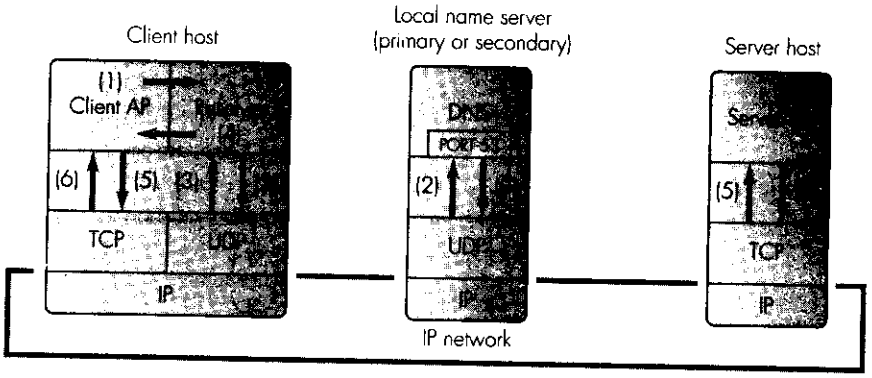
called a **recursive name resolution**. Alternatively, in order to reduce the amount of processing done by each name server, an iterative approach can be used. Before we describe each of these approaches, however, we shall describe first how a simple name resolution is carried out using a name server that is local to the host making the request.

Local name resolution

As we show in Figure 14.4, an AP running in a host that is attached to the Internet obtains the IP address of a named host through a piece of software called a **resolver**. Normally, this is a library procedure that is linked to the AP when the AP is first written. The resolver is given the IP address of the local name server – either a primary or a secondary server – that it should use to carry out name-to-address resolutions. Note that the (well-known) port number of a name server is 53.

As we shall see later, as part of all applications – file transfers, email transfers, and so on – the source (client) AP is given the name of the host in which the required destination (server) AP is running. Then, prior to initiating the networked application/transaction, the source AP invokes the resolver to obtain the IP address of the given destination host name (1). In the figure it is assumed that the resolver first sends a type-A query to its local name server requesting the IP address of the (destination) host specified in the query name field (2).

In this example it is assumed that the local name server has the resource record containing the IP address and hence this is returned in a type-A



(1) = resolver invoked by client AP with the name of the server host
(2) = resolver sends a type-A query containing the name of the server host to its local DNS
(3) = local DNS returns a type-A resource record containing the IP addresss of the server
(4) = resolver returns IP address of the server to the client AP
(5)/(6) = client and server APs carry out networked application/transaction

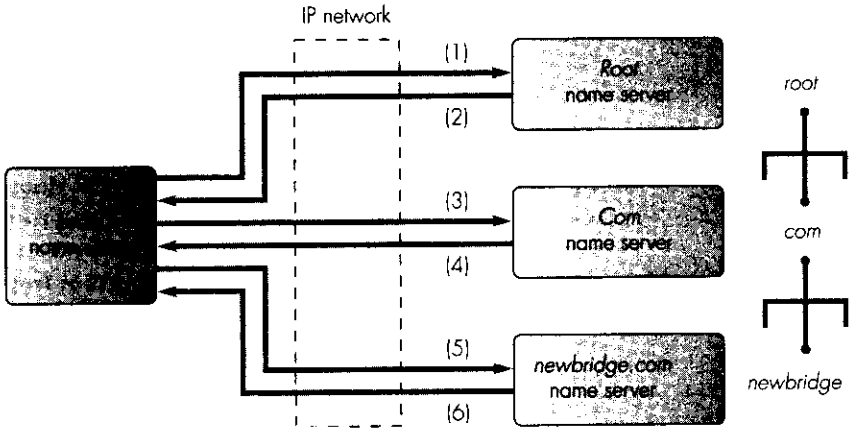
Figure 14.4 Example showing the sequence of messages exchanged for a local name resolution.

resource record (3). This is passed first to the resolver in the source – using UDP – and the resolver then returns the IP address contained within the record to the linked source AP (4). Once the source (client) AP has the IP address of the destination (server) AP, the two can start to carry out the networked application using TCP (5)/(6).

Recursive name resolution

When a local name server does not have a resource record relating to a given destination host name, it carries out a search for it. A schematic diagram showing the procedure followed using the recursive search method is given in Figure 14.5.

As we indicated earlier, all primary name servers have the IP addresses of the set of top-level root name servers. Hence as we can see, the local name server first sends a *recursive query* message containing the name of the required host – for example the *newbridge.com.* gateway – to one of the *root* name servers (1). From the name in the query message, the root server determines that the *com* name server should be queried and hence it returns the IP address of this in the reply – called an answer – message (2). On receipt of this, the local server sends a second query message to the *com* name server



(1) = local name server sends a recursive query message containing name of the destination host – for example, the *newbridge.com.* gateway – to the *root* name server
(2) = the *root* server returns the IP address of the *com* server
(3) = local server sends a recursive query to *com* name server
(4) = the *com* server returns the IP address of the *newbridge.com* server
(5) = local server sends a recursive query to *newbridge.com* server
(6) = the *newbridge.com* server sends IP address of *newbridge.com.* gateway (host)

Figure 14.5 Example showing the sequence of messages exchanged for a recursive name resolution.

using the returned IP address (3). In response, the *com* name server determines from the name in the query that the *newbridge.com* name server should be queried and hence it returns the IP address of this in the reply (4).

On receipt of this, the local server sends a third query message to the *newbridge.com* server (5) and, in the example, it is assumed that this has the requested resource record. Hence it returns this – containing the IP address of the destination host – to the local name server (6). The latter then relays the answer to the resolver in the source host and this, in turn, returns the IP address in the answer message to the source AP. The related client–server application can then start.

Note that in this example it was assumed that the local server was a primary server. If it was a secondary server, however, then this would send the first query to its (known) primary server and it is this that would initiate the sequence shown. Also, in order to reduce the number of queries that take place, each name server retains all resource records it receives – each containing an IP address – in a cache. In many instances, therefore, the answer to a query from a resolver is available in the cache so avoiding any external queries being sent out.

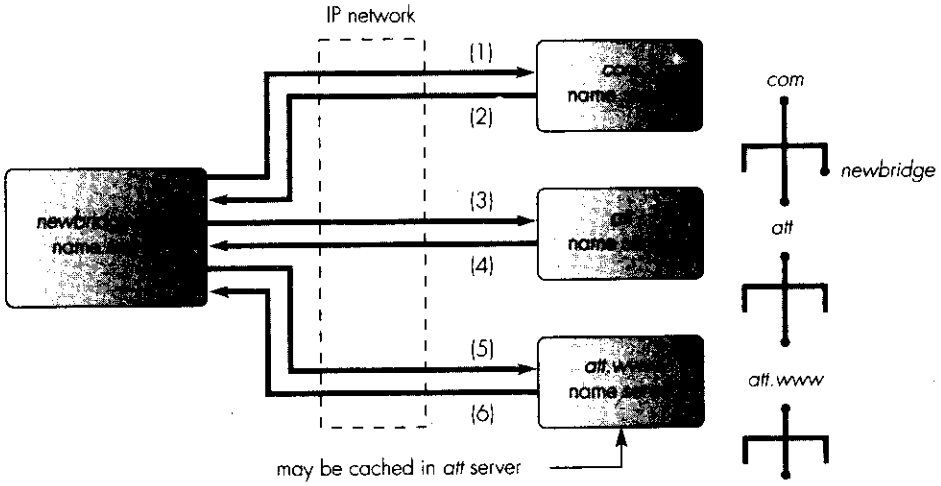
Iterative name resolution

In order to avoid always going to a root server – and hence to the top of the name tree – to initiate the search for an unknown resource record, with the iterative search method the local server starts its search for the requested record from the server that is nearest to it. Figure 14.6 illustrates the procedure.

To support the iterative method, instead of each primary name server having the IP addresses of the set of top-level root servers, it has the name and IP address of the next higher-level server to it in the naming hierarchy. Then, when a resolver sends a query – this time called an *iterative query* message – to its local server, if the local server does not have the requested resource record, instead of sending a query to a root server, it sends an (iterative) query to the next higher-level name server in the hierarchy – the *com* server in the example. If this has the record, it responds directly. If it hasn't, then it parses the name and returns a response containing the IP address of the server that it thinks might have (or is nearest to) the requested record. The local server then sends an (iterative) query to this server and so on until it receives a response message containing the requested record/IP address. As we can deduce from the figure, the search only proceeds up the tree to the level that is necessary to obtain the requested record thereby reducing the load on the top-level root servers.

Pointer queries

Although most queries of the DNS database relate to name-to-address translations, there are also queries that require an address-to-name translation. These are called *pointer queries* and, normally, they are from a system program that carries out a diagnostic operation, for example. They are also used by email servers and file servers to validate users.



(1) = local name server sends an iterative query message containing the name of the destination host – *att.www* – to the next higher-level server – *com*
(2) = the *com* server replies with the IP address of the *att* server
(3) = local server sends an iterative query to the *att* server
(4) = the *att* server returns the IP address of the *att.www* server
(5) = local server sends an iterative query to the *att.www* server
(6) = the *att.www* server returns the IP address of the requested Web host

Figure 14.6 Example showing the sequence of messages exchanged for an iterative name resolution.

To support this type of query, the resolver, given the IP address of a host, must initiate a search of the DNS and return the host name. As we indicated earlier, however, the search key of the DNS is a domain name. This means that with the database structure we showed in Figure 14.3, this type of query would require a complete search of the database starting with the set of top-level domain names. Clearly this is impractical and hence to support pointer queries an additional branch in the DNS name space to those shown in Figure 14.1 is present. This is shown in Figure 14.7.

As we can see, it starts with the top-level domain name *arpa*, followed by the second-level name *in-addr*. This is then followed by the four bytes (in dotted-decimal) that make up the IP address of each host whose name is in the DNS name space starting with the address-type byte. This order is used since the netid part of the address is assigned by *arpa.in-addr* and the hostid part by the authority that has been allocated the netid. Hence to be consistent with the other types of query message, the domain name in the pointer-type query message for the IP address 132.113.56.25 is:

25.56.113.132.in-addr.arpa

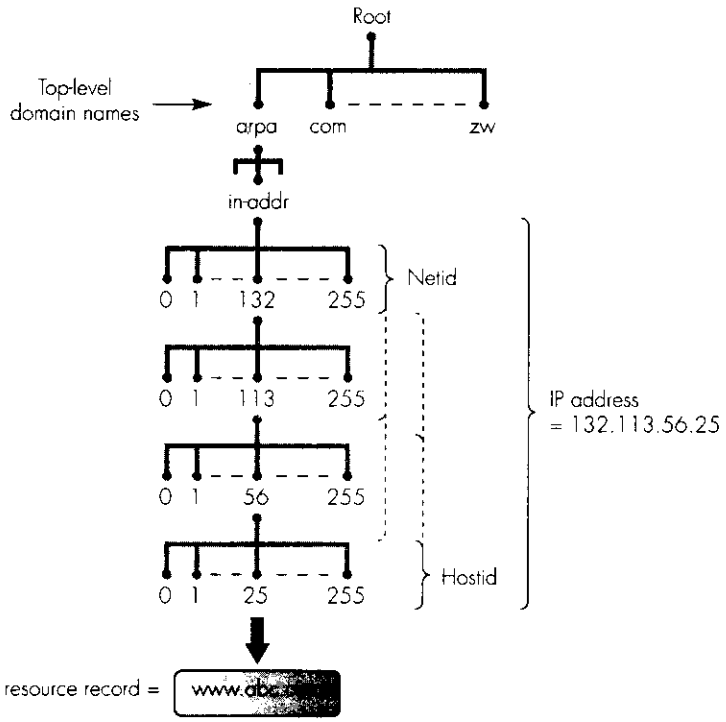


Figure 14.7 Pointer query principles.

that is, it starts with the last byte of the IP address first. The search of this portion of the database then yields a resource record as before but this time it contains the domain name of the host that has the given IP address. Hence in the example shown in the figure, this is assumed to be *www.abc.com*. Because of the reverse order of the labels in the domain name, a pointer query is also known as an **inverse query**.

14.3 Electronic mail

From a user perspective, electronic mail (email) – apart from Web surfing which we describe in the next chapter – is probably the most popular application on the Internet. We identified the standards relating to email over the Internet in Section 5.3.3 and, as we showed in Figure 5.10, an email system comprises two main components: an email client and an email server. Normally, an email client is a desktop PC or workstation which runs a program called the user agent (UA). This provides the user interface to the

email system and provides facilities to create, send, and receive (email) messages. To do this, the UA maintains an IN and an OUT mailbox and a list of selections to enable the user to create, send, read, and reply to a message, as well as selections to manipulate the individual messages in the two mailboxes, such as forward and delete.

The email server is a server computer that maintains an IN mailbox for all the users/clients that are registered with it. In addition, the server has software called the UA server to interact with the UA software in each client and also software to manage the transfer of mail messages over the Internet. The software associated with the latter function is called the **message transfer agent (MTA)** and is concerned with the sending and receiving of mail messages to/from other email servers that are also connected directly to the Internet.

The protocol stack that is used to support email over the Internet was shown earlier in Figure 5.11. Normally, the protocol stack associated with the access network – the internet service provider (ISP) network and the site/campus LAN shown in the figure – is either the PPP protocol we described in Section 9.9 – used with an ISP network – or, with a PC network, a protocol stack such as Novell NetWare. A number of different vendors then provide proprietary software to carry out the various interaction functions between the user and the UA client. In addition, there are a number of protocols that can be used to control the transfer of messages over the access network. For example, the **POP3 protocol** – post office protocol 3 – is often used to fetch messages from the user's IN mailbox in the server to the IN mailbox maintained by the UA. Essentially, POP3 defines the format of the various control messages that are exchanged between the UA client and UA server to carry out a transfer and also the sequence of the messages that are exchanged. POP3 is specified in **RFC 1939**.

The application protocol that is used to control the transfer of messages between two MTAs over the Internet is called the **simple mail transfer protocol (SMTP)**. It is specified in **RFC 821**. In this section we first describe the structure of mail messages and the use of the various fields in each message header. We then present an overview of how a typical message transfer is carried out and finally, the operation of SMTP.

14.3.1 Structure of email messages

When we send a letter using a postal service, we first write our own name and address at the head of the letter followed by the message we wish to send. Typically, this comprises the name and address of the intended recipient followed by the actual letter/message content. We then insert the letter into an envelope and write the name and address of the intended recipient on the front of the envelope. Also, to allow for the possibility of the recipient having changed address, we often write our own name and address on, say, the back of the envelope. We then deposit the envelope into a mailbox provided by

the postal service. The latter then uses the name and address on the front of the envelope to forward and deliver it to the address of the intended recipient. Alternatively, if this is not possible, it uses the address on the back of the envelope to return it to the sender. The recipient knows who sent the letter by the name and address at the head of the letter.

Thus there are two distinct procedures involved in sending a letter: the first involving the sender of the letter and concerned with the preparation of the letter itself and the second with the transfer of the addressed envelope to the intended recipient by the postal service. We note also that the structure and content of the letter itself has only meaning to the sender and recipient of the letter.

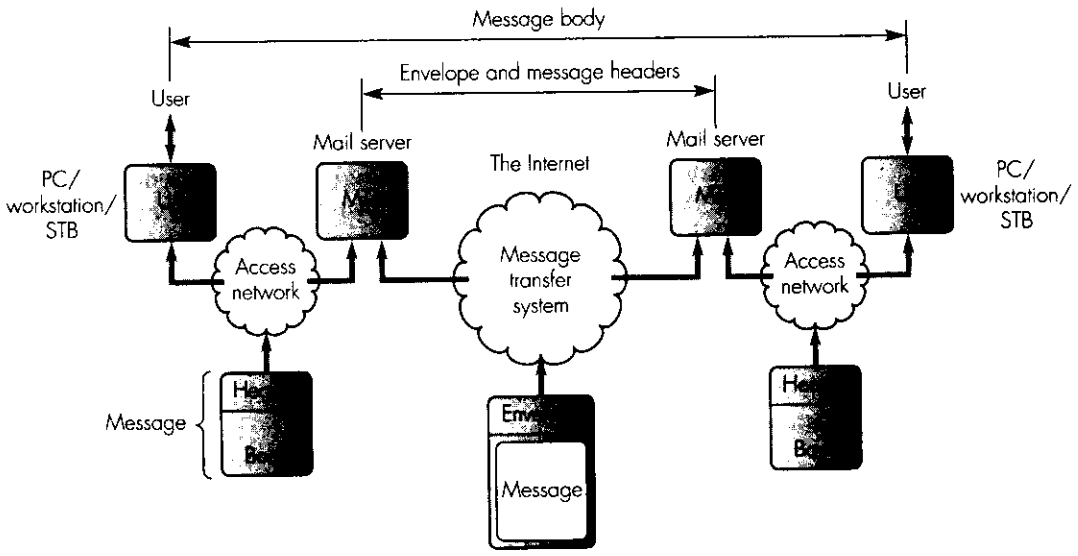
In a similar way, the sending of electronic mail involves two separate procedures. The first is concerned with the entry of various fields – including the sender's and recipient's name/address at the head of the message – and the actual message content via the UA; the second with the encapsulation of the message into an (electronic) envelope containing the sender's and recipient's address and with the transfer of the envelope over the network by the message transfer system. In the case of electronic mail, however, since the writing of the addresses on the envelope is performed by the mail system itself, it is necessary for the addresses at the head of the message to have a standard structure so that they can be extracted and used directly by the message transfer system. The terminology associated with the structure of an email message showing these two parts is shown in Figure 14.8(a).

As we can see, during its transfer across the network an email message is composed of an *envelope* and the *message*. The envelope contains the email address of the sender of the message (*MAIL FROM*) and its intended recipient (*RCPT TO*). In the case of the Internet, all email addresses are of the form *user-name@mailserver-name* where *mailserver-name* is the DNS name of the mail server and *user-name* is selected by the user and confirmed by the local mail manager at subscription/registration time. The manager also creates an IN mailbox for the user on the mail server at the same time.

The format of an email address is defined in RFC 821 and, as we shall see, the recipient mailserver-name is used by the message transfer system to route the message over the Internet to the intended recipient mail server and the user-name is then used by the MTA to determine the IN mailbox into which the mail should be deposited. Like a DNS name, the user-name is case insensitive and the two names are separated by the @ symbol.

The message itself is composed of a *header* and a *body*, the latter containing the actual message that has been entered by the user via the UA. As we show in Figure 14.8(b), the header comprises a number of fields some of which are optional. Also, since there are many different vendors of UA software, the optional fields that are used by the UA in the sender and those used by the recipient UA may differ. However, all of the header fields have a standard format which is defined in RFC 822. Each field comprises a single line of ASCII text starting with the (standardized) field name. This is terminated by

(a)



(b)

Used also by the message transfer system	From:	Email address of person who created the message
	To:	Email address of primary recipient
	Cc:	List of email addresses of other (secondary) recipients
	Received:	Route followed through message transfer system
	Return-Path:	Name of last MTA
Used by UA/user	Sender:	Email address of the sender of the message
	Date:	Date and time message was sent by UA
	Message-Id:	Unique identifier assigned to the message by the UA
	Reply-To:	Email address to which a reply should be sent
	Subject:	Single-line title of the message
User-defined	X-PhoneNumber:	Sender's phone number
	X-FaxNumber:	Sender's fax number

Figure 14.8 Email message structure: (a) terminology and usage; (b) selection of the fields in the message header and their use.

a colon and is followed by the field value. A single blank line is then used to separate the header from the message body.

As we can see, some fields are also used by the message transfer system and others by the UA/user. It is also possible for a user to add one or more private header fields. Some examples of fields in each category are shown in the figure together with their usage. Most are self explanatory. Note, however, that *From:* and *Sender:* may be different. For example, the person who created – and is sending – the message (*From:*) may be the secretary of the person

who is identified in the *Sender:* field. Also, if the reply to the message should be sent to a third party, then the email address of the person who is to receive the reply is in the *Reply-To:* field.

Note also that even though the *Received:* and *Return-Path:* fields are in the header of the message – rather than the envelope – they are used primarily by a network manager during fault diagnosis to determine the path that is followed by a message through the message transfer system. To initiate this procedure, the source MTA enters its own name together with the message identifier and the date and time the message was sent in a *Received:* field. A new *Received:* field containing the same information is then added to this by each MTA along the path that the message takes. In the case of the *Return-Path:* field, this contains only the name of the last MTA. In practice, however, this field is often not used and, if present, contains only the email address of the sender. Note that user-defined fields must always start with the character sequence *X-*.

In RFC 822, the transfer syntax used for all the header fields is the US version of the ASCII code we showed earlier in Figure 2.6(a) with the addition that each 7-bit character is first converted into an 8-bit byte by adding a 0 bit in the most-significant bit position. Also, the codeword used to represent an *end-of-line* is the 2-byte combination of a carriage return (CR) and a line feed (LF) and, because of this, the codeword for a CR is the 2-byte combination of a CR and a NUL. This modified version of the ASCII codeword set is called **network virtual terminal (NVT) ASCII**.

14.3.2 Message content

With the RFC 822 standard the content part of a message – the body – can only be lines of ASCII text with the maximum length of each line set at 1000 characters. The sending UA then converts each character into NVT ASCII as the transfer syntax with each character converted into its 8-bit form. This ensures that there can be no combinations of codewords in the content field that can be misinterpreted by an MTA as a protocol message.

The RCF 822 standard was first introduced in 1983 and is still used widely for sending text messages. As the use and coverage of the Internet widened, however, so the demand for alternative message types increased, for example, to allow messages to contain binary data and other media types such as audio and video. Also messages containing different languages and alphabets. As a result, an extension to the RFC 822 standard was introduced. This is known as **Multipurpose Internet Mail Extensions (MIME)**. It was first specified in **RFC 1341** and later updated in **RFCs 2045/8**.

The aim of MIME was to enable users to send alternative media types in messages but still use the same message transfer system. The solution was to add a number of extra header fields to the existing fields which collectively enable the user to define alternative media types in the message body. It also provided a way of converting the alternative media types that are supported into strings of ASCII characters which can then be transferred using NVT ASCII.

MIME headers

The additional MIME header fields and their meaning are listed in part (a) of Table 14.1. The first field following the standard header fields is the *MIME-Version:* which, when present, informs the recipient UA that an alternative message content to ASCII text is present in the message body. It also includes the version number of MIME that is being used. If the field is not present, the

Table 14.1 MIME: (a) additional header fields; (b) alternative content types.

(a) Header	Meaning
MIME-Version:	Defines the version of MIME that is being used
Content-Description:	Short textual description of the message content
Content-Id:	Unique identifier assigned by the UA
Content-Type:	Defines the type of information in the body
Content-Transfer-Encoding:	The transfer syntax being used
Content-Length	The number of bytes in the message body

(b) Type	Subtype	Content description
Text	Plain	Unformatted ASCII
	Richtext	Formatted text based on HTML
Image	GIF	Digitized image in GIF
	JPEG	Digitized image in JPEG
Audio	Basic	Digitized audio
Video	MPEG	Digitized video clip or movie
Application	Octet-stream	A string of bytes
	Postscript	A printable document in PostScript
Message	Rfc822	Another MIME message
	Partial	Part of a longer message
	External-body	Pointer to where message body can be accessed
Multipart	Mixed	Each part contains a different content and/or type
	Alternative	Each part contains the same content but with subtype of a different type
	Parallel	The parts should be output simultaneously
	Digest	Multiple messages

recipient UA assumes the content is NVT ASCII. When it is present, the following four fields then expand upon the type of content that is there and the transfer syntax that is being used.

The *Content-Description:* field is present to allow a user to enter a short textual string in ASCII to describe to the recipient user what the contents are all about. It is similar, therefore, to the *subject:* field in the standard header. One or other is often used to decide whether the message/mail is worth reading. Similarly, the *Content-Id:* field performs a similar function to the *Message-Id:* field in the standard header.

The *Content-Type:* field defines the type of information in the message body. The different types are defined in RFC 1521 and a selection of them are listed in part (b) of the table. As we can see, each type comprises two parts: a specification of the type of information – text, image, and so on – followed by a subtype. The type of information in the message body is then defined by a combination of the type and its subtype, each separated by a slash. Some examples are:

Content-Type:Text/Richtext
Content-Type:Image/JPEG

Some contain one or more additional parameters. The format used is as follows:

Content-Type:Text/Plaintext; charset=US-ASCII
Content-Type:Multipart/Alternative;boundary="NextType"

Clearly, for each type/subtype combination a standard format must be used so that the recipient UA can interpret (and output) the information in a compatible way with how it has been encoded by the sending UA. Hence associated with each combination is a defined (abstract) syntax. For example, the Text/Plaintext combination implies the contents are ASCII text while for the Text/Richtext combination a markup language similar to HTML is used, an example of which we showed earlier in Figure 2.8. Similarly, digitized images, audio, and video are all represented in their compressed form. As we saw in Chapters 3 and 4, there is a range of standard compression algorithms available with each of these media types. In the case of images, for example, the two alternative subtypes supported are GIF – Section 3.4.1 – and JPEG – Section 3.4.5. The subtype selected for audio is a form of PCM – Section 4.2.1 – and that for video a version of MPEG – Section 4.3.4. In the case of a movie (video with sound), the MPEG subtype is used with the audio and video integrated together using the format we described in Section 5.5.1.

The *Application:* type is used when the body contents require processing by the recipient UA before they have meaning on the user's display. For example, an *Octet-stream* subtype is simply a string of bytes representing, say, a compiled program. Typically, therefore, on receipt of this type of information the UA would prompt the user for a file name into which the data should be written. Similarly, for the *Postscript* subtype unless the UA contains a PostScript interpreter to display the contents on the screen.

The *Message:* type is used when the contents relate to another MIME message. For example, if the contents contain another RFC 822 message, then the **Rfc822** subtype is used, possibly to forward a message. Similarly, the *Partial* subtype is used when the contents contain a fragment of a (larger) message. Typically, additional parameters are then added by the sending UA to enable the recipient UA to reassemble the complete message. This feature is used, for example, to send a long document or audio/video sequence. The *External-body* subtype is used when the message content is not present in the message and instead an address pointer to where the actual message can be accessed from; for example, the DNS name of a file server and the file name. This feature is often used to send an unsolicited message such as a long document. Typically, the UA is programmed to ask the user whether he or she wishes to access the document or not. Some examples of parameters with the different message subtypes are:

Content-Type:Message/Partial;id="file-name@host-name";number=1;total=20
Content-Type:Message/External-Body;access-type="mail-server";server="server-name"

The *Multipart:* type is used to indicate that the message body consists of multiple parts/attachments. Each part is clearly separated using a defined delimiter in a parameter. With the *Mixed* subtype each part can contain a different content and/or type. With the *Alternative* subtype each part contains the same content but with a different subtype associated with it; for example, the same message in text or audio or in a number of different languages/alphabets. Normally, the alternative parts are listed in the preferred order the user/sending UA would like them to be output. The *Parallel* subtype indicates to the recipient UA that the different parts should be output together; for example a piece of audio with a digitized image. Finally the *Digest* subtype is used to indicate that the message body contains multiple other messages; for example to send out a set of draft documents that the sender may have received from a number of different members of a working group.

An example of a simple multimedia mail showing a number of the features we have just considered is shown in Figure 14.9. The message is simply ***Happy birthday Irene*** and this is sent in three different formats. The first is in the form of an audio message which would necessitate the recipient's PC/workstation having a sound card with associated software. Also, the recipient UA must have software to interpret the contents of the accessed audio file and output this to the sound card. If this is not available, normally the UA is programmed to move to the next option and, if the UA cannot interpret richtext, then the message will be output in plaintext. As we can see from this, what we have described in this section relates only to the (abstract) syntax of the messages that are used by a UA to send a multimedia mail to another UA. What the user sees/hears depends on how the UA has been programmed.

Transfer encoding

The next-to-last field in the MIME header we showed earlier in Table 14.1 is the *Content-Transfer-Encoding:* field and an example of its use was shown in

```
From: xyz@abc.com
To: abc@xyz.com
Subject: Happy birthday Irene
MIME-Version: 1.0
Content-Type: Multiport/Alternative; boundary = "TryAgain";

- - TryAgain

Content-Type: Message/External-body;
        name = "Irene.audio";
        directory = "Irene";
        access-type = "anon-ftp";
        site = "myserver.abc.com";
Content-Type: Audio/Basic;                    (Message in audio accessed remotely)
Content-Transfer-Encoding: Base64

- - TryAgain

Content-Type: Text/Richtext;
<B> ***Happy birthday Irene*** </B> (Message in richtext)

- - TryAgain

Content-Type: Text/Plain;
        ***Happy birthday Irene*** (Message in plaintext)

- - TryAgain
```

Figure 14.9 MIME: example type and subtype declarations.

Figure 14.9. As the name implies, it is concerned with the format of the message content during its transfer over the Internet. Since all the extension fields in the extension header are in 7-bit ASCII, they are encoded in 8-bit NVT ASCII. Recall also that with an RFC 822 message, the UA uses the same transfer syntax to send lines of 7-bit ASCII over the Internet in order to ensure there can be no combinations of codewords in the content field that can be misinterpreted by an MTA as an SMTP protocol message. With the additional media types associated with MIME, however, the message content may be in an 8-bit form with a binary 1 in the eighth bit – a string of 8-bit speech samples for example. As we indicated earlier, the aim of MIME is to use the same message transfer system which means that the message body should be encoded in NVT ASCII. Hence once the message contents have been input, the UA first converts all non-ASCII data first into lines of (7-bit) ASCII characters and then into lines of NVT ASCII. Collectively this is referred to as transfer encoding.

Two alternative transfer encodings are defined in RFC 1521 for use with an RFC 821-conformant message transfer system (MTA):

- quoted-printable: this is used to send messages that are composed of characters from an alternative character set that is mostly ASCII but has a small number of special characters which have their eighth bit set to 1. Examples are all the Latin character sets;

- base64: this is used to send blocks of binary data and also messages composed of strings of characters from a character set that uses 8-bit codewords such as EBCDIC.

When the MIME header field contains *Content-Transfer-Encoding: Quoted-printable* the UA converts the codewords of those characters which have their eighth bit set into a string of three characters. The first is the = character and this is followed by the two characters that represent the (8-bit) character in hexadecimal. For example, if the codeword for a special character was 1 110 1001, then the hexadecimal representation of the character is E9 (hex). Hence this would be converted into the three-character sequence =E9.

When the MIME header field contains *Content-Transfer-Encoding:Base64* the message content, instead of being treated as a string of 8-bit bytes, is treated as a string of 24-bit values. Each value is then divided into four 6-bit subvalues, each of which is then represented by an ASCII character. The 64 ASCII characters used to represent each of the possible 6-bit values are listed in Table 14.2. In the event that the contents of a message do not contain a multiple of three bytes, then one or two = characters are used as padding.

Table 14.2 Base64 encoding table.

6-bit value (Hex)	ASCII char	6-bit value (Hex)	ASCII char	6-bit value (Hex)	ASCII char	6-bit value (Hex)	ASCII char
00	A	10	Q	20	g	30	w
01	B	11	R	21	h	31	x
02	C	12	S	22	i	32	y
03	D	13	T	23	j	33	z
04	E	14	U	24	k	34	0
05	F	15	V	25	l	35	1
06	G	16	W	26	m	36	2
07	H	17	X	27	n	37	3
08	I	18	Y	28	o	38	4
09	J	19	Z	29	p	39	5
0A	K	1A	a	2A	q	3A	6
0B	L	1B	b	2B	r	3B	7
0C	M	1C	c	2C	s	3C	8
0D	N	1D	d	2D	t	3D	9
0E	O	1E	e	2E	u	3E	+
0F	P	1F	f	2F	v	3F	/

Example 14.1

A binary file containing audio data contains the following string of four 8-bit samples:

10010101 11011100 00111011 01011000

Assuming Base64 encoding,

(i) using the codewords for ASCII characters given earlier in Figure 2.6(a), derive the contents of the file in NVT ASCII;

(ii) show how the original contents are derived from the received NVT ASCII string.

Answer:

(i) Input first converted into two 24-bit values using two = characters as padding:

Value 1 = 10010101 11011100 00111011

Value 2 = 01011000 00[illegible]

Each 24-bit value is converted into four 6-bit values:

Value 1 = 10 0101 01 1101 11 0000 11 1011

Value 2 = 01 0110 [illegible]

Hence from codewords in Figure 2.6(a) contents in NVT ASCII are:

Value 1 = 0 110 1100 [illegible] 0 011 0011 0 011 2011

Value 2 = 0 101 0111 [illegible] 0 010 0100 0 011 0000 0 011 1001

(ii) Received bitstream first converted back into the equivalent ASCII string using codewords in Figure 2.6(a):

Value 1 = *ldu7*

Value 2 = *WD09*

The equivalent four 6-bit subvalues then obtained from Table 14.2:

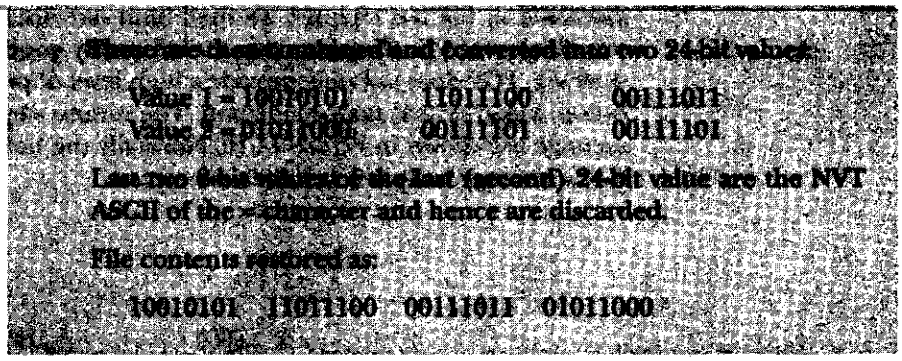
Value 1 = 2 5 1 D 3 B 3 B

= 10 0101 01 1101 11 0000 11 1011

Value 2 = 1 D 0 3 3 9 3 D

= 01 0110 00 0011 11 0000 11 1101

14.1 Continued



Encryption

As with all Internet applications, it is relatively straightforward to read messages during their transmission over the access networks and the Internet itself. It is now becoming common practice with many companies, for example, to monitor the email that is being sent and received by their employees. Increasingly, therefore, people and organizations are applying encryption methods to the mail messages that they send.

When two people are communicating using ASCII text and wish to foil a casual eavesdropper from reading their mail, a simple approach to obtaining privacy is to encode the message body using Base64 before it is entered. This can be done very easily by interpreting the stream of ASCII characters that make up the message as a bitstream. Then, by applying Base64 to the bitstream, the resulting transmitted (NVT) ASCII character string is meaningless to the casual eavesdropper.

For example, using the table of ASCII codewords in Figure 2.6(a), the character string *I LOVE* in 7-bit ASCII is:

1001001(I) 0100000(SP) 1001100(L) 1001111(O) 1010110(V) 1000101(E)

Hence, when this bitstream is interpreted as a string of 6-bit groups, it yields:

10 0100 10 1000 00 1001 10 0100 11 1110 10 1101 00 0101

In hexadecimal these are equivalent to:

24 28 09 24 3E 2D 05

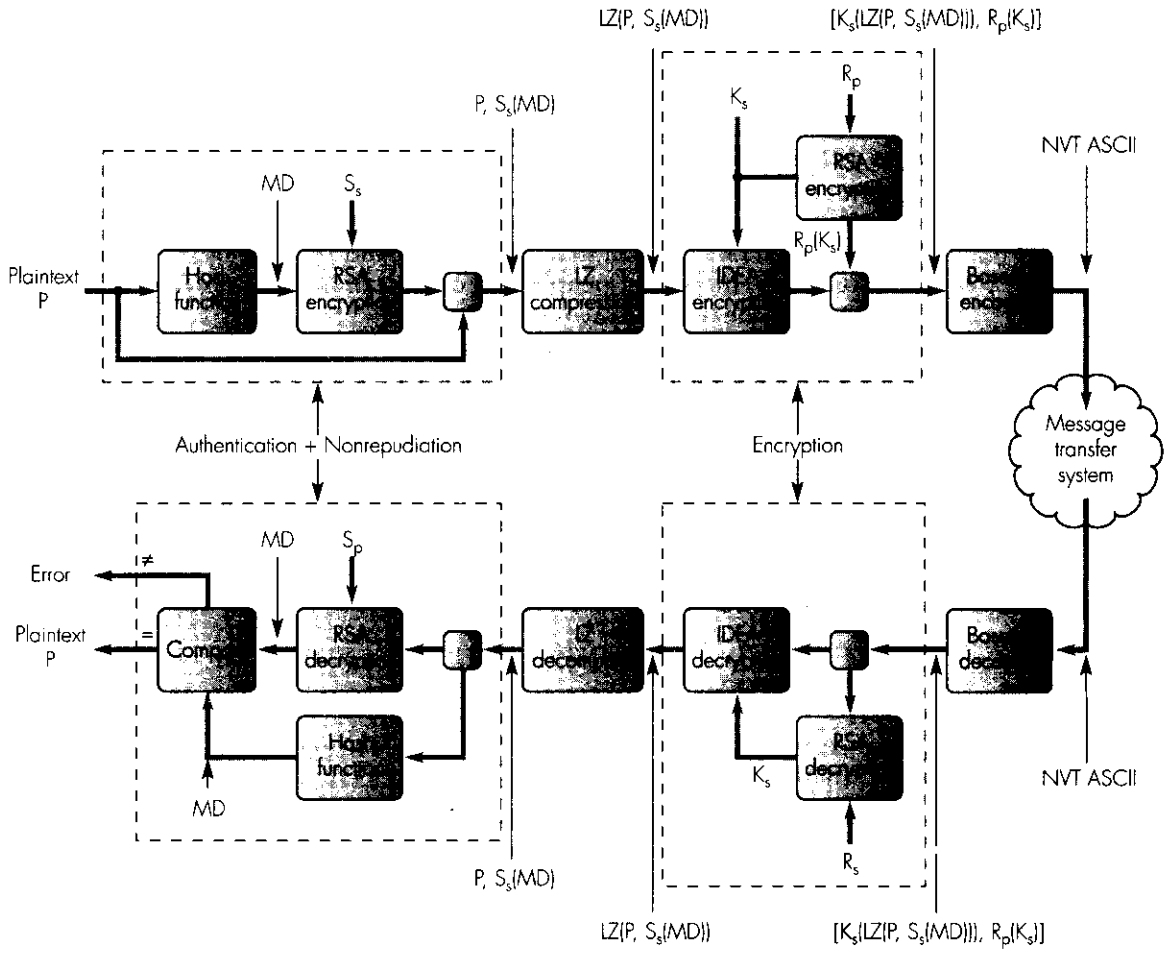
or, using the Base64 encoding in Table 14.2, the ASCII character string:

k o J k + t F

which, of course, is less interesting should it be observed.

Clearly it is a trivial task to break this code and for applications that demand a high level of security, a number of more sophisticated encryption schemes are now used. Since it uses a number of the encryption methods we

discussed in the last chapter, we shall limit our discussion to a widely used scheme devised by Zimmermann called **pretty good privacy (PGP)**. As we shall see, PGP does not only provide a high level of privacy, but also authentication, integrity, and nonrepudiation. The various steps followed to encrypt a message are shown in Figure 14.10. Normally the header fields are repeated in the message body which is then encrypted.



S_s = Sender's RSA secret key
S_p = Sender's RSA public key
K_s = Sender's IDEA secret key
R_s = Recipient's RSA secret key
R_p = Recipient's RSA public key

Figure 14.10 Email privacy: PGP encryption and decryption.

As we can see, the overall process involves a combination of MD5 (Section 13.5), RSA (Section 13.4.5), IDEA (Section 13.4.4) and Base64. It also uses the Lempel–Ziv (LZ) compression algorithm we discussed in Section 3.3.4.

The first block is concerned with authentication and nonrepudiation. It uses the same scheme that we showed in Figure 13.6. This produces the plaintext, P, and a 128-bit message digest (MD). The MD is then encrypted using the sender's RSA secret key, Ss. At the recipient side, the MD is again computed using the same hash function and, if this is the same as the decrypted MD that was sent with the message, this is taken as proof that the message was indeed sent by the sender in the *From:* field of the message header. This is then taken as authenticating the sender and, should it be necessary, nonrepudiation.

The output of the authentication and nonrepudiation block – P, S_s (MD) – is compressed using the LZ algorithm and the output of this is passed to the encryption block. This is based on IDEA and uses a 128-bit secret key, K_s, to encrypt the LZ compressed block. K_s is also encrypted using the recipient's RSA public key, R_p, to produce $R_p(K_s)$. This, together with the output of the IDEA compression block, forms the binary output. Finally, this is Base64 encoded to yield an ASCII string which is then transmitted in the form of an NVT ASCII string.

The processing at the receiver is the inverse of that carried out at the sending side. First the sender's IDEA secret key, K_s, is obtained by using the recipient's RSA secret key, R_s, to decrypt $R_p(K_s)$. K_s is then used to decrypt the LZ compressed block. After being decompressed, the resulting output is passed to the authentication and nonrepudiation block.

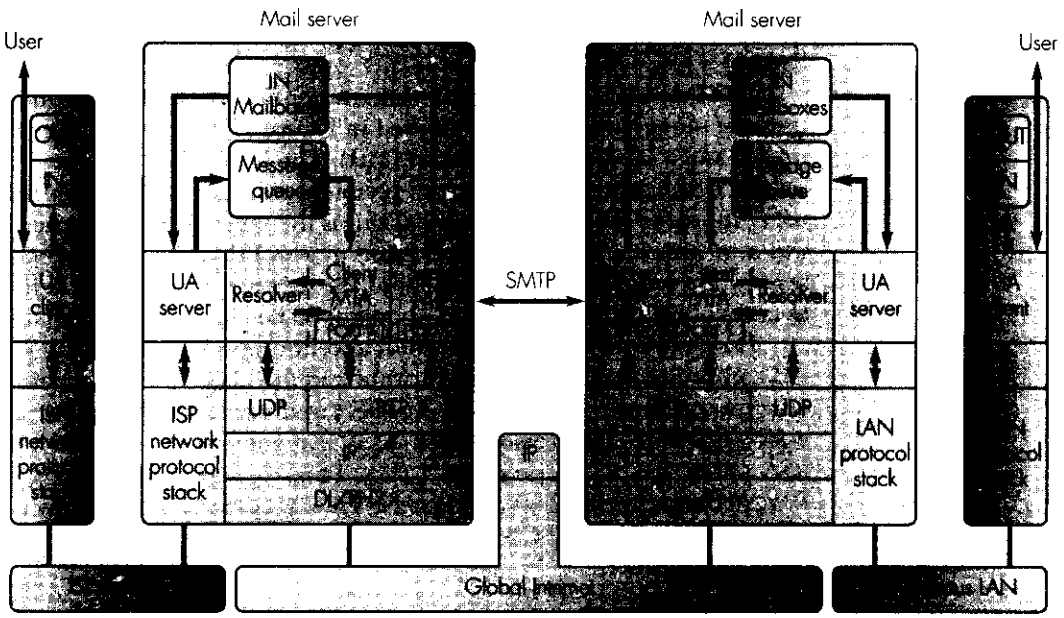
It should be noted that, because of the high processing overheads associated with RSA, the two RSA stages operate only on small block sizes, the first the 128-bit MD and the second the 128-bit secret key used by IDEA. Also the message contents are encrypted using IDEA which, as we saw in Section 13.4.4, is very fast.

14.3.3 Message transfer

The main components that make up the message transfer system are shown in Figure 14.11(a). Once a user has created a mail message and clicked on the SEND button, the UA first formats the message into NVT ASCII and then sends it to the UA server in its local mail server using the protocol stack of the access network. On receipt of the message, the UA server deposits the message into the message queue ready for sending by the MTA over the Internet.

The client MTA checks the contents of the message queue at regular intervals and, when it detects a message has been placed in the queue, it proceeds to format and send the message. The MTA first reads the email addresses from the *From:* and *To:* fields of the message header and writes them into the *MAIL FROM:* and *RCPT TO:* fields in the envelope header. It then examines the *Cc:* field in the message header and, if other recipients are listed, it proceeds to create further copies of the message each with a different *RCPT TO:* value in the envelope header.

(a)



Port X/Y = ephemeral ports Port 25 = well-known port number of server MTA

(b)

Commands (Client MTA → Server MTA)	Descriptions
HELO Mailserver-name	Sends DNS name of the client mail server
MAIL FROM: <email address of sender>	email address of sender
RCPT TO: <email address of recipient>	email address of recipient
DATA	Request to send the message
QUIT	Requests recipient server to close TCP connection
RSET	Abort current mail transfer

(c)

Responses (Server MTA → Client MTA)	Descriptions
220	Recipient server is ready
221	Recipient server is closing TCP connection
250	Command carried out successfully
354	Indicates the recipient server is ready to receive message
421	Service request declined
450	Mailbox unavailable
⋮	⋮
551	Addressed user is not here

Error responses

Figure 14.11 SMTP: (a) components; (b) command messages; (c) response messages.

As we saw earlier, all email addresses are in the form *user-name@mailserver-name* where *mailserver-name* is the DNS name of the mail server. Hence before the client MTA can send any of the formatted messages over the Internet, it must first obtain the IP address of each of the recipient mail servers. As we saw in Section 14.2.5, this is done using the resolver procedure that is linked to all Internet APs. Once the set of IP addresses have been obtained, the client MTA is ready to initiate the transfer of each message. The protocol that is used to control the transfer of a message from one MTA to another is the simple mail transfer protocol (SMTP). The various control messages that are used by SMTP and the sequence in which they are exchanged are defined in RFC 821. All the control messages are encoded in NVT ASCII.

Each message is transferred over a previously established TCP connection. Hence to send each message, the client MTA first initiates the establishment of a logical connection between itself and the MTA server in the recipient mail server using the latter's IP address and port 25 which is the well-known port number for SMTP. The server MTA accepts the incoming (TCP) connection request and, once this is in place, proceeds to exchange SMTP control messages (PDUs) with the client MTA to transfer the message. The control messages that are sent by the MTA client are called *commands* and a selection of these are shown in Figure 14.11(b). As we can see, most are composed of four uppercase characters. The MTA server responds to each command with a three-digit numeric reply code with (optionally) a readable string. A selection of the reply codes are given in Figure 14.11(c).

A typical exchange sequence of SMTP control messages to send a mail message is shown in Figure 14.12. To avoid confusion, the TCP segments that are used to transfer the messages are not shown. As we can see, once the server has received the acknowledgment indicating a TCP connection is now in place, the server MTA returns a 220 response indicating it is ready to start the message transfer sequence. This starts with the MTA client sending a HELO command and the MTA server returning a 250 response indicating it is prepared to accept mail from the sending server. The client MTA then sends the sender's email address and, if this is accepted, it sends the intended recipient's email address. In the example, this is accepted by a 250 response. If this was not acceptable, typically the MTA server would return either a 450 (mailbox unavailable) or a 551 (addressed user is not here).

Assuming both addresses are valid, the client proceeds to send a DATA command to determine if the server MTA is now ready to receive the message itself. If it is, the server returns a 350 response and, on receipt of this, the transfer of the message takes place. The message consists of multiple lines, each of up to 1000 characters, with a single "." character on the last line. All characters are encoded in NVT ASCII. Note also that the number of TCP segments used to transfer the message is determined entirely by the two TCP entities. On receipt of the (reassembled) message, the server MTA transfers the message to the IN mailbox of the recipient user and returns a 250 response to the client MTA. The latter then sends a QUIT command to request the MTA server closes the TCP connection.

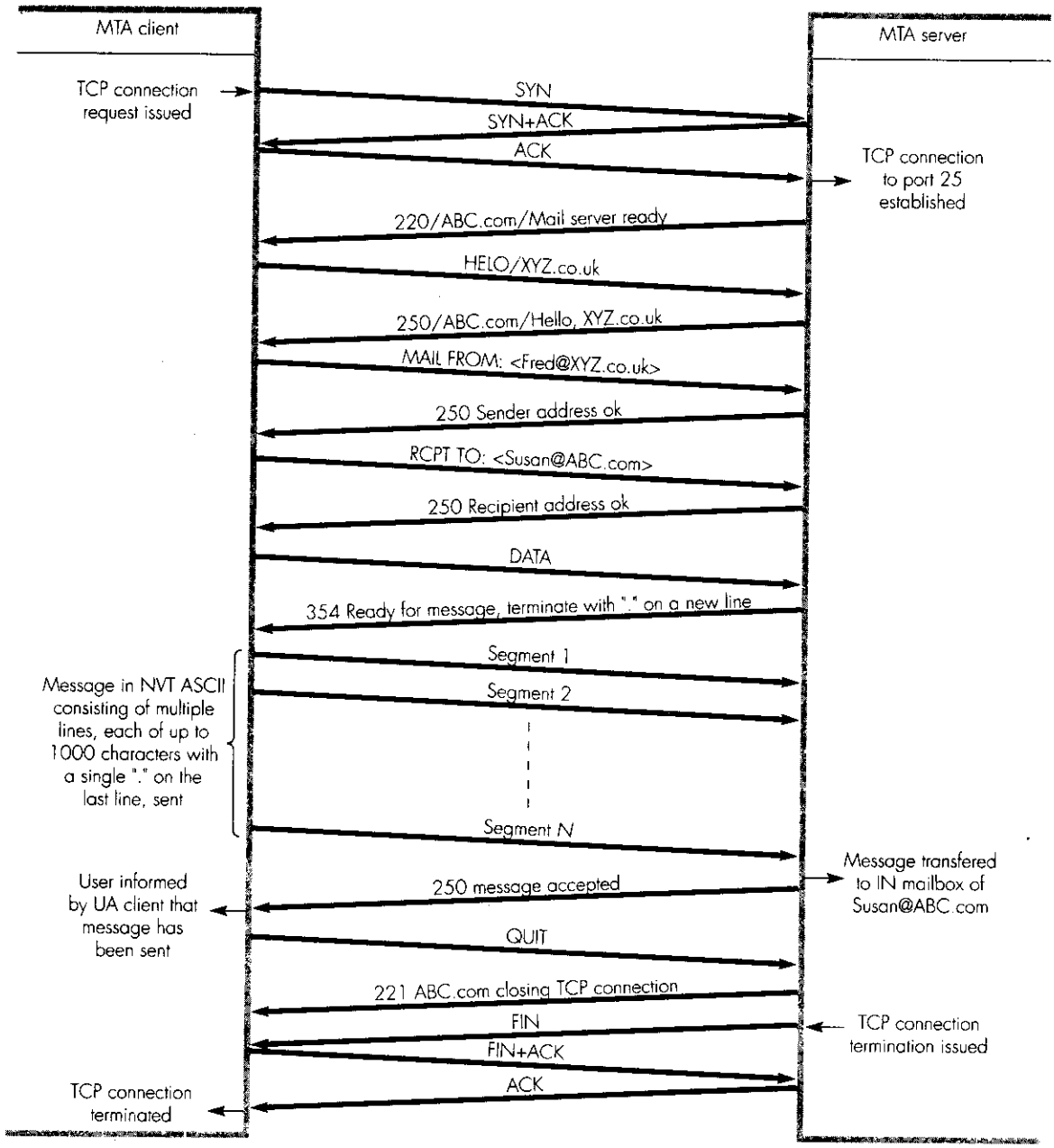


Figure 14.12 Example email message transfer from Fred@XYZ.co.uk to Susan@ABC.com using SMTP.

The UA client in each user terminal periodically sends an enquiry to its local UA server to determine whether any new mail has arrived. Hence when the UA server next receives an enquiry from the UA client in the recipient's terminal, it transfers the received message to the UA client. The latter then places the message in the user's IN mailbox and, typically, outputs a message indicating a new (mail) message has arrived.

At the sending side, once the MTA client receives the final 250 response indicating the message has been transferred successfully, it informs the UA server. The latter then informs the UA client in the sending terminal and this, in turn, informs the user that the message has been sent.

14.4 FTP

The transfer of the contents of a file held on the file system of one computer to a file on another computer is a common requirement in many distributed/networked applications. In some applications the two computers involved may both be large servers each running a different operating system with a different file system and character set. In another application, one of the computers may be a server and the other an item of equipment such as a cable modem or a set-top box which does not have a hard disk. Hence in this case all the data that is transferred must have been formatted specifically for running in the cable modem or set-top box. Clearly, therefore, the file transfer protocol associated with the second type of application can be much simpler than the first. Hence to meet the different requirements of these two types of application, there are two Internet application protocols associated with file transfer. The first is called the **file transfer protocol** (**FTP**) and the second the **trivial file transfer protocol** (**TFTP**). In this section we give an overview of the operation of FTP and we describe the operation of TFTP in the next section.

14.4.1 Overview

FTP is a widely used Internet application protocol that has been designed to enable a user at a terminal to initiate the transfer of the contents of a named file from one computer to another using the TCP/IP protocol suite. The two computers may use different operating systems with different file systems and, possibly, different character sets. It also supports the transfer of a number of different file types such as character and binary. It is specified in **RFC 959**. We describe first how the file contents are represented and then the operation of the protocol itself. We conclude with some examples.

14.4.2 File content representation

Although FTP has been designed to enable files stored in many different types of computer to be transferred, to gain an understanding of FTP's operation without including too much detail, we shall limit our description to file

transfers involving just two different file types, ASCII and binary, and files containing a stream of bytes with no internal structure. As with the contents of email messages, for a file containing 7-bit ASCII characters the file contents are first converted into NVT ASCII by the sending side before they are transferred. They are then converted back again into 7-bit ASCII at the recipient side for storage. With a binary file, the end of the file is signaled by the sending side initiating the closure of the TCP connection.

14.4.3 FTP operation

A schematic diagram showing the essential components involved in a file transfer using FTP is shown in Figure 14.13. The computer initiating the transfer request is called the client and the computer responding to the request the server.

As we can see, each FTP entity consists of two parts: a control part and a data transfer part. The control part is concerned with the exchange of control messages – commands and their replies – relating to the file to be transferred, and the data transfer part with the actual transfer of the file contents.

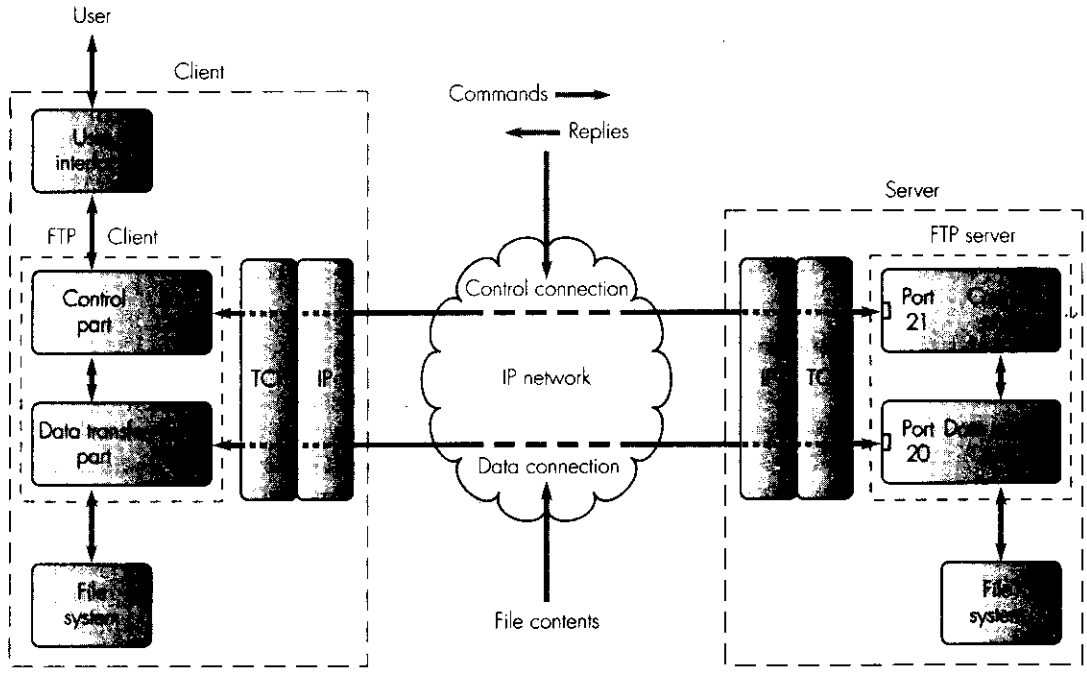


Figure 14.13 FTP components and terminology.

The user interacts with his or her local FTP through an appropriate (user) interface. The user interface software then converts each command that is selected/entered by the user into a standard format that is understood by the FTP control part. There is also a standard format used for each FTP command and response message exchanged by the control part in the two computers.

On receipt of the first command from the user, the (FTP) control part in the client initiates the establishment of a TCP connection between itself and the control part in the server. This is called the **control connection** and it remains in place until the related file transfer has been carried out. The port number at the client side is an ephemeral port – shown as 1216 in Figure 14.13 – and that at the server side port 21 which is the well-known port number for the FTP control connection. The reply message to a command is returned by the control part in the server over the control connection.

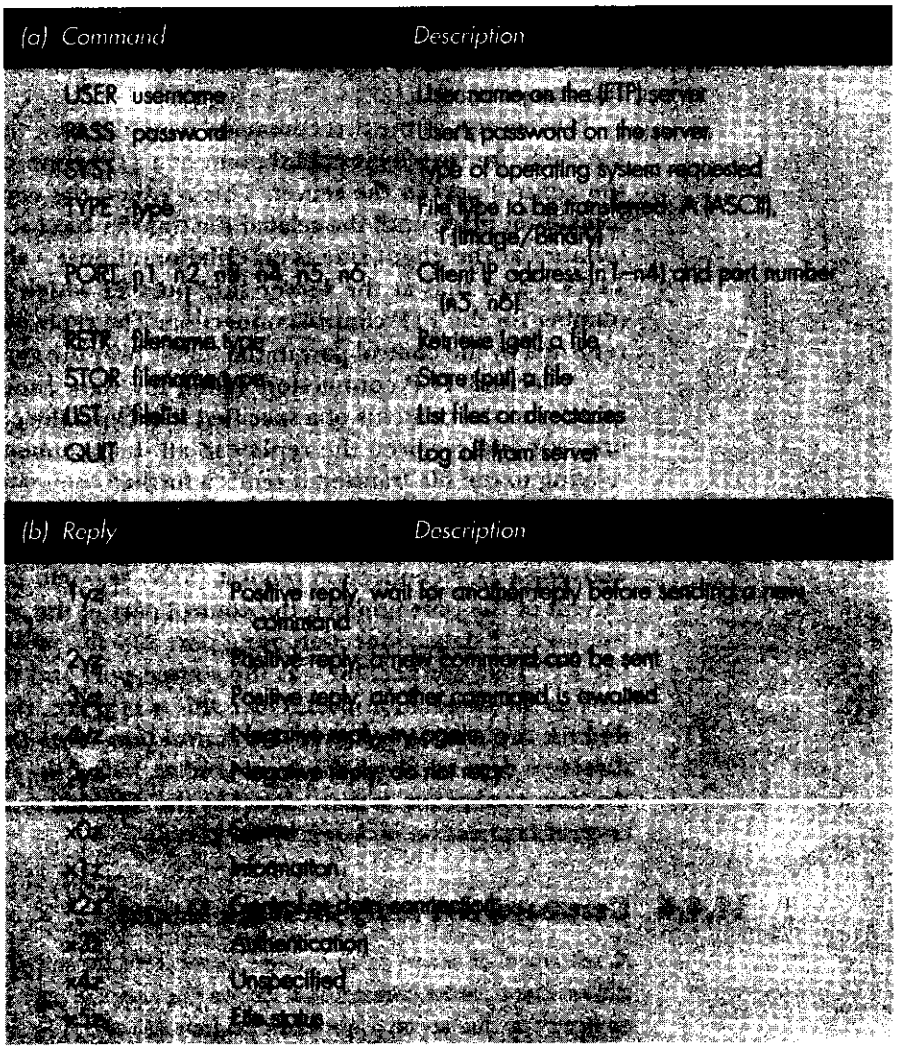
A second TCP connection called the **data connection** is used for the transfer of the contents of a specified file. Once the control part in the client has sent and received the replies to all the command messages it has sent relating to the file transfer, it sends a further command informing the server side of the ephemeral port number that should be used for its side of the data connection. The control part in the client then issues a passive open and waits for a TCP connection request (SYN) segment from the server side. On receipt of the port number, the control part in the server proceeds to establish the TCP data connection using port 20 – the well-known port number for the FTP data connection – as the source port and the received ephemeral port as the destination port. Once this is in place, the contents of the specified file are transferred over this connection. Note that this can be in either direction depending on the command and after the transfer has taken place, the data connection is then closed by the side that sends the data. Finally, the control connection is closed by the client.

14.4.4 Command and reply message format

A selection of some of the more common command messages that are sent across the control connection (from the control part in the client to the control part in the server) are listed in part (a) of Table 14.3 and the structure of the reply messages that are returned in the opposite direction in part (b). All the command and reply messages are made up of ASCII characters. They are encoded for transmission into 8-bit NVT ASCII with each command/reply terminated by a CR/LF pair of characters.

As we can see, all the commands are in upper-case and many have parameters – referred to as arguments – associated with them. Most of the commands are self explanatory. In the case of the *PORT* command, however, the six parameters associated with it (n1–n6) are all decimal numbers. The four decimal numbers n1–n4 form the IP address of the client host in dotted decimal. The two numbers n5–n6 then specify the ephemeral port number

Table 14.3 FTP client–server communication: (a) example commands; (b) structure of the replies.



(a) Command	Description
USER username	User name on the (FTP) server
PASS password	User's password on the server
SYST	Type of operating system requested
TYPE type	File type to be transferred: A (ASCII), I (Image/Binary)
PORT n1, n2, n3, n4, n5, n6	Client IP address (n1–n4) and port number (n5, n6)
RETR filename	Retrieve (get) a file
STOR filename	Store (put) a file
LIST filelist	List files or directories
QUIT	Log off from server

(b) Reply	Description
1yz	Positive reply, wait for another reply before sending a new command
2yz	Positive reply, a new command can be sent
3yz	Positive reply, another command is awaited
4yz	Negative reply, repeat command
5yz	Negative reply, do not repeat
x0z	Syntax
x1z	Information
x2z	Connections
x3z	Authentication
x4z	Unspecified
x5z	File status

for the data connection on the client side. Each port number is 16 bits long and n5 is the decimal equivalent of the most significant 8 bits and n6 the least significant 8 bits. Hence the two parameters n5 and n6 for port 1217 would be 4, 193; that is, $4 \times 256 + 193$.

Each of the reply messages comprises a 3-digit code followed by an optional text message. The first digit indicates the type of reply, positive (successful) or negative. The second digit expands on this by indicating to what the reply relates (control or data connection, data, and so on) and the third digit gives additional information relating to error messages. A selection of

some of the more common reply messages, together with a typical text message, are as follows:

220 FTP server ready
331 Password required for <username>
230 User <username> logged in
215 Server OS Name Type: Version
200 File type acknowledged
200 PORT command successful
150 Opening ASCII/Binary mode data connection for <file name>
226 File transfer complete
221 Goodbye
425 Data connection cannot be opened
500 Unrecognized command
501 Invalid arguments
530 User access denied

14.4.5 Example

In order to illustrate the use of some of the listed commands and their replies we shall show some example message exchanges.

There are three types of file transfer supported over the data connection:

- the transfer of the contents of a named file from the client file system to the server system;
- a similar transfer in the server–client direction;
- the transfer of the listings of the files (or the directories in a file) held by the server and saved in a named file on the client.

A typical exchange of commands and replies to carry out (successfully) the transfer of a named file and file type from the server file system to the client system is shown in Figure 14.14. The following additional comments should be noted when interpreting the exchange sequence:

- The client FTP control part has a resolver procedure linked to it and, when the DNS name of the server is passed to it by the user interface, it uses the resolver to obtain the IP address of the server.
- If the user had issued a put<filename.type>, then the client (control part) would send a STOR<filename.type> command. Also, since the file transfer is in the client–server direction, if the TYPE is I, then the client would initiate the closure of the data connection.

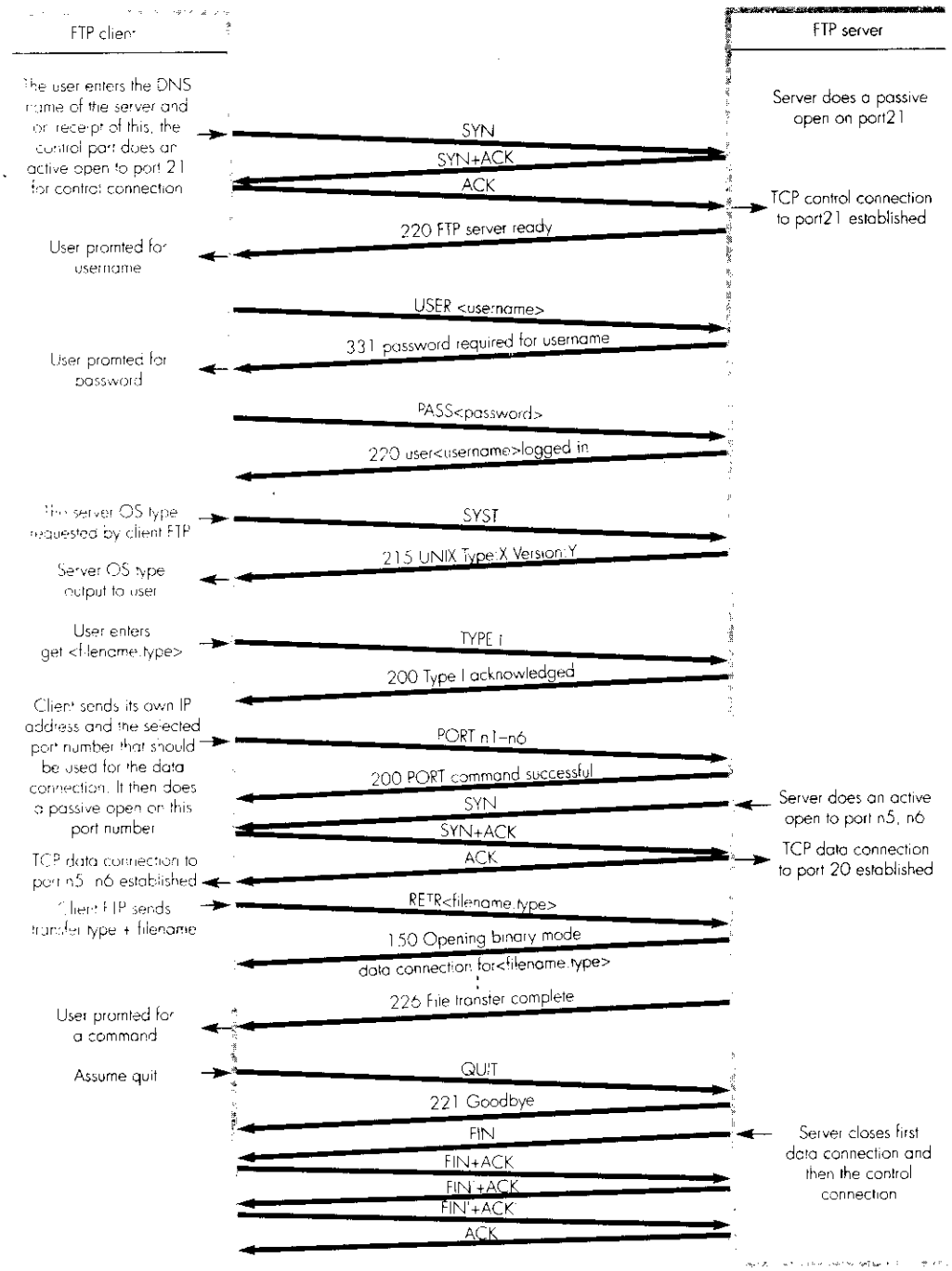


Figure 14.14 Example of command–reply message exchange sequence to get a file from the FTP server.

14.4.6 Anonymous FTP

The example shown in Figure 14.14 assumed that the (client) user had a username/password on the named server. This is not always the case since FTP is also used to access information from a server that allows unknown users to log in to it. To access information from this type of server, the user must know the DNS name of the server but, when prompted for a username, he or she enters *anonymous* and, for the password, his or her email address. Normally, in response, the server replies with something like

230 Visitor login ok, access granted

at which point the same procedure shown in the example in Figure 14.14 follows.

In some instances, however, before granting the user access, the server carries out a rudimentary check that the client host has a valid domain name. Although the IP address of the client host has not been formally sent at this point – this does not occur until the PORT command is sent – it is present in the (IP) source address field of each of the IP datagrams that have been used to set up the (TCP) control connection and to send the username and password. Hence before granting access, the control part in the server uses its own resolver to check that the IP address of the host is in the DNS database. As we saw earlier in the latter part of Section 14.2.5, this involves the control part issuing a pointer query to the resolver with the host IP address in the query name. If a valid domain name is returned, then access is granted as before. If a negative response is received then access to the server is blocked and, typically, the server sends a reply of

530 User access denied, unknown IP address

14.5 TFTP

As we mentioned at the start of Section 14.4, TFTP is used mainly in applications in which one of the two communicating hosts does not have a hard disk. Typically, TFTP is then used to download – normally referred to as bootstrapping – the application code that is to be run on the diskless host. We showed an application of TFTP earlier in Figure 11.5 and, as we explained in the accompanying text, it is used to download the application code for cable modems from the cable modem termination system (CMTS). As we showed in the figure, TFTP uses UDP as the transport protocol since this is less complicated than TCP and hence requires less memory. The specification of TFTP is given in **RFC 1350**.

14.5.1 Protocol

There are just five message types (PDUs) associated with the TFTP protocol and their format is shown in Figure 14.15. The first field in each message is called the *opcode* and indicates the message type. The different types are shown in the figure.

As with FTP the host/item of equipment that initiates a transfer is called the client and the host that responds to the request the server. Each transfer starts with the client making a request to the server either for a named file – read request – or to receive a named file – write request. Hence in an application such as downloading code, the diskless host is the client and the first message that is sent is a *read request* (*RRQ*). The *filename* field in the message is used to specify the name of the file on the server to be transferred/downloaded. The filename is an NVT ASCII string that is terminated by a byte of zero. This is followed by the *mode* field which is also an NVT ASCII string indicating whether the file contents are lines of ASCII text – *netascii* – or a string of 8-bit bytes – *octet*. In both cases the string is terminated by a byte of zero.

The contents of the requested file are transferred by the server in one or more *DATA* messages. Since UDP is a best-effort protocol, normally an error control protocol is used to transfer the complete file contents. The protocol supported is based on the stop-and-wait (idle RQ) error control scheme. It is a variant of the protocol we showed earlier in Figure 6.23, the difference being that there are no NAK messages and hence the retransmission of a lost DATA message is triggered by the timer for the message expiring. An example message exchange illustrating the main features of the protocol is shown in Figure 14.16.

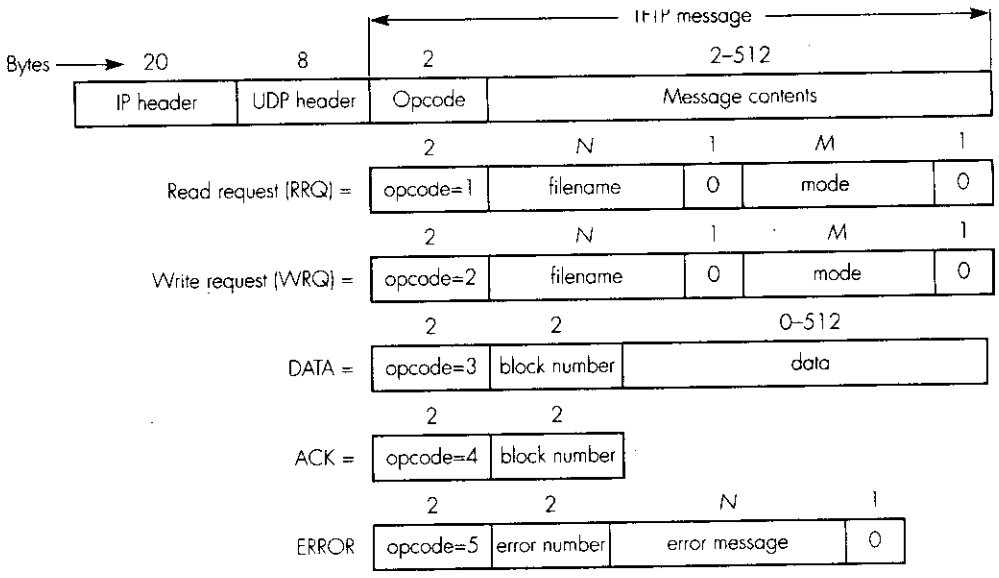


Figure 14.15 TFTP PDU message formats.

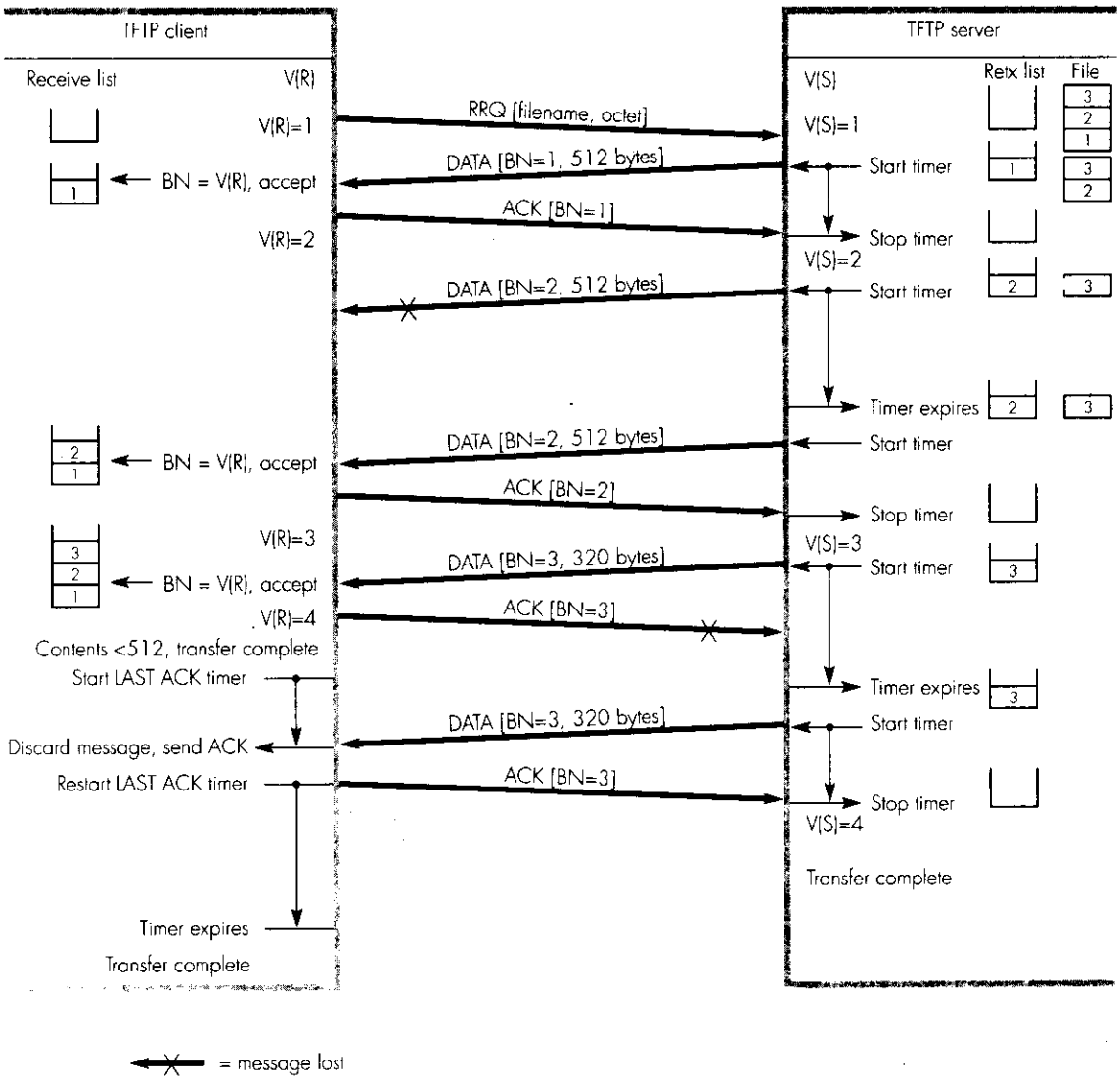


Figure 14.16 TFTP example PDU message exchange.

Recall that with the idle RQ protocol, the total message – the file contents – are first divided into multiple blocks the size of each block determined by the characteristics of the underlying data link/transport service. With the TFTP protocol the maximum size of each block – the *data* field in each DATA message – is 512 bytes. The end of a message/file transfer is then indicated when a DATA message is received containing less than 512 bytes in it, that is, a data field containing from 0–511 bytes.

As with idle RQ, to detect duplicates, each DATA message contains a sequence number – called the *block number* (*BN*) in TFTP – in the message header and, on receipt of each error-free DATA message, an *ACK* message is returned containing the same BN within it as that contained in the DATA message. To implement the scheme, the server side maintains a send sequence variable, V(S), which contains the BN that is to be allocated to the next DATA message to be transmitted, and the client side maintains a receive sequence variable, V(R), which indicates the BN in the next in-sequence DATA message it expects to receive. Both are initialized to 1.

The transfer starts with the client sending a RRQ message containing the filename and file type. In response, the server proceeds to send the contents of the filename. In the example file transfer shown in Figure 14.16 the file contents are assumed to require three DATA messages shown as 1, 2, and 3. The following should be noted when interpreting the sequence shown:

- The first DATA message (BN = 1) is assumed to be received and acknowledged correctly and hence both V(S) and V(R) are now incremented to 2.
- The second DATA message (BN = 2) is corrupted and hence is not received. This can be due, for example, to the IP datagram containing the DATA message being discarded during its transfer or the checksum in the UDP header failing in the client.
- At the server side, the absence of an ACK for BN2 means that the retransmission timer expires and another attempt is made to send it.
- This time it is assumed to be received correctly and both V(S) and V(R) are now incremented to 3.
- When the last DATA message is sent (BN = 3), this is received free of errors and hence V(R) is incremented to 4 and an ACK is returned with BN = 3.
- During its transfer, the ACK is corrupted/lost and hence the retransmission timer in the server expires. The server retransmits another copy of BN3 which is assumed to be received error free.
- The client determines from the BN that the message is a duplicate – BN = 3 instead of 4 – and hence discards it but returns an ACK to stop the server from sending another copy.
- The client determines that the file has now been received by the fact that the contents of BN3 are less than 512 bytes.
- The LAST ACK timer is used to allow for the possibility of the last ACK being lost.
- Should the number of attempts to send a block exceed a defined limit, then an error message is sent and the transfer aborted.

In the case of a write request, the client sends a WRQ message with the filename and mode in it and, if the server is prepared to accept the file, it returns an ACK message with a BN = 0. The client side then proceeds to send the file contents using the same procedure as for a read request.

A number of error messages are also provided. The particular error message is indicated by the value in the *error number* field. The contents in the *error message* field then contain an additional text message in NVT ASCII that is terminated by a byte of zero. In addition, even though there is no authentication procedure associated with TFTP, in most implementations the server only allows the client to read or write from/to a specific file name or names.

14.6 Internet telephony

Unlike email which uses the services provided by the message transfer system as an intermediary between the two (or more) communicating participants, Internet telephony requires a communications path that supports real-time communications between the two or more participants involved in the call/session. Also, although the early IETF standards relating to Internet telephony were concerned with providing a basic two-party telephony service between two IP hosts, the more recent standards provide a more versatile facility supporting multiparty calls/sessions involving audio, video, and data integrated together in a dynamic way. The number of participants involved can vary as the session proceeds. Also, the location of each participant is not necessarily at a fixed IP address. For example, at one point in time a participant may be using a workstation attached to an enterprise network while at another time using a PC at home or, possibly, a fixed or mobile phone.

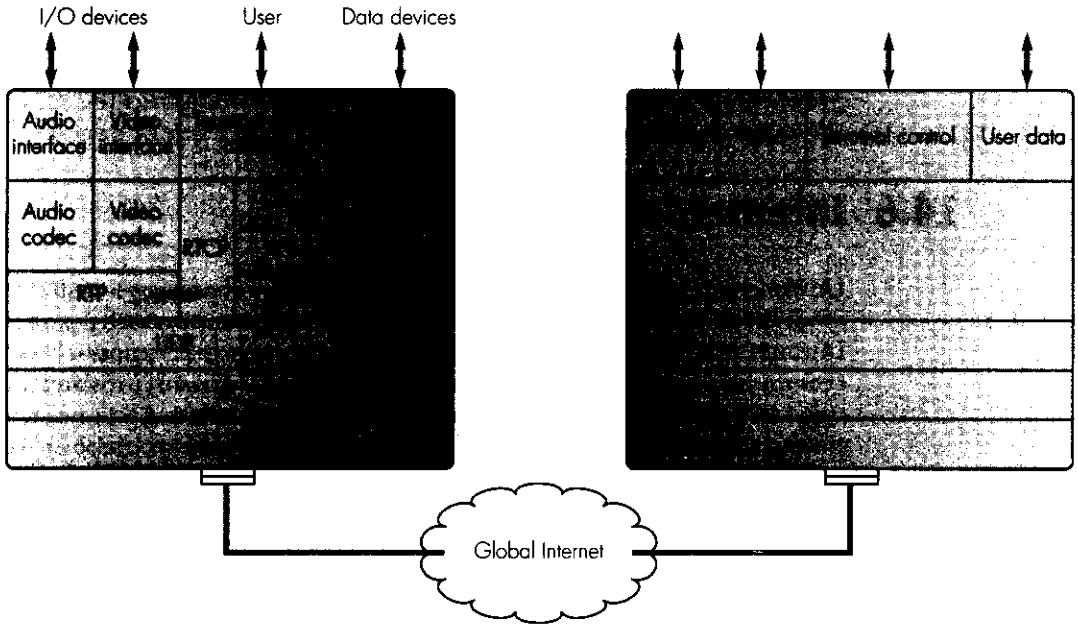
Thus the main requirement associated with Internet telephony is a set of signaling protocols that support, in addition to call/session establishment, features for dynamic user location and the negotiation of a suitable set of capabilities that are supported by all the user devices involved. In this section we describe the main features of three of the protocols that have been defined by the IETF to provide these services. These are the **session initiation protocol** (**SIP**), the **session description protocol** (**SDP**), and the **gateway location protocol** (**GLP**).

14.6.1 SIP

SIP provides services for user location, call/session establishment, and call participation management. It is a simple request-response – also known as transaction – protocol and is defined in **RFC 2543**. The user of a host device that wants to set up a call sends a request message – also known as a command or method – to the user of the called host device which responds by returning a suitable response message. Both the request and response are made through an application program/process called the **user agent** (**UA**) which maps the request and its response into the standard message format used by SIP.

Each UA comprises two parts, a **UA client** (**UAC**), which enables the user to send request messages – to initiate the setting up of a call/session for example – and a **UA server** (**UAS**) which generates the response message determined by the user's response. A schematic diagram showing a typical stack associated with Internet telephony is shown in Figure 14.17(a) and a list of the request and response messages used by SIP – together with their usage – is shown in part (b).

(a)



RTP = real-time transport protocol
RTCP= real-time transport control protocol
UAC = user agent client
UAS = user agent server
SIP = session initiation protocol

(b)

SIP message	Usage
INVITE	Invites a user to join a call/session
ACK	Used to acknowledge receipt of an INVITE response messge
REGISTER	Used to inform a SIP redirect server of the current location of a user
OPTIONS	Used to request the capabilities of a host device
CANCEL	Terminates a search for a user
BYE	Inform the other user(s) that the user is learning a call/session

Figure 14.17 Internet telephony: (a) example host device protocol stack; (b) SIP request/response signaling message types.

Each request and response message comprises a header and a body. The header contains a set of fields a number of which are similar to those used with email. For example, a selection of the header fields associated with the INVITE request/response message include:

To:	The SIP address of the called participant
From:	The SIP address of the caller
Subject:	A brief title of the call
Call-ID:	Unique call identifier assigned by the caller
Require:	List of capabilities the host device can support
Content-Type:	Type of information in the message body
Content-Length:	Length of body contents

Each SIP name/address is similar to an email name/address with the addition that it has a prefix of *sip*. Hence two example SIP name/addresses might be *sip:tom.C@university.edu* and *sip:karen.S@company.com*.

The type of call/session being set up is determined by the contents of the message body which describes the individual media streams to be used in the call. These are defined using the companion SDP protocol which we describe in the next section.

As we have just seen, a SIP name/address is similar to an email name/address. Hence before a SIP message can be sent over the Internet it is necessary first to obtain the IP address of the intended recipient host. As we indicated earlier, a user may be contactable at a number of alternative locations. Hence when a user registers to use the Internet telephony service, the user provides a number of alternative SIP addresses where it can be located. For example, karen.S may be contactable at either *sip:karen.S@company.com* or *sip:karen.S@organization.org*. This list of names is then sent to a server called the *redirect server* at each of the sites involved – for example, *company.com* and *organization.org* – using a REGISTER message with the list of alternative addresses in the *Contact:* header field.

To explain how a call/session is set up we shall use two examples. In the first, we assume the called user is currently at the given SIP address. Figure 14.18(a) shows the protocols and network components that are used to set up the call. To simplify the description, assume that the SIP name/address of the calling user is *sip:tom.C@university.edu* and that of the called user is *sip:karen.S@company.com*.

Associated with each access network – enterprise network, campus, and so on – is a second server called a *proxy server* to which all SIP INVITE messages are sent. In the figure we show these as PS-A and PS-B. Each host knows the IP address of its local proxy server. Also, the proxy server knows the SIP name and address of each user that is currently logged in at the site and the IP address of the user's host device. Typically, the latter is determined using ARP.

On receipt of an INVITE request message from the calling host, PS-A first reads the SIP name/address from the *To:* field in the message header and proceeds to obtain the IP address of the proxy server for *company.com* (PS-B) using the domain name service (DNS). On receipt of this, the SIP in PS-A sends the INVITE request message to PS-B. The latter first reads the SIP name/address from the *To:* field in the message header and determines from

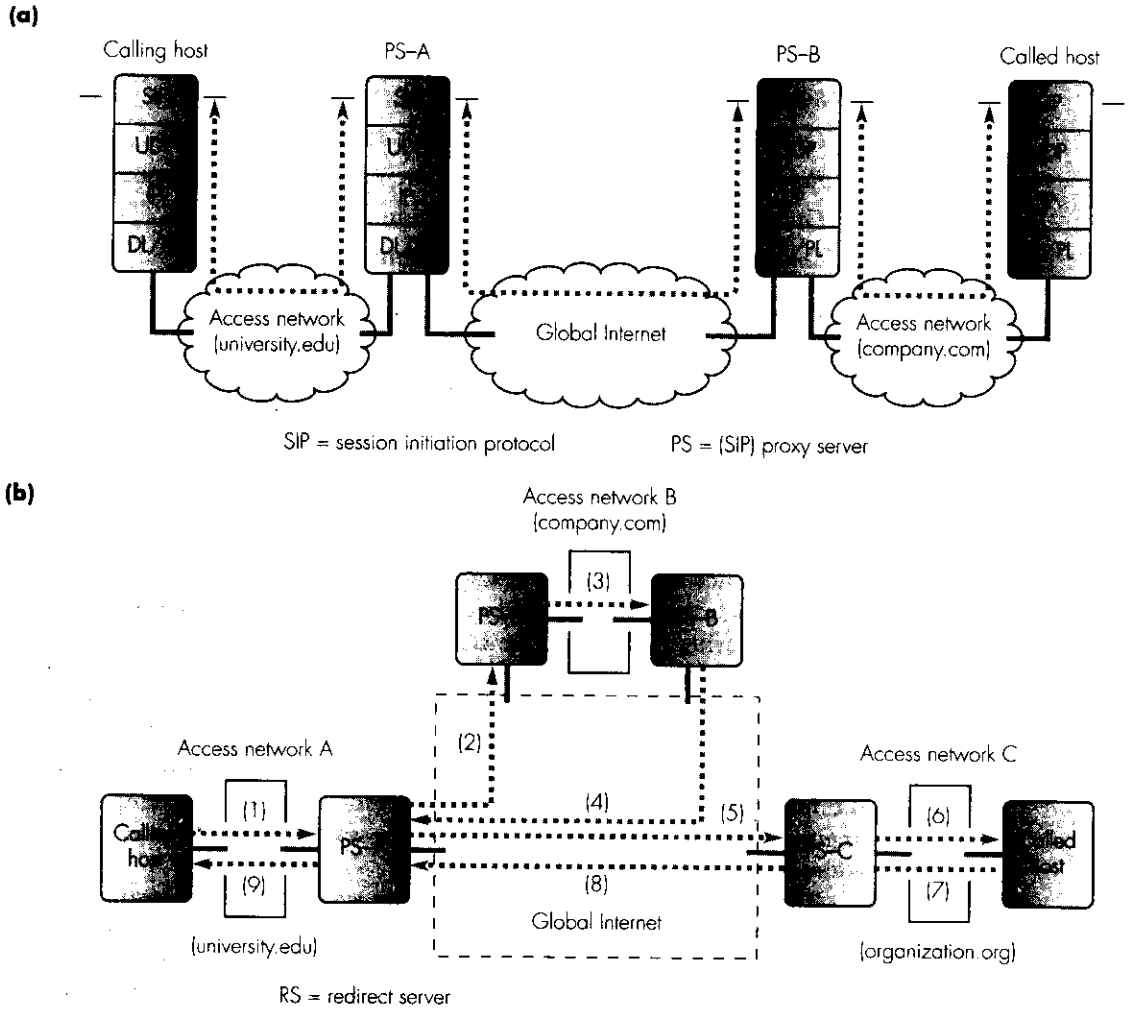


Figure 14.18 SIP message routing examples: (a) direct using proxy servers; (b) indirect using a redirect server.

this firstly, that *karen.S* is currently logged in at this location and secondly, the local IP address of the called host. It then uses the IP address to send the INVITE request to the called host. Assuming the latter is able to accept the call, an INVITE response message is returned over the same path. On receipt of this, the SIP in the calling host returns an ACK message and it is at this point that the two users/hosts can start to exchange the information relating to the call.

In the second example we assume that the called user *sip:karen.S* is currently located at *organization.org* not *company.com*. Hence this time, on receipt of the INVITE request from PS-A, PS-B determines that *karen.S* is not

currently logged in at this location and hence forwards the request to the redirect server for the site, RS-B. As we indicated earlier, the latter has the list of alternative SIP addresses for this user and determines from this that *karen.S* can also be located at *organization.org*. The SIP in RS-B returns this information in a *Contact:* header field in the INVITE response message. On receipt of this, PS-A proceeds as before but this time by first obtaining the IP address of the SIP proxy server at *organization.org* using the DNS. A summary of the message sequence involved is shown in Figure 14.18(b).

14.6.2 SDP

As we indicated earlier, the role of the session description protocol (SDP) is to describe the different media streams that are to be involved in a call/session and also additional information relating to the call. This is described in each SIP message body in a textual format and includes:

- media streams: a multimedia call/session may involve a number of different media streams including speech, audio, video, and more general data. The proposed list of media types and their format are contained in the message body. Each SIP INVITE request message contains a list of the media types and the compression formats that are acceptable to the calling user (host device) and the INVITE response message contains a possibly modified version of this that collectively indicates what is acceptable to the called user;
- stream addresses: for each media stream, the destination address and UDP port number for sending and/or receiving each stream is indicated;
- start and stop times: these are used with broadcast sessions and enable a user to join a session during the time the broadcast is being carried out.

14.6.3 GLP

The two examples we considered in Section 14.6.1 assumed that the two host devices – or more if multicast addresses are used – were both attached directly to the Internet. In some instances, however, one of the host devices may be attached to a different network such as a PSTN or ISDN. Such cases require a device called a gateway to convert the different signaling messages – a **signaling gateway** (**SGW**) – and the different media formats – a **media gateway** (**MGW**).

In practice, both the PSTN/ISDN and the Internet are global networks/internetworks. Hence when the called user is attached to a PSTN/ISDN, because of the potentially higher bandwidth associated with the Internet, it is preferable to utilize the Internet for as much of the connection path as possible. To do this requires, firstly, a gateway associated with each PSTN/ISDN access network and secondly, when a call is made, to utilize the gateway that is closest to the called – or calling – user. As we saw earlier in Figure 9.9, the

Internet is composed of an interconnected set of networks/internetworks. Typically, therefore, each Internet regional/national network has a number of gateways associated with it. Also, each of these networks has one or more **location servers** (**LS**). This general architecture is shown in Figure 14.19.

As we can deduce from the figure, when the called user device is attached to a segment/subnet of a PSTN/ISDN, the SIP INVITE message must be sent to the (signaling) gateway that is nearest to the segment/subnet to which the called terminal equipment is attached. The main issue, therefore, is, given a conventional telephone number, how is the INVITE message routed to the gateway that is nearest to the subscriber with this number? This is the role of the location servers and their operation is as follows.

Each gateway knows the regional/national code of the segment of the PSTN/ISDN to which it is attached. Typically this is entered by network management. In addition, on the Internet side, each gateway has an IP address and it also knows the IP address of the LS(s) that is (are) attached to the same (Internet) regional/national network. Each gateway then uses the IP address of each of its local LSs to inform them of the regional/national code of the segment of the PSTN/ISDN to which it is attached. In this way, each LS

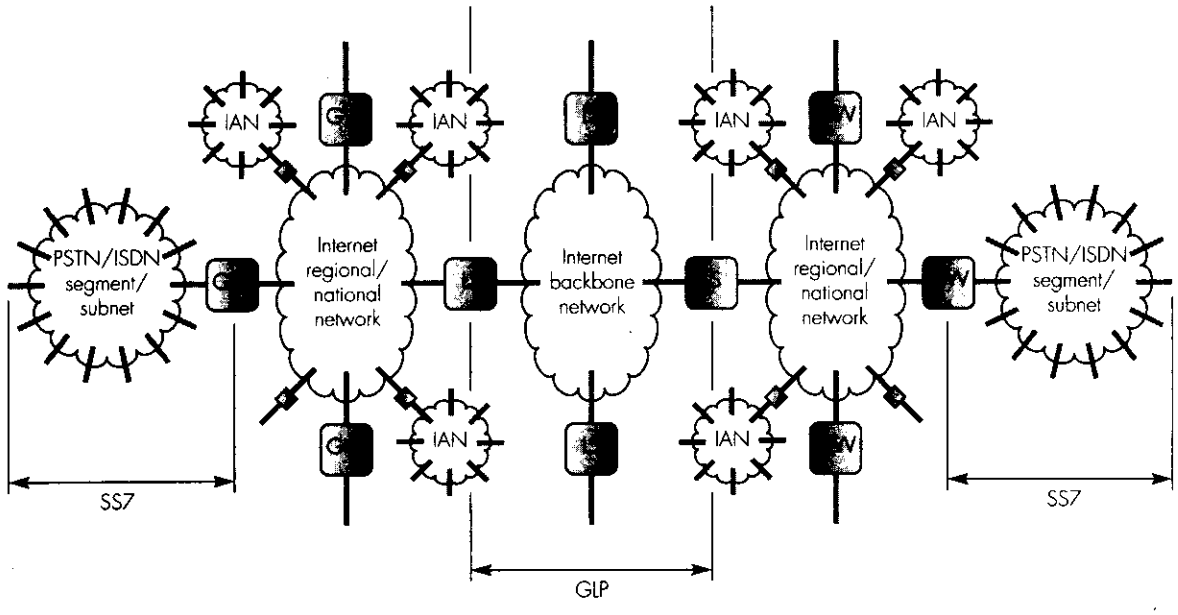


Figure 14.19 Interworking between Internet hosts and PSTN/ISDN terminal equipment.

learns the regional/national codes of the segments of the PSTN/ISDN to which all of its gateways are attached and, from this, the IP address of the gateway that should be used for each of these codes.

Each LS then exchanges this information with each of the other LSs using the gateway location protocol (GLP). In this way each LS builds up a database of the IP address of the LS that should be used to reach all of the gateways in other regional/national networks and also the PSTN/ISDN codes associated with each of these gateways. Thus the gateway local protocol which carries out this function is very similar to the interdomain routing protocol BGP – the border gateway protocol – which we described earlier in Section 9.6.5. Indeed, GLP is based on BGP and hence we shall not expand upon it here.

14.7 SNMP

The simple network management protocol (SNMP) is concerned not with providing Internet-wide application services to users – SMTP, FTP, and so on – but rather with the management of all the networking equipment and protocols that make up the Internet. As we saw in earlier chapters, the Internet is composed of a range of different items of networking equipment. These include LAN bridges, subnet routers, access gateways, interior gateways/routers, exterior gateways, communication links/subsystems, and so on, all of which need to be functioning correctly.

Clearly, in any networking environment, if a fault develops and service is interrupted, users will expect the fault to be corrected and normal service to be resumed with a minimum of delay. This is often referred to as **fault management**. Similarly, if the performance of the network – for example, its response time or throughput – starts to deteriorate as a result of, say, increased levels of traffic in selected parts of the network, users will expect these to be identified and additional equipment/transmission capacity to be introduced to alleviate the problem. This is an example of **performance management**. In addition, most of the protocols associated with the TCP/IP suite have associated operational parameters, such as the time-to-live parameter associated with the IP protocol and the retransmission timer associated with TCP. As a network expands, such parameters may need to be changed while the network is still operational. This type of operation is known as **layer management**. Others include **name management**, **security management**, and **accounting management**.

Associated with each managed element – a protocol, bridge, gateway, and so on – is a defined set of management-related information. This includes variables – also known as **managed objects** – that can be either read or written to by the network manager via the network. It also includes, when appropriate, a set of **fault reports** that are sent by a managed element when a related fault occurs. In the case of IP, for example, a read variable may relate to, say, the number of IP datagrams/packets discarded when the time-to-live parameter expires, while a write variable may be the actual time-to-live timeout value. Similarly, in the case of an exterior gateway, if a neighbor gateway ceases to respond to hello

messages, in addition to modifying its routing table to reflect the loss of the link, the gateway may create and send a fault report – via the network – to alert the management system of the problem. If the management system receives a number of such reports from other neighbors, it can conclude that the gateway is probably faulty and not just a communications line failure.

SNMP is an application protocol so a standard communication platform must be used to enable associated messages – PDUs – to be transferred concurrently with the messages relating to user services. To achieve this, normally SNMP uses the same TCP/IP protocols as the user application protocols. The general scheme is shown in Figure 14.20.

The role of the SNMP is to allow the **manager process** in the manager station to exchange management-related messages with the management processes – each referred to as a **management agent** – running in the various managed elements: hosts, gateways, and so on. The management process in

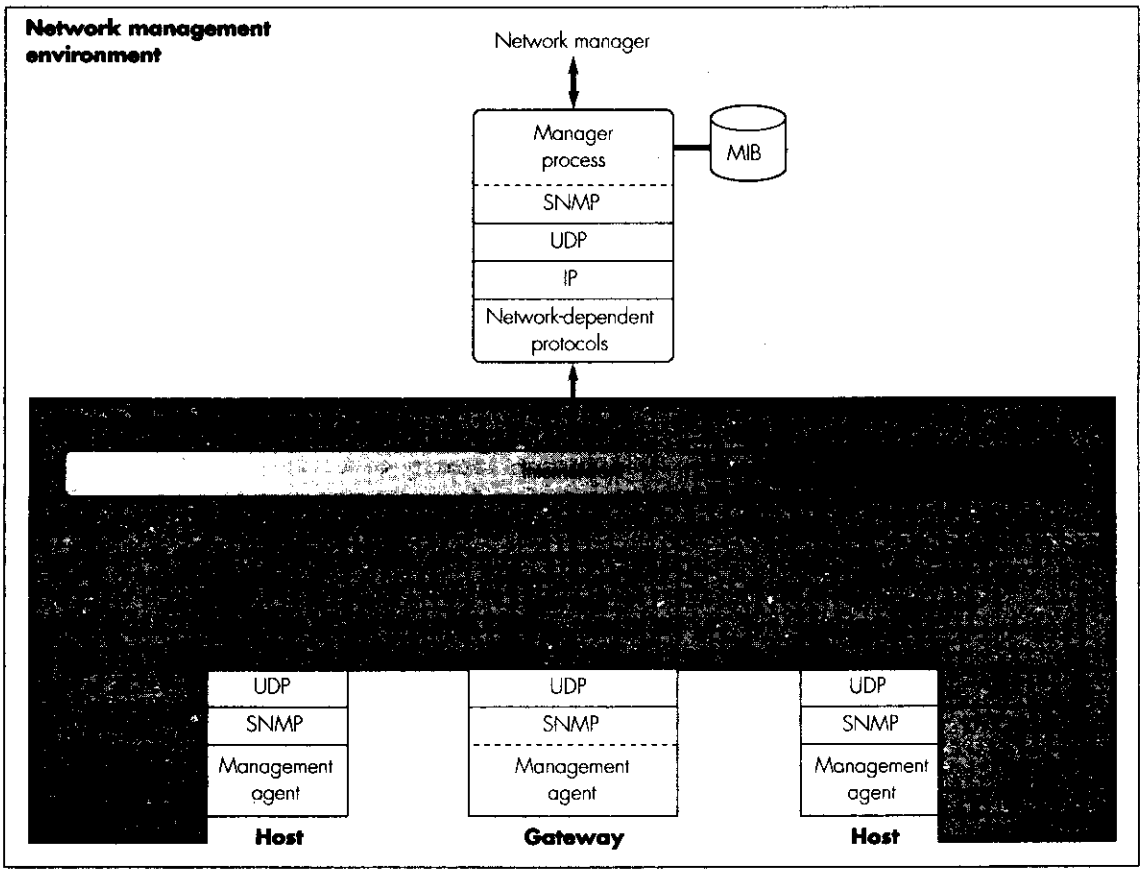


Figure 14.20 Network management schematic and terminology.

these elements is written to perform the defined management functions associated with that element. Examples include responding to requests for specified variables (counts), receiving updated operational variables, and generating and sending fault reports.

Management information associated with a network/internet is kept at the network manager station (host) associated with that network/internet in a **management information base (MIB)**. A network manager is provided with a range of services to interrogate the information in the MIB, to initiate the entry and collection of additional information, and to initiate network configuration changes. Clearly, the manager station is the nerve center of the complete network, so strict security and authentication mechanisms are implemented. Normally, there are various levels of authority depending on the operation to be performed. In large internetworks like the Internet, multiple manager stations are used, each responsible for a particular part of the Internet. Examples include each campus/enterprise access network, each regional/national network, the global backbone network and its gateways, and so on. We shall describe first the structure of the management information associated with the Internet and then the operation of the SNMP protocol.

14.7.1 Structure of management information

The management agent software in each networking element maintains a defined set of variables – managed objects – which are accessible by the network manager process using SNMP. In some instances, a variable can only be read (read-only) and in others it can also be written to (read-write). The MIB contains a similar set of variables/objects each of which reflects the current value/state of the same variable in the managed element.

Clearly, in a large network/internet, the managed equipment may come from a variety of different vendors, each with its own preferred processor/ microprocessor. Hence, since all the management information relating to each item of equipment is to be processed by a single management process – often running in a computer that has a different processor from that used in the managed equipment – it is essential to ensure that the management information relating to each item of equipment has the same meaning in both the equipment and the manager station. As we explained in section 13.2, one way of achieving this is to define the data types of each of the managed objects associated with each item of equipment using ASN.1. Then, to ensure that the value(s) associated with each managed object is interpreted in the same way in both the equipment and the manager station, before each value is transferred over the network it is first converted into the related standard transfer syntax using the basic encoding rules associated with ASN.1. This is now the standard approach used with most network management systems including that associated with the Internet.

The current version of the MIB for the Internet is **MIB-II** and is defined in **RFC 1213**. The data types of all the variables (managed objects) in the MIB are a subset of the ASN.1 types we described in Section 13.2.1. These are listed in Figure 14.21(a). In addition, a number of subtypes are used and a selection of these, together with a description of their use, is given in Figure 14.21(b).

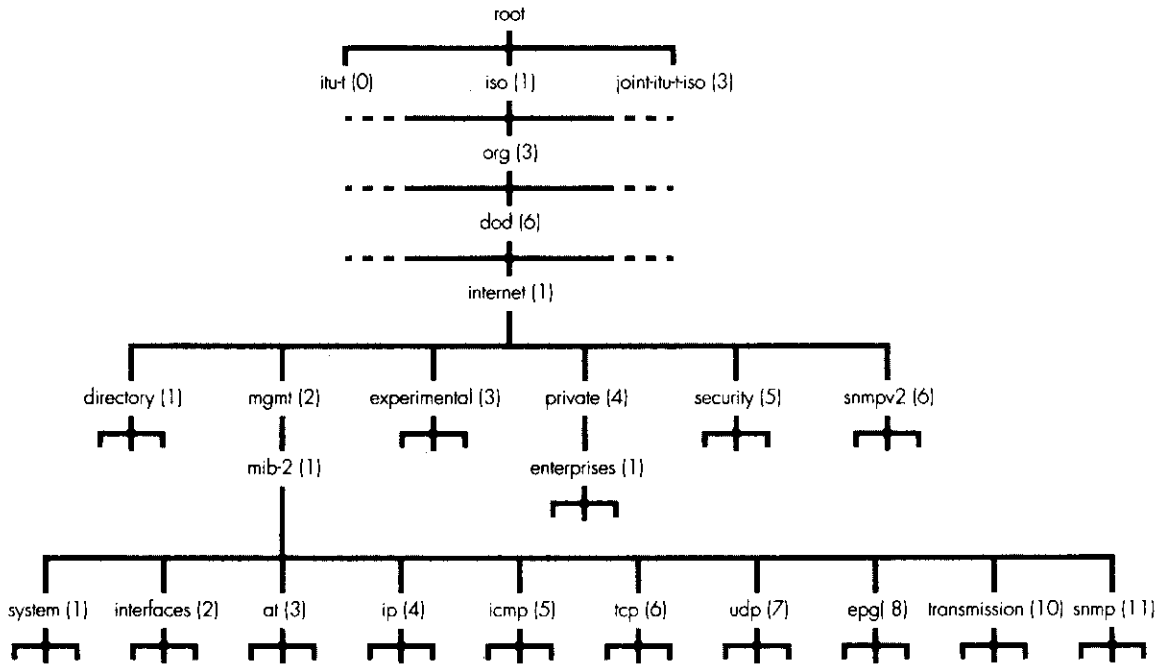
(a)

Simple types:	**Constructed types:**
INTEGER	SEQUENCE
BITSTRING	SEQUENCEOF
OCTETSTRING	CHOICE
Display String	
NULL	
OBJECT IDENTIFIER	

(b)

Subtypes:		
IpAddress	–	OCTETSTRING of length 4 (IP address in dotted decimal)
PhyAddress	–	OCTETSTRING of length 6 (MAC address)
Counter32	–	a 32-bit counter, that increments modulo 2^{32}
Gauge32	–	an unsigned integer in the range 0 to $2^{32}-1$
Integer32	–	a 32-bit INTEGER
TimeTicks	–	a 32-bit counter that increments at 1/100s intervals

(c)



Internet identifier = 1.3.6.1 mib-2 managed object identifier = 1.3.6.1.2.---

Note: at group not used in MIB-II and there is no group (9)

Figure 14.21 Structure of management information: (a) ASN.1 types used; (b) subtypes; (c) portion of ASN.1 object naming tree relating to the Internet.

As we explained at the end of Section 13.2.1, the OBJECT IDENTIFIER data type is used to identify a managed object within the context of an internationally defined object naming tree. The portion of this tree that relates to the Internet MIB is shown in Figure 14.21(c).

As we can see, each branch node in the tree is identified by means of a label and a number. A specific node is then identified by listing either the label – together with its number – or simply the number of each branch node starting at the root. In this way, all the managed objects within the total Internet MIB can be uniquely identified and all start with

iso(1) org(3) dod(6) internet(1) mgmt(2) mib-2(1) ...

or, more usually,

1.3.6.1.2.1 ...

Note that the address translation (at) group – arp and rarp – is not present in MIB-II as this is now part of the ip group. Note also that there is no group (9) and that path 1.3.6.1.4.1 is defined for vendor-specific MIBs. This is necessary to ensure that there is no ambiguity when equipment from a number of different vendors is being used.

Each item of equipment that is to be managed, and the various managed objects associated with it, are defined using the object name tree shown in Figure 14.21(c) as a template. The system(1) group contains a number of variables that include the name (textual description) of the equipment, the vendor's identity, the hardware and software it contains, the domain name of the equipment, and its location on the Internet. All the management information that is subsequently obtained from this location relates to the given named item of equipment in the object name tree.

Each managed object can be defined either as an individual entity or, more usually, as a member of a larger group of related objects. Also related groups of objects can be defined in the form of a module. For example, a group may contain all the managed objects (variables) associated with a particular protocol such as IP while a module may contain the complete set of managed object groups associated with a particular item of equipment.

Each managed object definition is in the form of a macro with a minimum of four defined parameters associated with it. An example of an object/variable definition relating to the eighteenth variable in the IP group is as follows:

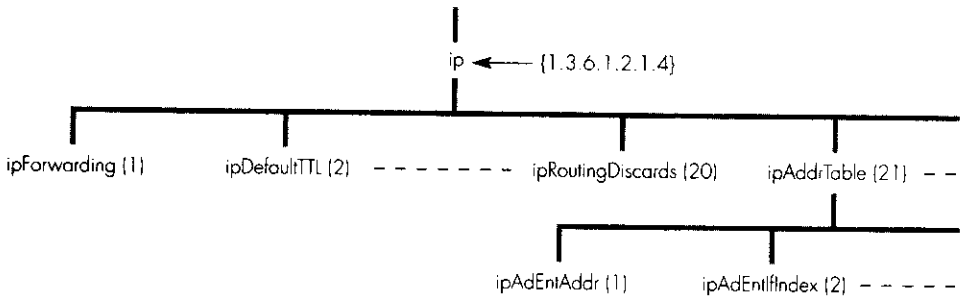
IpFragFails OBJECT TYPE
SYNTAX Gauge32 -- a count value up to $2^{32}-1$
MAX-ACCESS read-only -- the manager station can only read this object
STATUS current -- the object is currently supported
DESCRIPTION "Number of IP datagrams described because don't fragment flag set"
::={ip18}

As we can see, the object/variable name precedes the reserved word OBJECT TYPE. The meaning of the four required parameters associated with each object definition are:

- SYNTAX: this defines the data type of the object;
- MAX-ACCESS: defines whether the variable is read-only or read-write (as viewed from the manager station);
- STATUS: indicates whether the variable is current or obsolete;
- DESCRIPTION: an ASCII string describing what the object is used for. When the macro is invoked, the final ::= sign places the variable into the object name tree of the device.

When a number of groups are collectively defined in a module, the module is defined using a macro that starts with MODULE-IDENTITY. Its parameters include the name and address of the implementer of the module and its revision history. This is followed by an OBJECT-IDENTITY macro, which identifies where the module is located in the object name tree, and a list of OBJECT TYPE macros.

Each managed object in the MIB is uniquely identified and some examples relating to the *ip* group are shown in Figure 14.22. As we can see, an object can be either a simple variable with a single value – for example *ipForwarding(1)* – or a table containing a set of variables – for example *ipAddrTable(21)*. The object identifier of a simple variable in the MIB is the name/identifier of the variable/object with .0 appended to it. For example, the simple variable *ipForwarding* – which indicates whether the system is forwarding IP datagrams (=1) or not (2) – is accessed using either *ipForwarding.0* or, more specifically, as 1.3.6.1.2.1.4.1.0 since this is the form used by the



ipDefaultTTL identifier = 1.3.6.1.2.1.4.2.0

ipAdEntAddr identifier = 1.3.6.1.2.1.4.21.1

Figure 14.22 Example MIB objects in the ip group.

SNMP protocol to transfer the value over the Internet. In the case of a table of values, normally the index used is the name/identifier of the table and this is then followed by a string of get-next value commands until the complete table of values has been obtained. For example, assuming the table *ipAddrTable(21)* – which contains the list of addresses (IP address, subnet mask) and other information – this would be accessed using first *ipAddrTable* – 1.3.6.1.2.1.4.21.

14.7.2 Protocol

As we can deduce from the previous section, the management of an item of equipment – host, router, and so on – involves the manager process reading the current value of a defined set of variables (managed objects) that are being maintained by the agent process in the equipment and also with transferring a value to the agent for writing into a given variable. It also involves receiving fault reports from the agent in the equipment should these occur. To perform these functions, there is a set of request-response messages supported by the SNMP and also a separate command – known as a trap – message for fault reporting. The list of message/PDU types used in SNMPv1 are defined in RFC 1157. They are summarized in Figure 14.23(a).

Each SNMP message is transferred over the Internet as a separate entity using UDP. The UDP well-known port number of the SNMP in the agent for the three request messages is port 161 and that in the manager station for trap messages port 162. As we can deduce from this, the use of UDP as the transport protocol means that there is no guarantee that a message is delivered. Hence when this is deemed to be necessary, a timer is often used and, if a response is not received within a defined time interval, the request is resent.

The role of each of the five messages that are used is as follows:

- *Get-request*: this is used by the manager to get the current value(s) of one or more named variables from an agent. The agent then returns the value(s) using a *Get-response* message. The name/identity of each variable is in its numeric form and each value in the response message is encoded using the basic encoding rules of ASN.1 As we saw in Section 13.2.2, each returned value is in the form of a variable-length byte/octet string comprising a type, length, and value field. In the case of the *Get-request* message, the type of each value field(s) is set to Null;
- *Get-next-request*: this is used by the manager to get the next variable that is located in the MIB name tree immediately following each variable in the list of names in the message. These are then returned in a *Get-response* message. This type of message is used primarily to obtain the consecutive values relating to a table variable;
- *Set-request*: this is used by the manager to write a given set of values into the corresponding named variables;

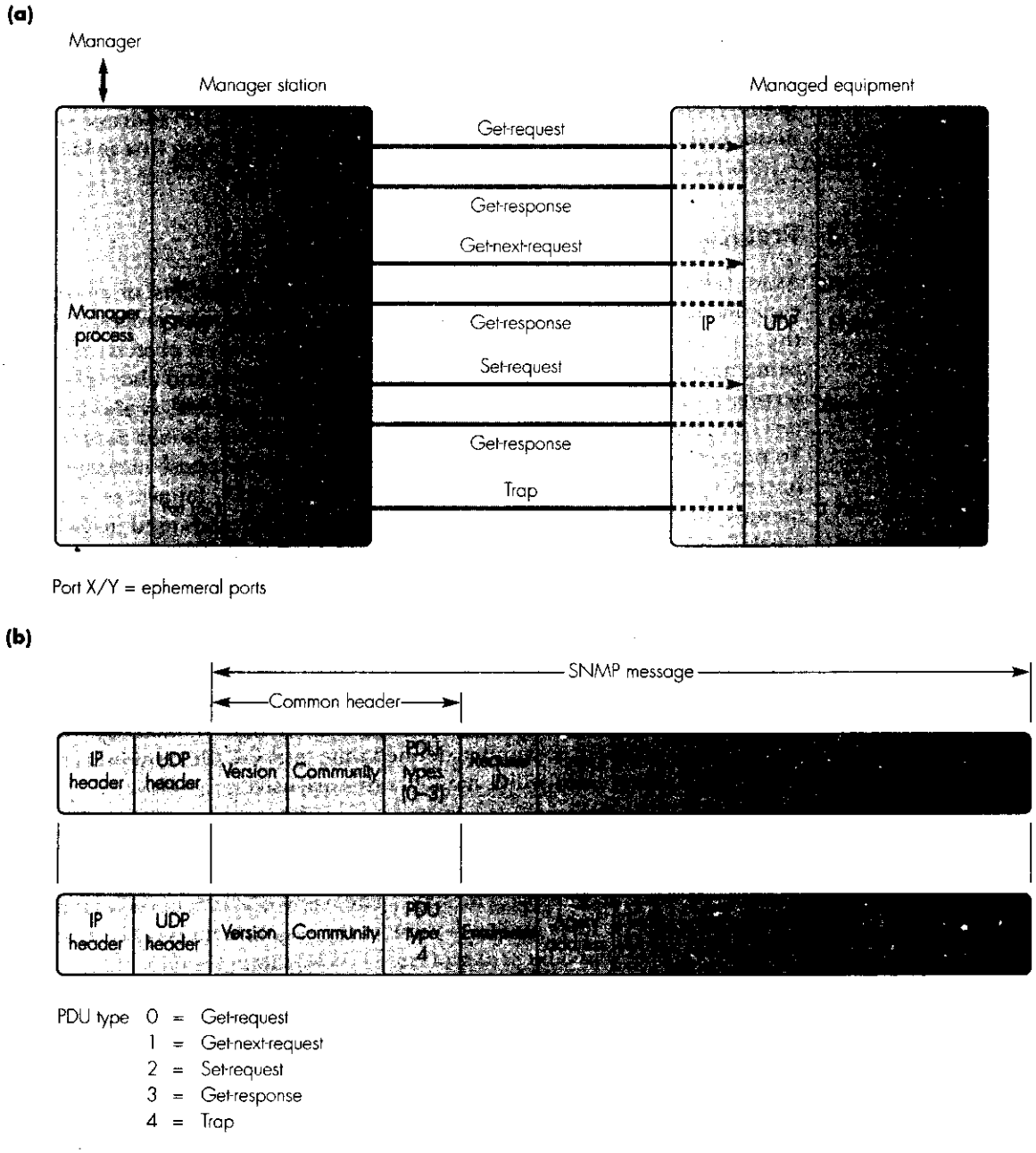


Figure 14.23 SNMPv1 messages/PDUs: (a) types and their sequence; (b) formats.

- *Trap*: this is used by the agent in the equipment identified in the *enterprise* field to notify the manager of the occurrence of a previously defined event. The event type is specified by the value in the *trap type* field together with the value in the specific code field. The time of occurrence of the event is specified in the *time-stamp* field and, where appropriate, a number of related variable values may be returned.

The *community* field in the common header contains a character string that is a password in cleartext exchanged by the manager and agent. Typical examples are public and secret. The *request ID* field in PDU types 0–3 is used to enable the manager to relate a response to a specific request message. It is selected by the manager and is returned in the related response message. Finally, the *error status* is an integer value that is returned by an agent in a *Get-response* message. For example, a value of 0 indicates no errors, a value of 1 indicates the response is too big to fit into a single SNMP message and a value of 2 there is a nonexistent variable in the list. The latter is then identified by the value in the *error index* field.

SNMPv2

SNMP is continuously evolving and there is now a second version defined in **RFC 1441**. This is directed primarily at internets in which multiple manager stations are involved. The main additions in SNMPv2 are:

- A new message type called *Get-bulk-request*. This has been added to enable the retrieval process of the contents of large tables to be carried out more efficiently.
- A new message type called *Inform-request*. This has been added to enable a manager process in one manager station to send information to a manager process in another manager station.
- An additional MIB for handling the variables associated with manager-to-manager communication.
- The encryption of the password contained in the *community* field.

Summary

In this chapter we have studied the essential features of a selection of the most widely used application protocols of the Internet. These were the application protocols associated with electronic mail (SMTP and MIME), file transfers (FTP and TFTP), and Internet telephony (SIP, SDP, and GLP). In addition, we discussed two system-level application protocols that are essential for the correct functioning of the Internet (DNS and SNMP). The various topics that we have studied relating to these protocols are summarized in Figure 14.24.

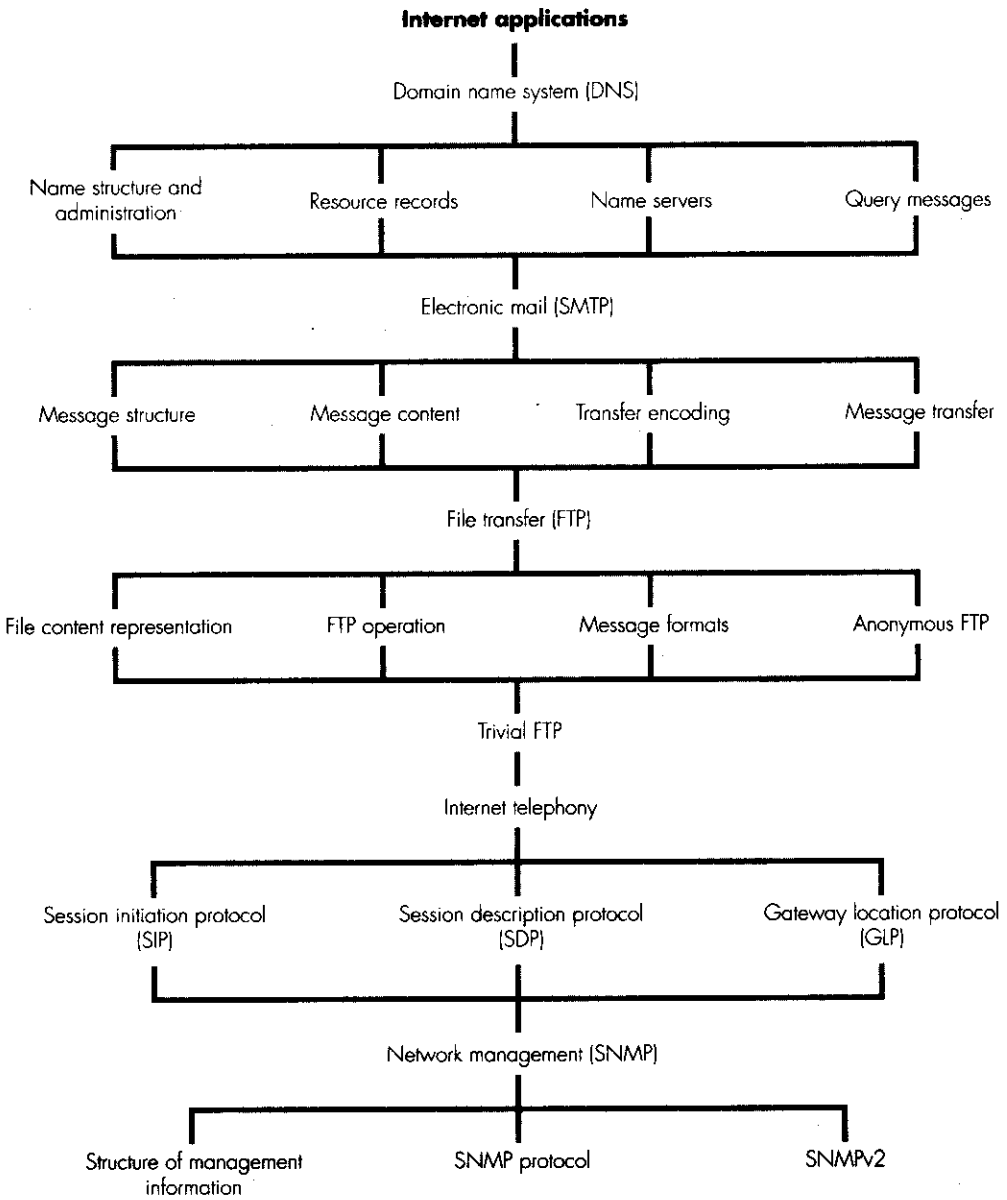


Figure 14.24 Summary of topics relating to Internet applications.

Exercises

Section 14.2

14.1 Explain why a domain name system (DNS) is required with the Internet and describe its main functional parts.

14.2 In relation to the DNS, explain why a hierarchical naming structure is used instead of a flat structure.

14.3 Show the structure adopted for the DNS in the form of a diagram. Include in your diagram:
(i) the root domain,
(ii) a selection of generic domains,
(iii) a selection of country domains,
(iv) some examples of fully qualified domain names.

14.4 Each domain name in the DNS name space/database has a resource record associated with it. With the aid of diagrams, explain:
(i) the format of a resource record,
(ii) the format used for a domain name,
(iii) a selection of the resource record types,
(iv) the use of the time-to-live field.

14.5 Show in outline the format of a DNS query and response message.

Hence give an example of a query relating to a type-A name-to-address resolution query. State the use of the identification and flag fields used in the query header.

14.6 Explain the meaning of the following terms relating to the administration of DNS name servers:
(i) zones,
(ii) primary name server,
(iii) secondary name server,
(iv) authoritative (resource) records.

14.7 With the aid of a diagram, show the sequence of query/response messages that are exchanged to carry out a local name-to-address resolution. Include in your diagram a client host, a local name server, and a server host. Also include the resolver and the set of higher-level protocols that are used in each.

14.8 With the aid of a diagram, show the sequence of query/response messages that are exchanged to carry out a recursive name-to-address resolution. Include in your diagram a root name server, a top-level name server, and a lower-level name server.

14.9 With the aid of a diagram, show the sequence of query/response messages that are exchanged to carry out an iterative name-to-address resolution. Include in your diagram a top-level name server and a lower-level name server.

14.10 Explain the meaning of the term "pointer query" and why pointer queries cannot be resolved using the database structure shown in Figure 14.1. Hence with the aid of a diagram, show the extension to this structure that is used with pointer queries. Give an example showing how an IP address in a query message is resolved into a host name.

Section 14.3

14.11 With the aid of a diagram, explain the function/meaning of the following terms relating to email over the Internet. Include in your diagram two communicating hosts and two related mail servers. Also include the structure of the messages during their transfer over the access networks and the Internet:
(i) user agent,
(ii) message transfer agent,
(iii) POP3,
(iv) message transfer system,
(v) SNMP.

14.12 List a selection of the fields that are present in a message header and the envelope header of an email message. Identify the use of each field in relation to the diagram you used in Exercise 14.11.

Show how user-defined fields are added to the header and give an example of their use.

14.13 Explain the transfer syntax that is used for both the contents of a text-only email message and all the header fields. Include in your description how each field is delimited and how the end of the message header is indicated.

14.14 List a selection of the additional header fields that are used with MIME. Explain the use of each field and how the recipient UA determines the type of contents that are present in the message body and their transfer syntax.

14.15 By means of an example, show how a short message can be sent in both richtext and plaintext. Include the MIME headers that are required and how each form of the message is separated. Is the order of the two message types important?

14.16 Explain the transfer encoding scheme that is used to transfer messages comprising strings of 8-bit bytes. Include in your explanation the use of Base64 and how this relates to NVT ASCII.

14.17 A binary file containing a string of 8-bit audio samples is to be sent in an external file attached to an email message. Assuming the first three bytes in the file are:

10010101 11011100 00111011

use the Base64 table in Table 14.2 and the list of ASCII codewords in Figure 2.6(a) to show how these three bytes are converted and sent in NVT ASCII. Also how the recipient UA determines the original three bytes from the received NVT ASCII string.

14.18 A number of companies have been invited to prepare and submit a tender for a contract using email. To provide a high level of privacy and to authenticate each tender and ensure nonrepudiation, each company has been asked to encrypt their tender using PGP. By means of a block schematic diagram, assuming no compression is to be used, show the various steps that are followed to carry out:

(i) the authentication and nonrepudiation steps and
(ii) the encryption and decryption of the tender contents.

14.19 With the aid of a diagram, explain the function of the various components that make up the application process in a mail server to transfer an email message over the Internet. Include in your diagram the protocol stack that is used in two peer mail servers and also how the UA in each server interacts with the UA in each of the client hosts.

14.20 Using the list of SNMP command and response messages given in Figure 14.11, show a typical message interchange between an MTA client and an MTA server to transfer an email message over the Internet.

Section 14.4

14.21 With the aid of a diagram, describe the role of the control and data transfer parts of the AP in both a client and a server to transfer the contents of a file over a CTP/IP network using the file transfer protocol (FTP). Include in your description how the FTP in the client and the server are involved in the establishment and closing down of a TCP connection.

14.22 With the aid of a time sequence diagram, use the list of FTP command and reply messages given in Table 14.3 to show a typical exchange of messages to carry out the transfer of a named file from the file system on a server to the file system on a client. Explain how the port numbers in both the client and server sides are determined.

14.23 Outline the steps that are taken by a user to log in to a remote server using anonymous FTP. Explain how the server side performs a check on the user before granting the user access.

Section 14.5

14.24 Explain how trivial FTP (TFTP) is different from FTP. Give an example of the use of each protocol.

14.25 List the five message types associated with TFTP. Hence show an example message exchange that illustrates the main features of

the protocol. Include in your example how the sending side detects a lost/corrupted data message and a lost/corrupted acknowledgment message. Also include the use of a LAST ACK timer.

Section 14.6

14.26 Outline the different types of call/session associated with Internet telephony. Hence identify the main requirements of the set of signaling protocols that are associated with it. Namely, the session initiation protocol (SIP), the session description protocol (SDP), and the gateway location protocol (GLP).

14.27 By means of a diagram, identify the protocol stack that is present in each host device that wishes to take part in an Internet telephony call/session. Describe briefly the role of each protocol.

14.28 In relation to the SIP, explain briefly the usage of the following (SIP) message types: INVITE, ACK, REGISTER, OPTIONS, CANCEL, and BYE.

14.29 List a selection of the header fields associated with a SIP INVITE message and state their use. Give an example of a SIP address.

14.30 Outline how a user informs the system that he or she can be contacted at a number of different locations using a SIP REGISTER message.

14.31 With the aid of a diagram, explain how a call/session is set up using SIP between a user and a called user who is currently located at their primary SIP address. Include in your diagram the proxy server at each of the sites involved and the protocol stack that is used in both hosts and proxy servers. Explain clearly the role that is carried out by the proxy servers.

14.32 With the aid of a diagram, explain how the sequence followed in Exercise 14.31 is different when the called user is currently located at a secondary address. Explain clearly the role carried out by the redirect server in setting up the call/session.

14.33 Assuming the session description protocol is being used, explain the use of the following fields that may be present in the body part of a SIP INVITE message:
(i) media streams,
(ii) stream address,
(iii) start and stop times.

14.34 With the aid of a diagram, explain how interworking between a host that is attached to an IP network and a terminal equipment – a telephone for example – that is attached to a PSTN/ISDN is carried out. Include in your diagram a signaling/media gateway and a location server. Hence explain the role of the gateway location protocol (GLP) in relation to the interworking procedure.

Section 14.7

14.35 Describe the role of the simple network management protocol (SNMP) in relation to the Internet. Include in your description the role of fault, performance, and layer management and the meaning of the term "managed object".

14.36 By means of a diagram, show the protocol stack associated with a host and a router that enables them to be managed from a remote management station attached to the Internet. Include in your diagram a management agent and a manager process and explain how the two interact.

14.37 Explain the role of the management information base (MIB). Also explain why ASN.1 is used to define the structure of all the management information that it contains.

14.38 All managed objects within the Internet are identified within the context of an internationally defined object naming tree. By means of a diagram, show the structure of this tree down to a level that includes the various managed objects in a particular item of equipment. Give an example of the identifier of one of the managed objects in the tree.

14.39 Using the object naming tree you derived as part of Exercise 14.37 as a template, explain how a specific managed object within a particular item of equipment is (uniquely) identified.

14.40 By means of an example, show how a managed object/variable relating to the IP protocol is defined. Include in your definition an OBJECT TYPE, SYNTAX, STATUS, and DESCRIPTION parameter. State the role of each parameter and how the variable is placed in the object name tree of the device in the MIB.

14.41 By means of examples, show how each managed object/variable in the MIB is identified uniquely. Use for example purposes a variable with a single value and also one comprising a table of values.

14.42 With the aid of a diagram, list the five message/PDU types associated with SNMPv1. Include in your diagram the protocol stack and the well-known port numbers that are used.

Explain briefly the role of each of the five message types and how a lost message can be detected.

14.43 With the aid of the SNMP message/PDU formats defined in Figure 14.23(b), explain how a manager process obtains the current value of a named variable – managed object – from the agent process in a specified item of (managed) equipment. Include in your explanation the use of the Community, RequestID, Error Status, and Error Index fields in the SNMP messages/PDUs.

14.44 Outline the extensions to SNMPv1 that are present in SNMPv2. Hence explain how the security features of SNMPv1 have been enhanced.