Internet applications

8.1 Introduction

As we showed in Figure 7.1 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 that, 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.

8.2 Domain name system

As we saw in Figure 7.1 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.

8.2.1 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 e-mail system – then the majority of service requests can be resolved locally and relatively few referred to another site.

As we show in Figure 8.1, 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,

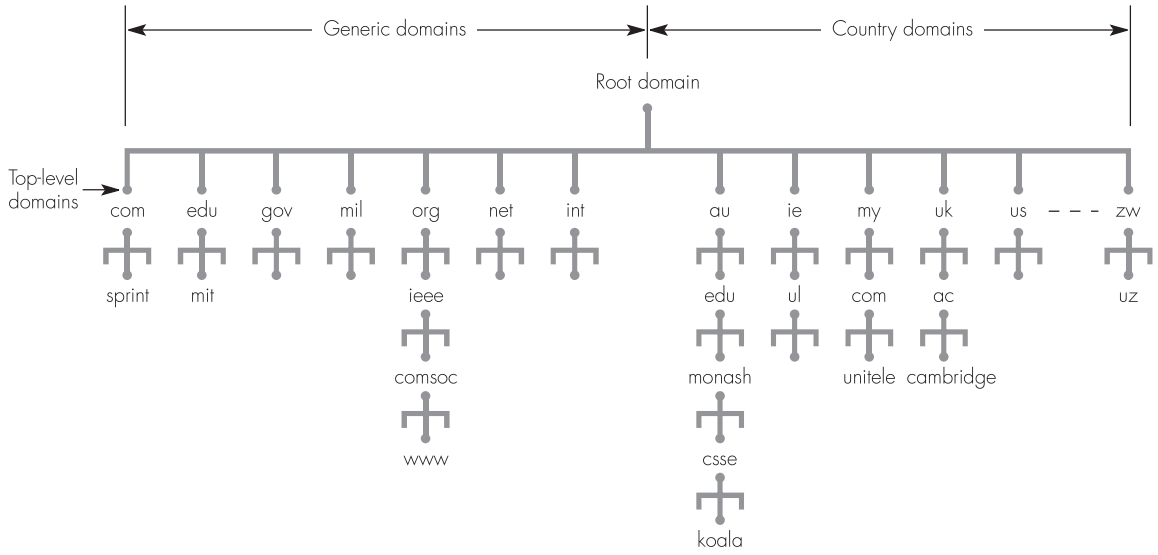


Figure 8.1 The structure of the domain name system together with some examples.

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 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 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:

sprint.com.
cambridge.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.

8.2.2 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 8.2(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 8.2(b). Each label is preceded by a one-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 8.2(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 e-mail gateway, for example – that is prepared to accept e-mail 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.

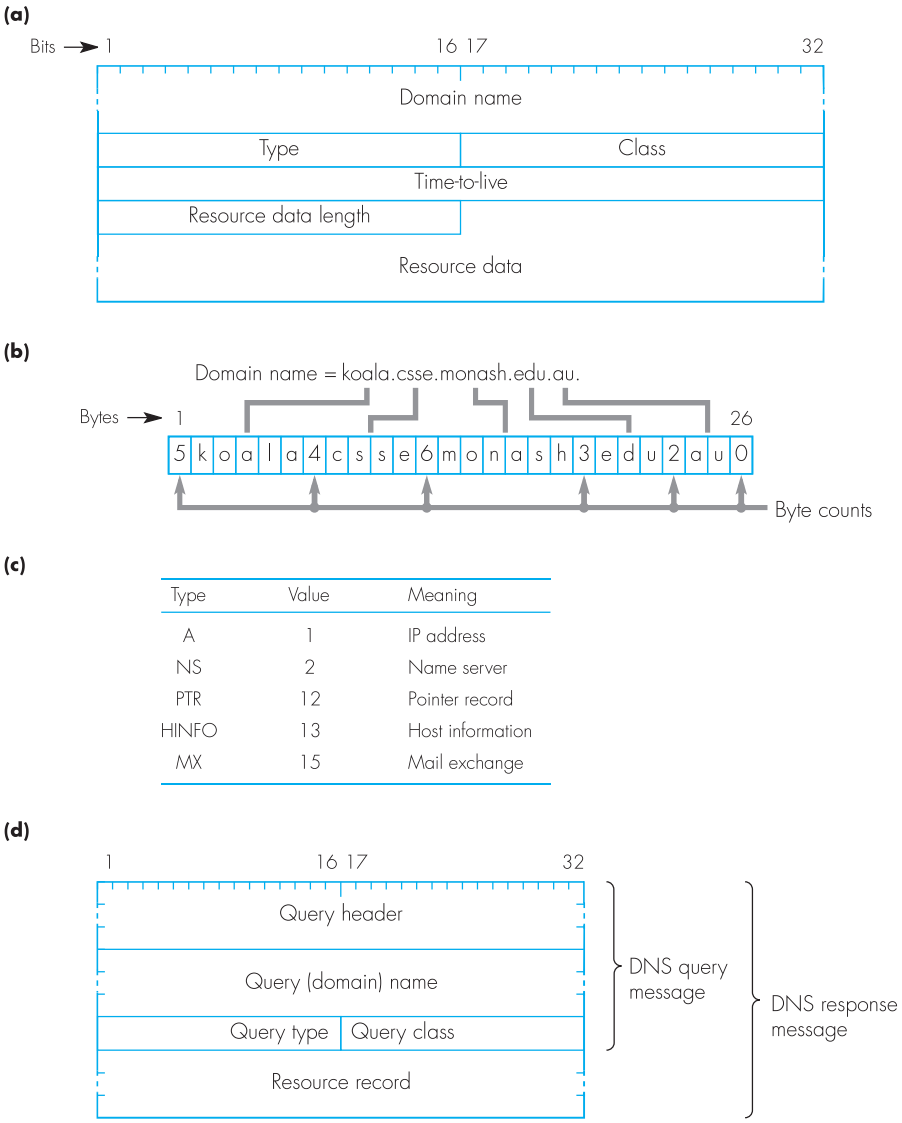


Figure 8.2 DNS resource records and queries: (a) resource record format; (b) domain name format; (c) a selection of resource record types; (d) query and response message formats.

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.

8.2.3 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 8.2(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 8.2(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.

8.2.4 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 therefore 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 8.3. 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.

8.2.5 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

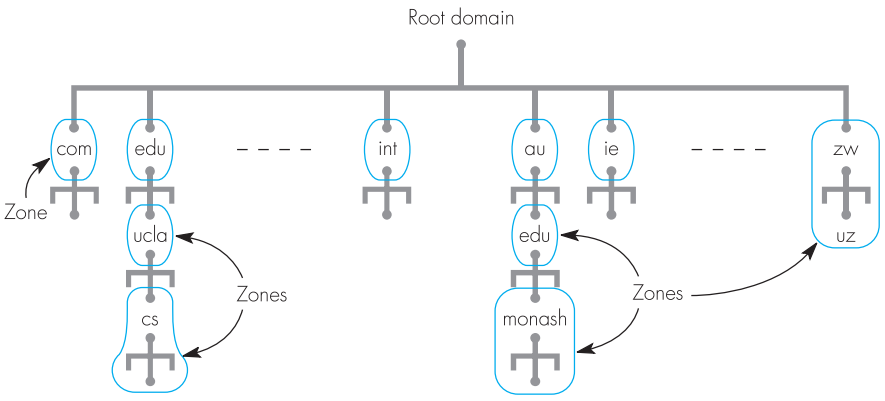


Figure 8.3 Some examples of DNS zones.

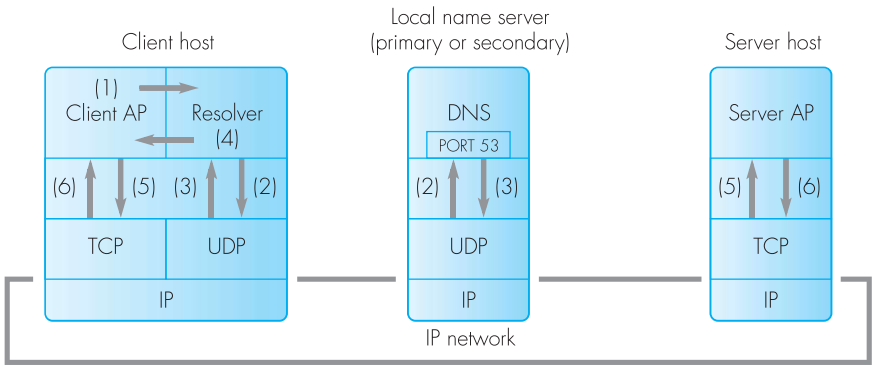
called a **recursive name resolution**. Alternatively, in order to reduce the amount of processing done by each name server, an iterative approach can be used. Before we describe each of these approaches, however, we shall describe first how a simple name resolution is carried out using a name server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

Local name resolution

As we show in Figure 8.4, 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, e-mail 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

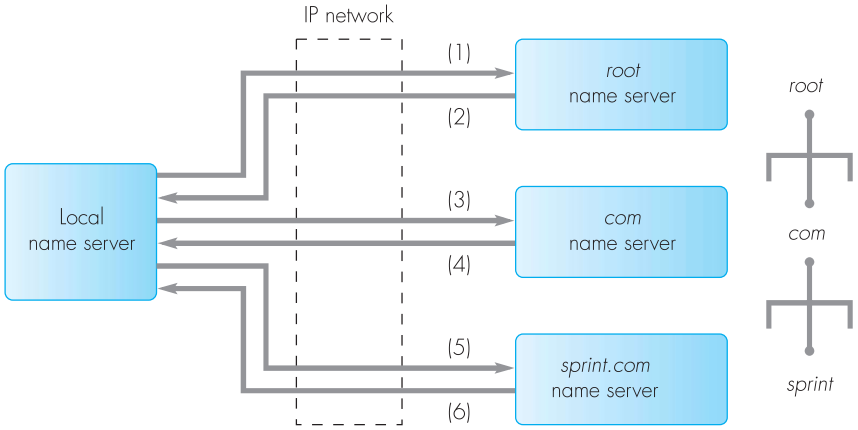
Figure 8.4 Example showing the sequence of messages exchanged for a local name resolution.

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 8.5.

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 *sprint.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



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

Figure 8.5 Example showing the sequence of messages exchanged for a recursive name resolution.

server using the returned IP address (3). In response, the *com* name server determines from the name in the query that the *sprint.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 *sprint.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.

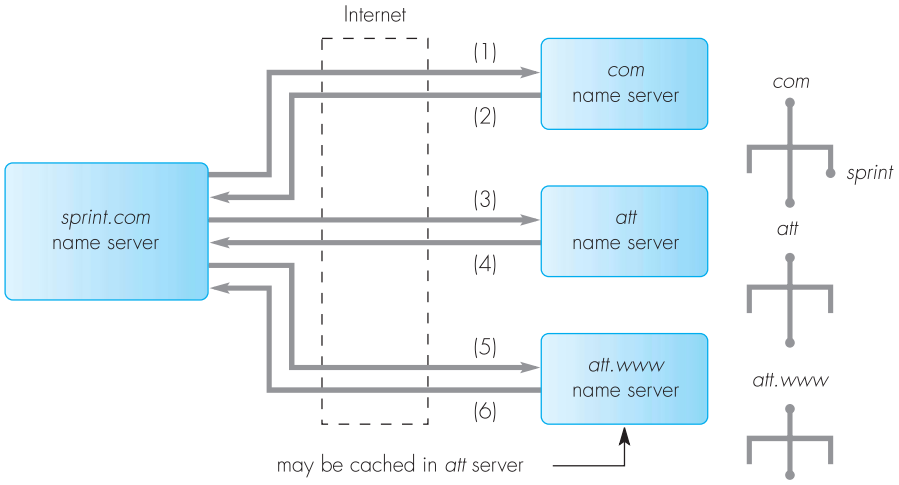
Iterative name resolution

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 8.6 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 e-mail 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 8.6 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 8.3, 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 8.1 is present. This is shown in Figure 8.7.

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

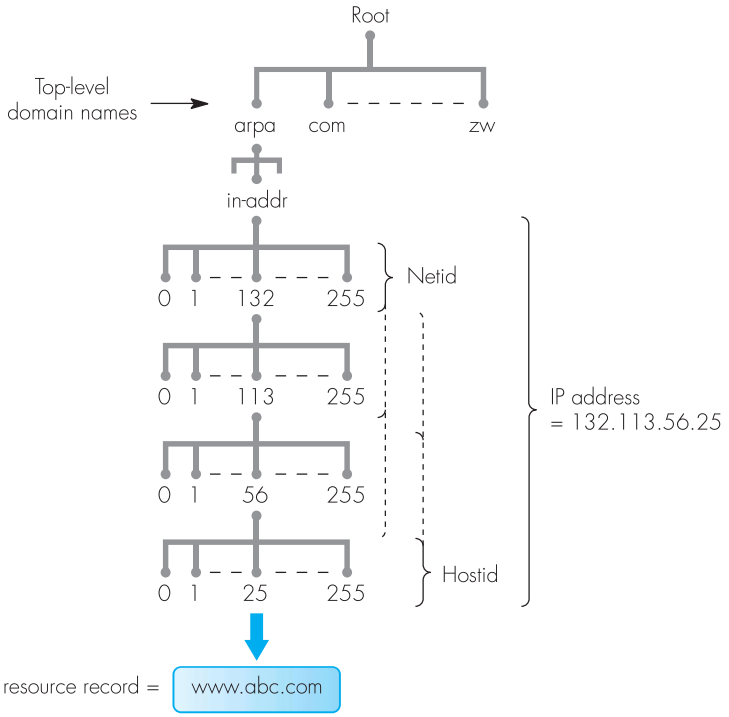


Figure 8.7 Pointer query principles.

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**.

8.3 Electronic mail

From a user perspective, electronic mail (e-mail) – apart from Web browsing which we describe in the next chapter – is probably the most popular application on the Internet. As we show in Figure 8.8, an e-mail system comprises two main components: an e-mail client and an e-mail server. Normally, an e-mail client is a desktop PC or workstation running a program called the **user agent** (**UA**). This provides the user interface to the e-mail system and provides facilities to create, send, and receive (e-mail) messages.

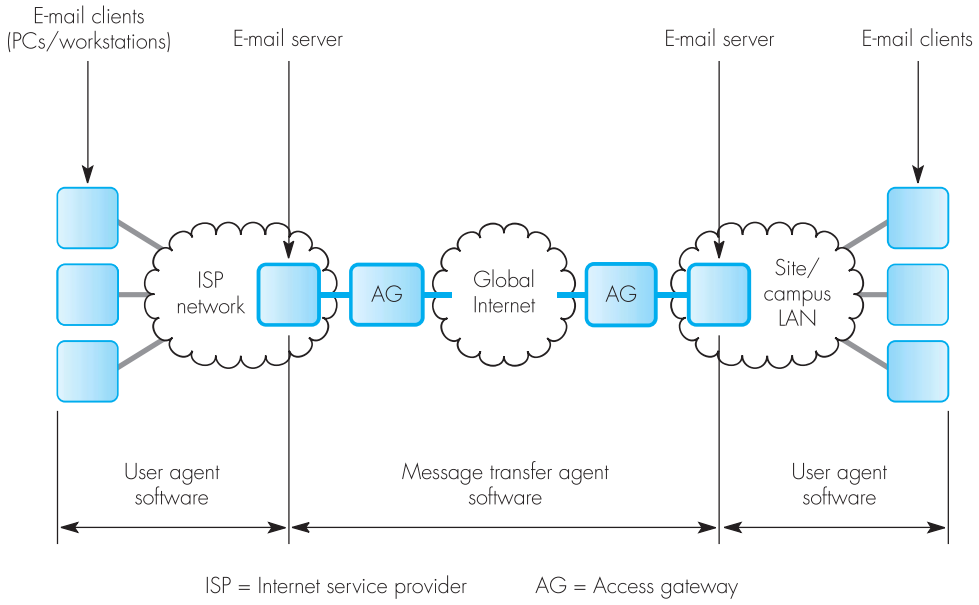


Figure 8.8 E-mail over the Internet.

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 e-mail server is a server computer that maintains an IN mailbox for all the users/clients that are registered with it. The set of mailboxes in the e-mail server are contained in a database known as the **message store**. 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 e-mail servers that are also connected directly to the Internet.

The protocol stack that is used to support e-mail over the Internet is shown in Figure 8.9. 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 2.6.4 – 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

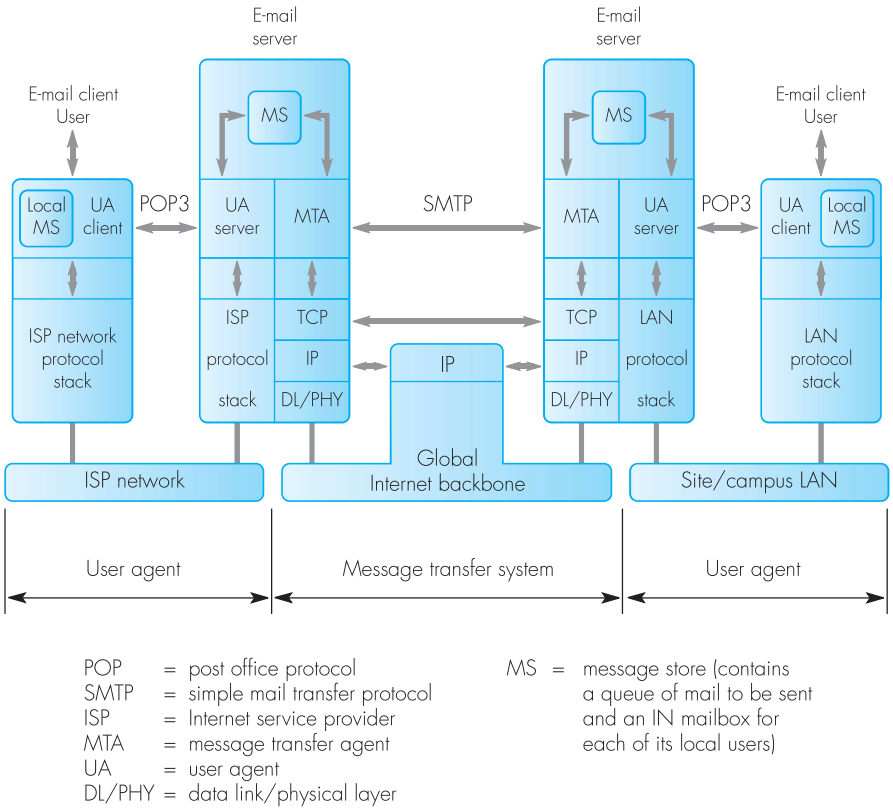


Figure 8.9 Protocol stack to support e-mail over the Internet.

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.

8.3.1 Structure of e-mail 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.

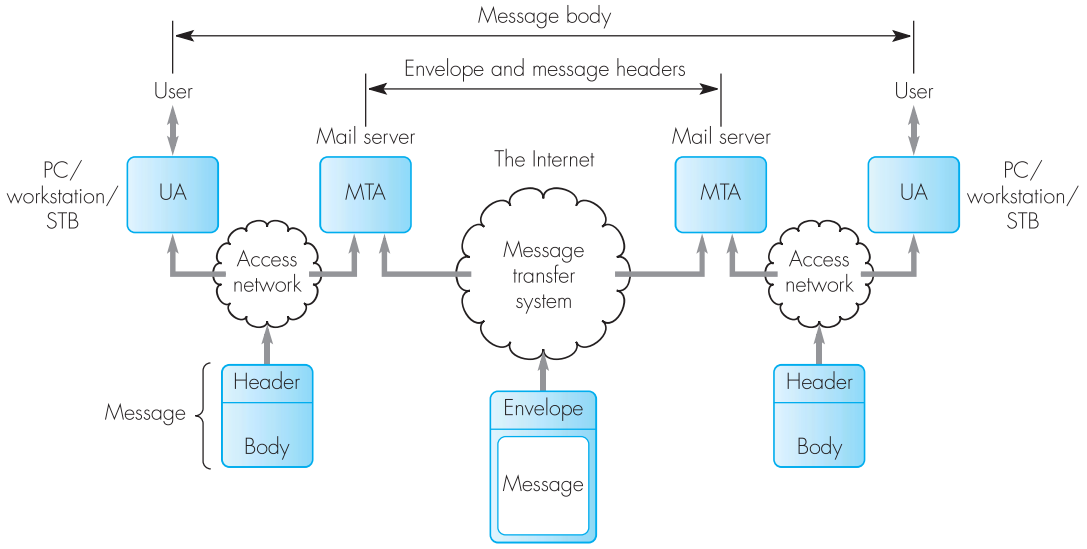
In a similar way, the sending of electronic mail involves two separate procedures. The first is concerned with the entry of various fields – including the sender's and recipient's name/address at the head of the message – and the actual message content via the UA; the second with the encapsulation of 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 e-mail message showing these two parts is shown in Figure 8.10(a).

As we can see, during its transfer across the network an e-mail message is composed of an *envelope* and the *message*. The envelope contains the e-mail address of the sender of the message (*MAIL FROM*) and its intended recipient (*RCPT TO*). In the case of the Internet, all e-mail 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 e-mail 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 8.10(b), the header comprises a number of fields some of which are optional. Also, since there are many different vendors of UA

(a)



(b)

Used also by the message transfer system	From:	E-mail address of person who created the message
	To:	E-mail address of primary recipient
	Cc:	List of e-mail addresses of other (secondary) recipients
	Received:	Route followed through message transfer system
	Return-Path:	Name of last MTA
Used by UA/user	Sender:	E-mail 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:	E-mail 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 8.10 E-mail message structure: (a) terminology and usage; (b) selection of the fields in the message header and their use.

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 that 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 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 e-mail 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 e-mail 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 1.14 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**.

8.3.2 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 that 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 that can then be transferred using NVT ASCII.

MIME headers

The additional MIME header fields and their meaning are listed in part (a) of Table 8.1. The first field following the standard header fields is the *MIME-Version:* which, when present, informs the recipient UA that an alternative

Table 8.1 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

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 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 show in Figure 9.10. Similarly, digitized images, audio, and video are all represented in their compressed form. As we explain in Appendix A, 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 and JPEG. The subtype selected for audio is a form of PCM and that for video a version of MPEG. In the case of a movie (video with sound), the MPEG subtype is used with the audio and video integrated together.

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, and 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 8.11. 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 8.1 is the *Content-Transfer-Encoding:* field and an example of its use was shown in Figure 8.11. As the name implies, it is concerned with the format of the

```
From: xyz@abc.com
To: abc@xyz.com
Subject: Happy birthday Irene
MIME-Version: 1.0
Content-Type: Multipart/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 8.11 MIME: example type and subtype declarations.

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 that 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 8.2. 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 8.2 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 8.1

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 1.14, 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	00111101 (=)	00111101 (=)

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

Value 1 =	10 0101	01 1101	11 0000	11 1011
	2 5	1 D	3 0	3 B
Value 2 =	01 0110	00 0011	11 0100	11 1101
	1 6	0 3	3 4	3 D

Using Table 8.2:

Value 1 represented as the four ASCII characters: *ldw*7
Value 2 represented as the four ASCII characters: *WD*09

Hence from codewords in Figure 1.14, contents in NVT ASCII are:

Value 1 =	0 110 1100	0 110 0100	0 111 0111	0 011 0111
Value 2 =	0 101 0111	0 100 0100	0 011 0000	0 011 1001

(ii) Received bytestream first converted back into the equivalent ASCII string using codewords in Figure 1.14:

Value 1 = *ldw*7
Value 2 = *WD*09

The equivalent four 6-bit subvalues then obtained from Table 8.2:

Value 1 =	2 5	1 D	3 0	3 B
	= 10 0101	01 1101	11 0000	11 1011
Value 2 =	1 6	0 3	3 4	3 D
	= 01 0110	00 0011	11 0100	11 1101

These are then combined and converted into two 24-bit values:

Value 1 =	10010101	11011100	00111011
Value 2 =	01011000	00111101	00111101

Last two 8-bit values of the last (second) 24-bit value are the NVT ASCII of the = character and hence are discarded.

File contents restored as:

10010101 11011100 00111011 01011000

8.3.3 Message transfer

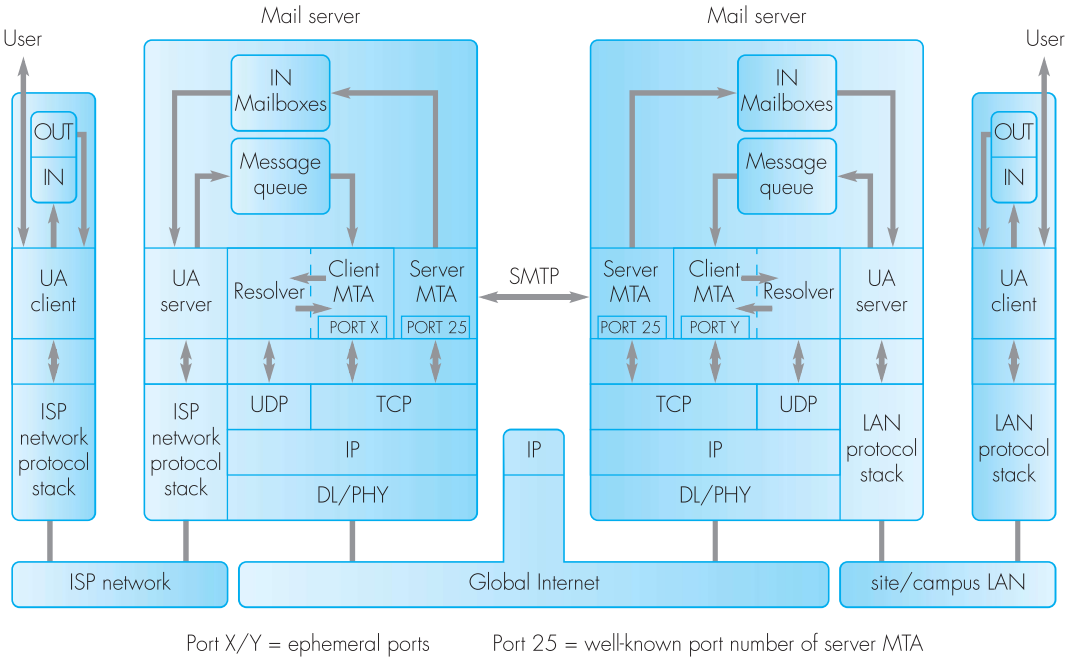
The main components that make up the message transfer system are shown in Figure 8.12(a). Once a user has created a mail message and clicked on the SEND button, the UA first formats the message into NVT ASCII and then sends it to the UA server in its local mail server using the protocol stack of the 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 e-mail 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.

As we saw earlier, all e-mail 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 8.2.5, 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)

(a)



(b)

Commands (Client MTA → Server MTA)	Descriptions
HELO Mailserver-name	Sends DNS name of the client mail server
MAIL FROM: <e-mail address of sender>	e-mail address of sender
RCPT TO: <e-mail address of recipient>	e-mail 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	Error responses
450	Mailbox unavailable	
⋮	⋮	
551	Addressed user is not here	

Figure 8.12 SMTP: (a) components; (b) command messages; (c) response messages.

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 8.12(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 8.12(c).

A typical exchange sequence of SMTP control messages to send a mail message is shown in Figure 8.13. 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 e-mail address and, if this is accepted, it sends the intended recipient's e-mail 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.

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 letter then informs the UA client in the sending terminal and this, in turn, informs the user that the message has been sent.

8.3.4 E-mail gateways

The structure we showed earlier in Figure 8.9 assumed that both e-mail servers were connected to the Internet and hence could communicate directly using the SMTP/TCP/IP protocol stack. With many companies and

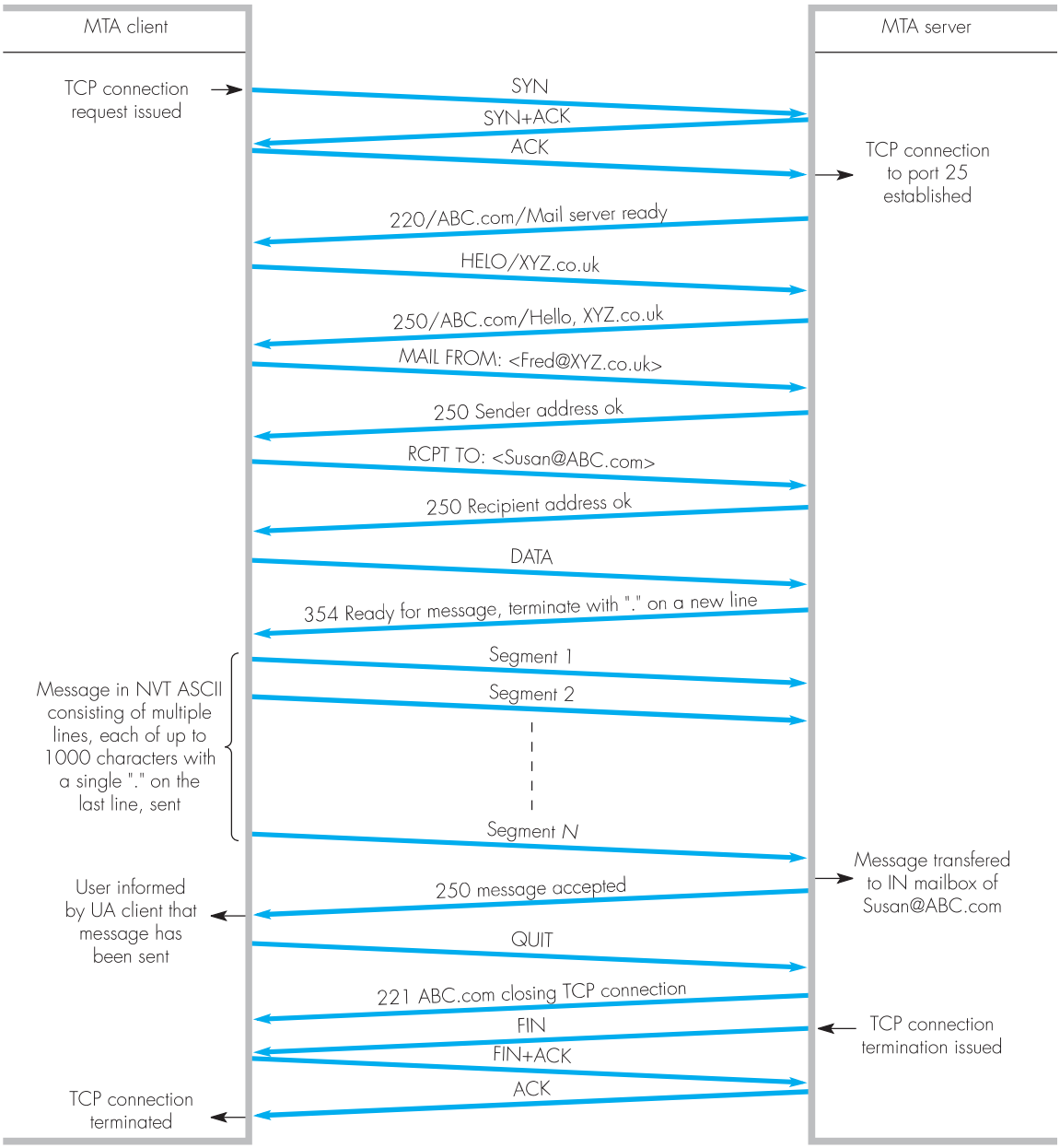


Figure 8.13 Example e-mail message transfer from Fred@XYZ.co.uk to Susan@ABC.com using SMTP.

large enterprises, however, often this is not the case and instead a different e-mail system is used. Nevertheless, in addition to sending and receiving mail to/from other employees within the company/enterprise, many of these employees may also require to send and receive mail to/from people whose computers/PCs are connected either to a different company/enterprise network or to the Internet. In practice, there are two problems associated with doing this: first the format of the mail messages is often different and second the application protocols are also different. To overcome these problems, a device known as an **e-mail gateway** is used, the general arrangement being as shown in Figure 8.14.

As we show in the figure, the gateway has a number of interfaces, one for connecting to the local e-mail server at the site and the others for connecting to those networks (including the Internet) with which the employees at the site wish to communicate. To transfer a message that is addressed to an outside network, the e-mail server first transfers the message to a message buffer in the e-mail gateway using the protocol stack associated with the company/enterprise network. The e-mail address in the header of the message is then read by an application-level program to determine the network over which the mail should be forwarded. Assuming the external network is the Internet, the program proceeds to reformat the message into the RFC 822 format and then forwards this using the TCP/IP protocol stack as described previously. A similar procedure is followed in the reverse direction except the message format has to be changed from RFC 822 to the format used by the company network.

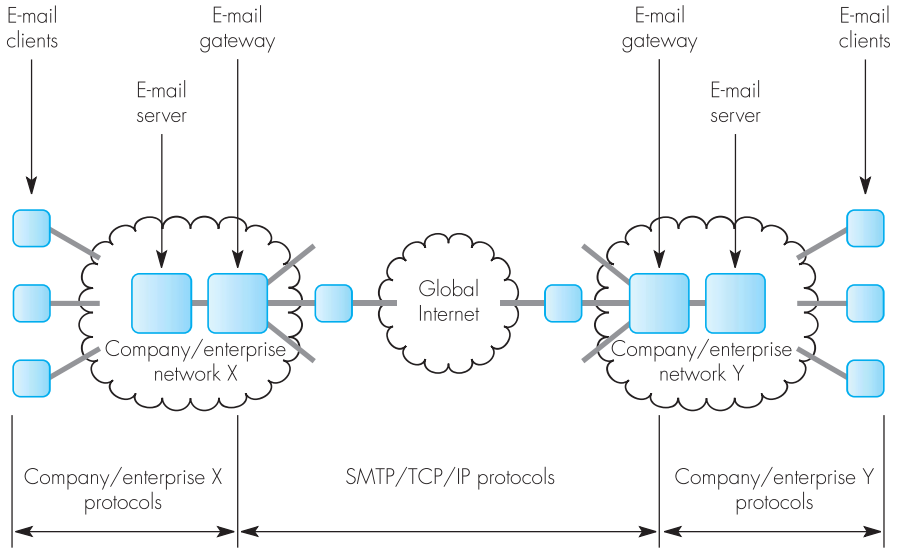


Figure 8.14 E-mail across dissimilar networks using an e-mail gateway.