*:96 Internet application layer protocols and standards

Compendium 4A: Basics, E-mail & NNTP

Not allowed during the exam

Last revision: 11 Jan 2005

E-mail

PGP

NNTP

The documents are not ordered in a suitable order for reading them,

see compendium 6 page 909-911

Excerpt from:

Internet application layer protocols

By Jacob Palme

Compendium 4 page 2

Message Handling

Table of contents

Introduction and Basic Concepts

1.1 A Simple Example

Internet application layer protocols were from the beginning very simple. A computer program on one computer could connect to a program running on another computer, and send simple textual commands. Figure 1 shows the interactions needed in a simple case to send an e-mail message.

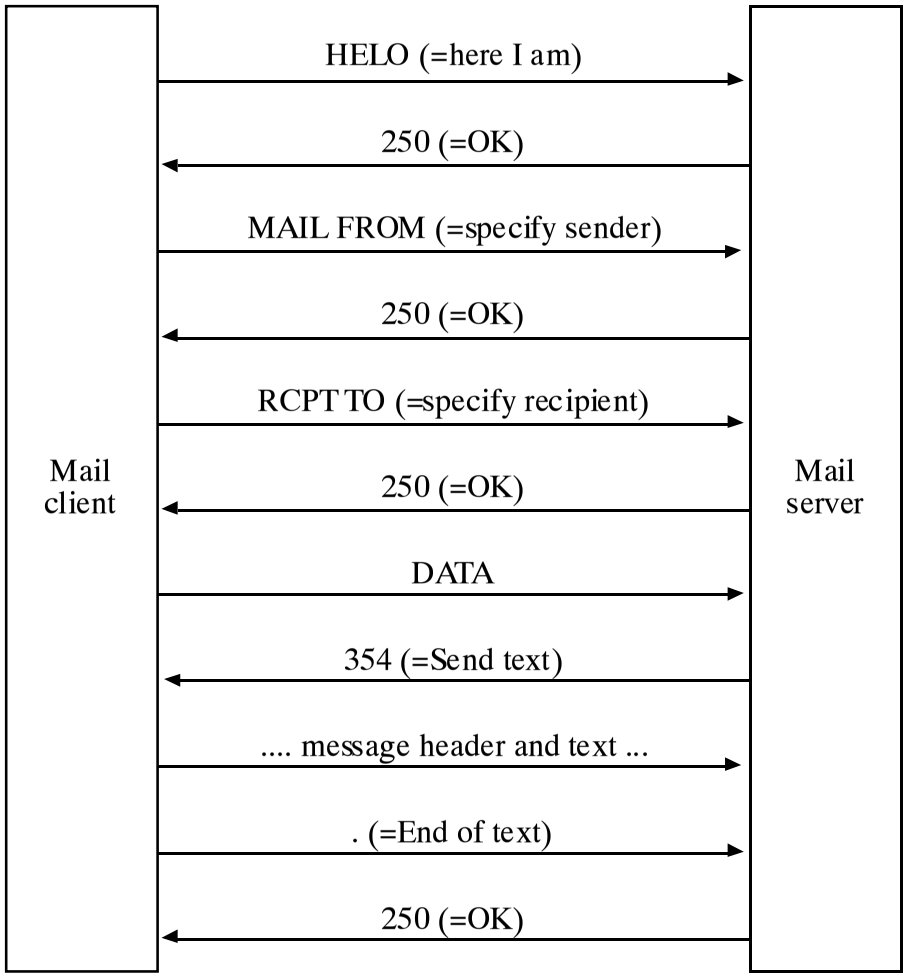


Figure 1: Interactions for sending an e-mail message

The following is all that is necessary to send e-mail from one host to another ("C:" is text sent by the client, "S:" is text returned by the server):

```
C: <connects to server>
S: <accepts connection>
C: HELO mail.duckland.com
S: 250 mail.northpole.com
```

```
C: MAIL FROM: donald-duck@duckland.com
S: 250 donald-duck@duckland.com... Sender ok
C: RCPT TO: father-christmas@northpole.com
S: 250 father-christmas@northpole.com... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: From: Donald Duck <donald-duck@duckland.com>
C: To: Father Christmas <father-christmas@northpole.com>
C: Date: 24 Dec 1999
C:
C: Peace be with you.
C: .
S: 250 PAA00329 Message accepted for delivery
C: QUIT
S: 221 mail.northpole.com closing connection
S: <closes connection>
```

So, just by sending a few lines of text back and forwards between two computers, an e-mail message is transmitted.

1.2 Process, Client, Host, Server

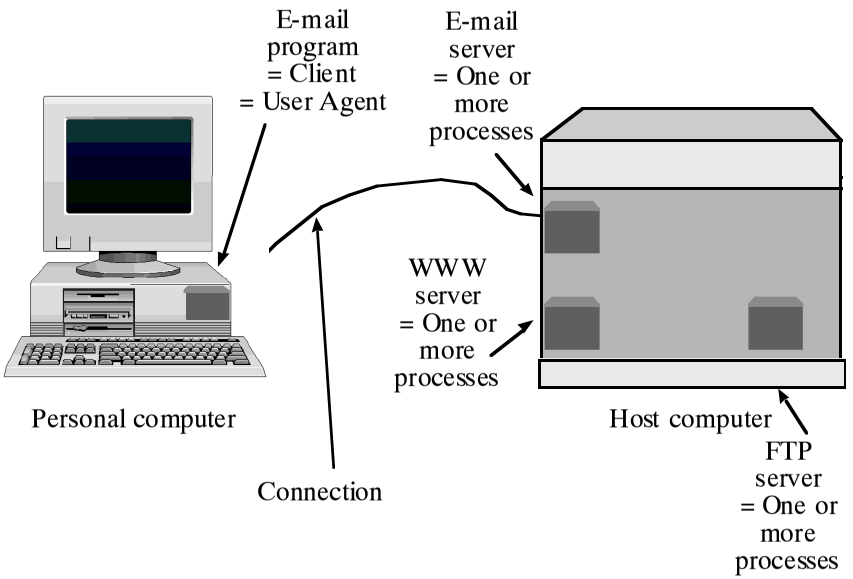


Figure 2: Process, Client, Host, Server, User Agent

Here are some important kinds of objects in network protocols (see also Figure 2):

Process: A program running on a computer. Most computers allow several processes to run at the same time.

Agent: A process connected to the network. The term “agent” is also sometimes used for a process which acts without immediate user control, and which may represent itself as a user to the network.

Client: A process which can establish connections to servers and send requests to them.

User agent: A client which represents a user. User agents often have a user interface, so that human users can directly control them.

Server: A process which accepts connections from clients and performs services for them. A server can often handle several simultaneous connections from different clients. Technically, this can be done by running a single process, or by running a separate process for each client.

Host: A computer connected to the network and providing services. Many different services can be provided by different servers on the same host.

Note that the same process can be a server for one agent, and a client for another agent. Figure 3 shows three agents, one user agent and two service agents. The middle service agent acts as a server to the user agent, and as a client to the other user agent.

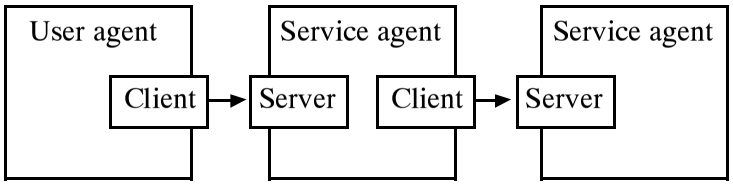


Figure 3: The middle agent in this figure acts as a server to a user agent and as a client to another service agent.

1.3 Protocols

The word *protocol* comes from the language of diplomacy. By protocol is meant the rules for the language used in communication between countries. Such rules are intended to ensure that countries correctly understand each other, to reduce the risk of misunderstanding. And the intention with network protocols is the same. The protocols specify clearly what can be said, and how it is to be said. Like programming languages, the words and phrases in some network protocols can have some similarity to natural language. But like programming language, the artificial languages are simpler, more restricted, more exact, and do not allow ambiguities. The similarity to natural languages is a way of making the protocols easier to understand for humans.

1.4 Layering Model

Layering is easiest to understand if you start with the case where only one computer and no network is involved. Even in that case, most programs are structured into layers. Suppose you are designing a program for designing flowcharts. Such a program may have the structure shown in Figure 4:

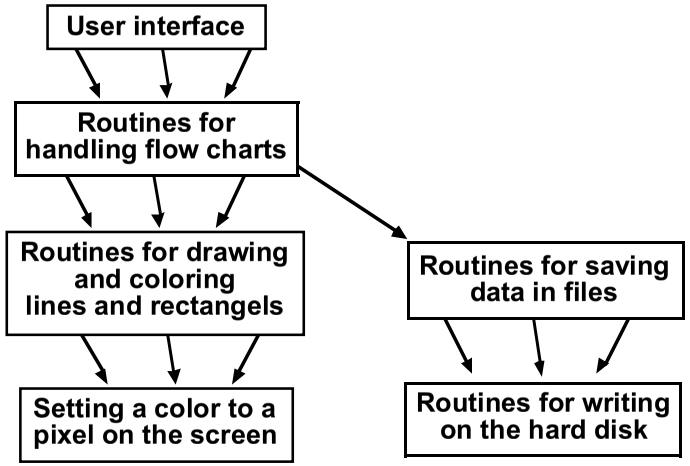


Figure 4: Structured programming: Routines in higher layers use features of lower layers.

Figure 4 shows an example of the structure of a program running on a single computer. The higher layers use features of the lower layers. Note that it is only the lowest layer which actually writes information on the screen or on the hard disk. The higher layers influence the contents of the screen and the hard disk only indirectly by using routines in layers below them. The layers piled on top of each other are often referred to as a "stack".

Network layering works in the same way. But in networks, there are more than one program on more than one host computer. So there is a need for a separate "stack" of layers on each computer. In the simple case where only two processes running on two computers are involved, this can be depicted as in Figure 5.

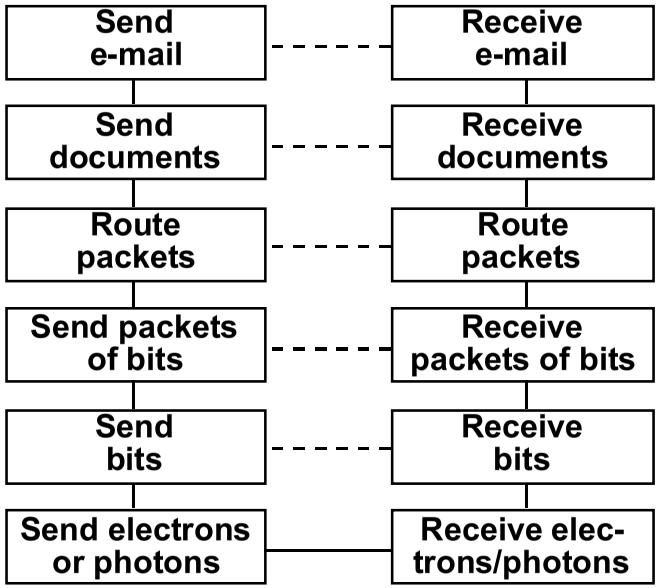


Figure 5: Two stacks of layers, one on each host, communicating with each other.

Why are the horizontal lines dashed between all the boxes except in the lowest layer? Think of the example in Figure 5. There, only the lowest layer physically writes on the screen or the hard disk. All the higher layers will indirectly write on the screen or the hard disk through the lowers below them. The principle for networked protocols are the same. The layer "Send bits" does not physically send bits to the layer "Receive bits". The layer "Send bits" asks the lower layer "Send electrons or photons" to send the bits, which are then forwarded up to the box "Receive bits". Since the communication between "Send bits" and "Receive bits" is indirect and not direct, it is shown as a dashed line.

There is another way to view layeriing, shown in Figure 6.

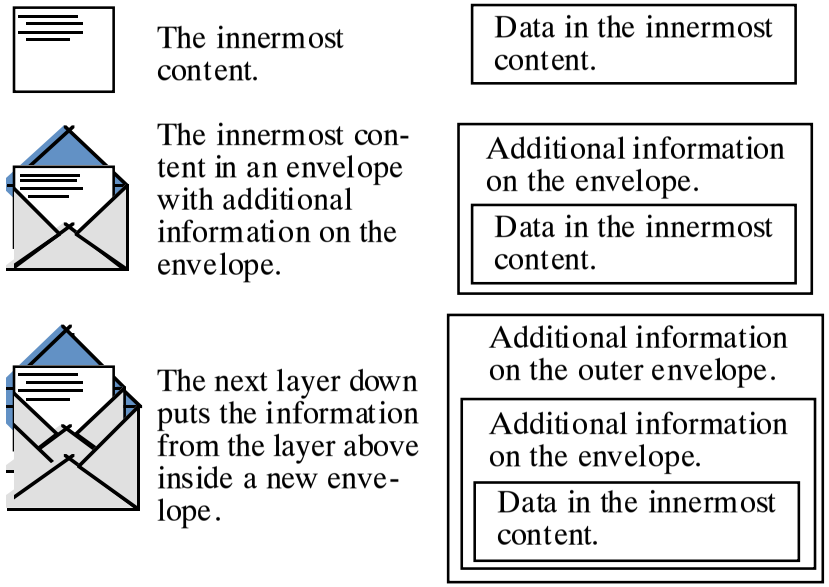


Figure 6: Layering seen as envelopes inside envelopes inside envelopes.

The important understandings of this view are two:

1. Each lower layer usually adds information. The amount of information transported increases with each added lower layer.
2. The code at each layer only looks at the information added at this layer. The information in higher layers is just regarded as a set of bits by the lower layer, it does not understant its inner content or structure.

For example, the e-mail transport system (SMTP) will transport an e-mail message from its sender to its recipient. This is done using the information on the e-mail envelope. The contents inside the envelope are just moved along, but not touched. And the e-mail transport system uses another layer below it (TCP), which moves a document from one host to another host. TCP has its own envelope, and the contents (the e-mail envelope and content) is just moved along, not touched.

This is not always quite true. For example, e-mail transport systems will sometimes convert the content of the message from one encoding to another encoding. This is regarded as a "layering violation" and is not regarded as quite neat, but is sometimes necessary.

The major layers in computer networking are listed in Table 1.

Application layer	The actual application, such as e-mail or the WWW.
Session layer	Keep an exchange of information going through interactions back and forward between the two hosts.
Transport layer	Transport a whole document from sender to recipient.
IP layer	Transport small packets of bits through the network.
Physical layer	A current or light transports a stream of bits.

Table 1: The major layers in computer networking.

1.4.1 Layers below the application layer

This book is only about the application layer. The layers below it are described in other books. They will only be mentioned as tools used by the application layer. On the Internet, most communication uses a standard called TCP/IP for the lower layers. TCP/IP provides a reliable way of

moving a document from one port on one host to another port on another host. Some Internet applications use an alternative to TCP, called UDB. The difference between TCP and UDB is that TCP ensures that all the packets, into which a document has been split, are reliably transported and collected together in the right order. If a packet is lost, TCP will ensure that it is resent. UDB just disregards lost packets. This makes UDB faster and suitable for same-time protocols like simultaneous transmission of voice and video. For such voice and video, it is more important to get a continuous stream at the right speed. Lost packets are accepted as noise. Most other applications, like e-mail, FTP (file transfer) and normal WWW communication use TCP, where reliability is more important than speed and same-time continuity.

1.4.2 Layering within the application layer

According to a fundamentalist view of networking, networks only have major layers of the kind shown in Table 1. But in reality, there are often layers within the application layer. The processing of data within an application often is structured into a series of operations performed when sending data, and the reverse series when receiving data. Many standards implicitly or explicitly specifies this, as is shown in Figure 7. A simple example of this is the e-mail standards. Figure 8 shows how the e-

mail standards are structured into a “Header and body” layer and a “Mail forwarding layer”.

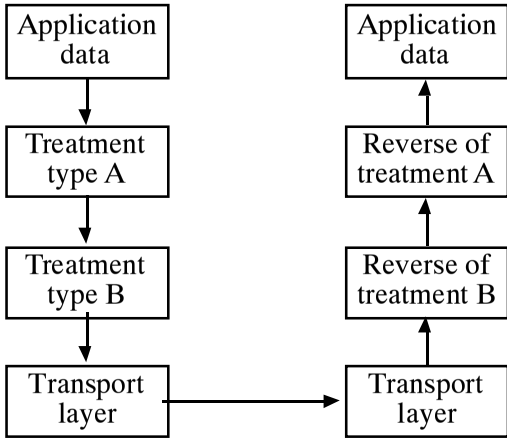


Figure 7: Layering within the application layer (layers below the transport layer omitted)

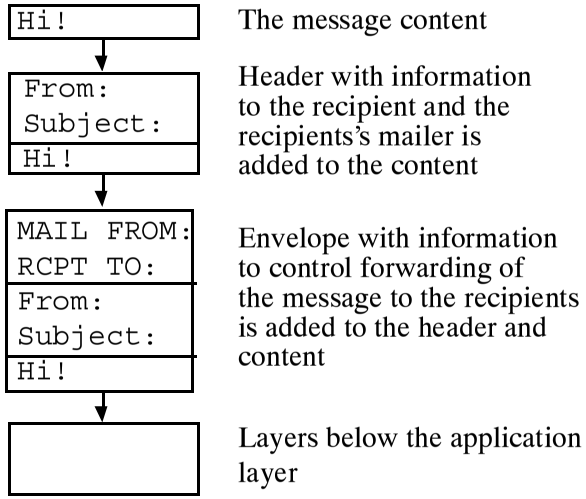


Figure 8: Layering within the application layer in e-mail standards

1.5 Ports and Applications

There are many different application layer protocols. The protocol used when sending e-mail is different from the protocol used when downloading web pages. An e-mail client is not able to communicate with a WWW server, because they speak different languages. Only a client and a server which speak the same language can communicate. But the same host can have several different client and server applications. A personal computer can, for example, at the same time run both mail programs and web browser. And the same host can at the same time run mail servers and WWW servers.

It is very important that messages sent by an e-mail client are sent to an e-mail server, and that messages sent by a web browser are sent to a WWW server. To ensure this, a host computer has different ports, as is seen in Figure 9. The transport layer establishes hoses between applications. And the hoses fit into different ports for different appliations. This is done by assigning a port number to each port. When a client connects to a server, it specifies which port number it wants to access. There are standard port numbers for each application protocol. Thus, when a web browser connects to a web server, it usually ask for port number 80, because this is the standard port number for web servers. If there is more than one web server on the same host, they may use different port numbers. In that case, the client can

use the port number to indicate which server it wants to connect to.

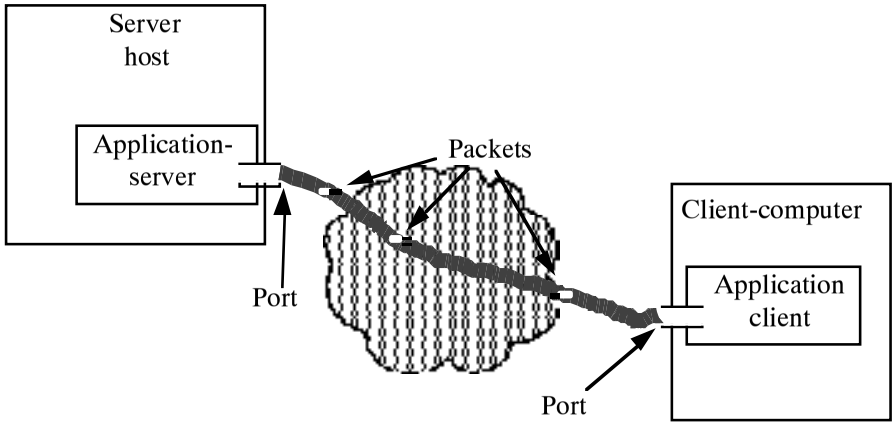


Figure 9: A TCP connection between two computers can be seen as a hose, through which packets of octets can be sent between ports. The port are openings to applications running on the computers.

There is a long list of such standard port numbers maintained by the standards organisation IANA (IANA may soon change to become a part of ICANN). IANA means Internet Assigned Numbers Authority, and its main task is to distribute numbers and names which are allocated for particular uses. Some common of these standard port numbers are listed in Table 2.

Table 2: Some common port numbers

20	ftp-data	File transfer, data
21	ftp	File transfer, control
23	telnet	Terminal emulator
25	smtp	E-mail forwarding
53	dns	Domain name lookup
70	gopher	Gopher search

79	finger	Finding info about a user
80	http	Retrieving web pages
109	pop2	Delivering e-mail to its final recipient
110	pop3	Delivering e-mail to its final recipient
119	nntp	Usenet News
143	imap2	Delivering e-mail to its final recipient
194	irc	Distributed chat system
220	imap3	Delivering e-mail to its final recipient
993	simamp4	Delivering e-mail to its final recient through a secure channel
995	spop3	Delivering e-mail to its final recient through a secure channel

A full list can be found at the IANA web site [1].

1.6 Telnet: A Simple Application

In the 1970s and 1980s, most computer user interfaces were based on sending one or more single characters to the host, and getting one or more single characters back. Many applications on unix computers still use such interfaces. Telnet is an Internet protocol for using such an interface on another computer than your own. A simple example of a Telnet usage is shown in Figure 10. Telnet is of special interest, because it works the same way most application layer protocols work: Sequences of characters are sent back and forward between two computers. Because of this, telnet clients can sometimes be used to connect to other applications than those designed to use telnet.

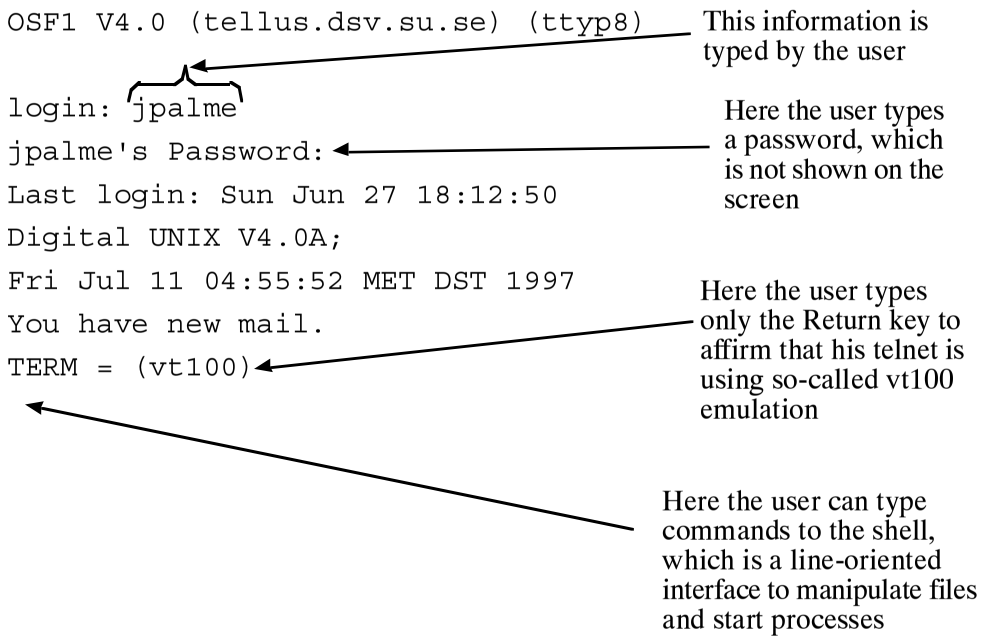


Figure 10: Example of a simple telnet usage. The host and the user take turns sending characters and new line codes to each other.

Here is an experiment which will make it easier to understand how network protocols work. All you need to perform this experiment is a telnet client on a computer connected to the Internet. The telnet client must have the capability to connect to other ports than the regular telnet port 23.

1.6.1 Manual test of e-mail protocols

Use the telnet client to connect to port 25 of the computer running the mail server you are using. Everything you type is underlined in the text

below. Text from the computer is not underlined. The character "¶" means that you should push the Return key. Replace "mail.foo.bar" with the domain name of the mail server you are using, and replace "mycomputer.foo.bar" with the domain name of your own computer.

The first line below is the unix command to start telnet and connect it to port 25 of the server mail.foo.bar. If you are running telnet from another computer than a unix computer, you replace the first line below with the command to get your telnet program to do this connection.

```
unix>telnet mail.foo.bar 25
Connected to mail.foo.bar.
Escape character is '^]'.
220 info.dsv.su.se ESMTP Sendmail
8.8.5/8.8.5; Fri, 24 Dec 1999
15:24:09 +0200 (MET DST)
helo¶
501 helo requires domain address
helo mycomputer.foo.bar¶
250 info.dsv.su.se Hello
tellus.dsv.su.se 130.237.161.217],
pleased to meet you
MAIL FROM: donald-duck@foo.bar¶
250 donald-duck@foo.bar... Sender
ok
RCPT TO: father@northpole.net¶
250 father@northpole.net...
Recipient ok
DATA¶
354 Enter mail, end with "." on a
line by itself
From: Donald Duck <donald-
duck@duckburg.com¶
To: Father
Christmas<father@northpole.net>¶
Date: 24 Dec 1999¶
¶
Peace be with you.¶
.¶
250 PAA00329 Message accepted for
delivery
quit¶
221 mail.foo.bar closing
connection
```


What you are doing in the example above is to perform a complete sending of an e-mail message, where you manually simulate the mail client with text, which you type on your keyboard.

Note that the sending of an e-mail message consists of a series of phrases sent back and forward between client and server. All application protocols do not work that way. HTTP 1.0 uses only a single sequence of octets from the client, and a single sequence of octets back from the server.

1.7 Architectures

Network protocols mean that several processes in different computers communicate with each other. By "Architecture" is meant the organisation of applications into different modules, and how these modules communicate.

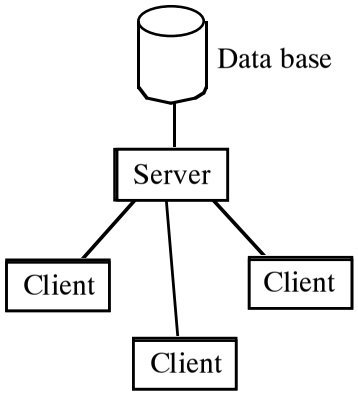


Figure 11: Simple client-server architecture

Figure 11 shows a simple and common architecture, with a single server, and a number of clients which access this server. With this architecture, the server is often responsible for a data base stored in the server, and the operations, which the clients perform on the server, may get och store data from this common data base. This architecture is simple to implement because there is only one protocol, the client-server protocol, and only a single data base, and thus no problems with co-ordinating information in different data bases. (Unless the clients have their own data bases, then co-ordination may be needed and the implementation and protocols may become more complex.)

The architecture in Figure 11 is common in local area networks, where users at local work stations access a common local data base.

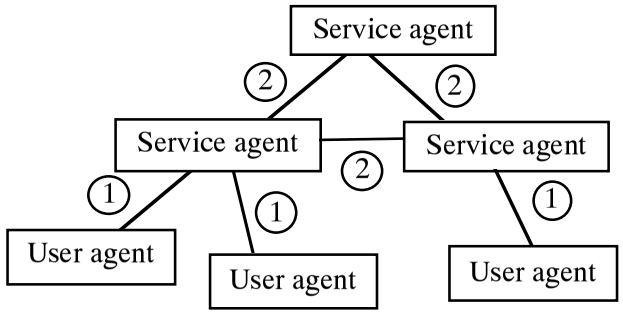


Figure 12: Architecture with several interconnected servers

Figure 12 shows a more complex architecture, which is used by some Internet applications, for example e-mail and Usenet News. There are (at least) two types of connections, labelled ❶ and ❷ in Figure 12. Connections between a user agent and a service agentis labelled ❶ in the the figure, and connections between two service agentis labelled ❷ in the figure. The needs may be different, and because of this, different protocols may be used.

In the case of e-mail, this is even more complex, because different protocols are used for sending mail from a user agent to a service agent, and for delivering mail from a service agent to a user agent.

In some cases, the same protocol definition is used both for the protocol between user agent and service agent, and between two service agents. But in such cases, sometimes different elements in the protocols are used, so that in reality the protocol between user agent and service agentis not exactly the same as between two service agents.

Even when a protocol is used between two similar agents, such as between two service agents, one of the agents is usually the client and the other the server in that particular connection. The client is the agent which opens the connection and requests services from the server, the server is the agent which receives requests from the client and performs them (or refuses them). Thus, in a connection between two service agents in Figure 12, one of them may be the client during that particular connection, and the other may be the server. That is why the boxes in Figure 12 are labelled “User agent” and “Service agent” and not simply “Client” and “Server”.

1.8 Chaining, Referral or Multicasting

If a database is distributed on several servers, then the information, which the client needs, may not be available on a single server. Information may have to be fetched from another or multiple servers. There are three common architectures for this, *chaining*, *referral* and *multicasting* as shown in Figure 13. With *chaining*, the user agent connects to only one server. This server may connect to another server, and that server may connect to yet another server, to get the information requested. The information is then returned recursively the reverse way. With *Referral*, the client connects to the first server. If this server does not have the requested information, it will tell the client to try another server. This is continued until a server with

the requested information is found. With *multicasting*, the same question is sent to several different servers, hoping that one of them has the answer. Multicasting can cost a lot, if you send a query to a number of servers for information which which you only need from one of the servers. In such a case, it can be more efficient to organize the data structure so that information can be located without multicasting. Multicasting can however be very efficient if the same information is to be sent at the same time to many recipients. In this case, special multicasting features in the lower layers are used, so that the same information need only be sent once between routers, even if there are many users of the data on each router. Examples of this is simultaneous video and audio broadcasting.

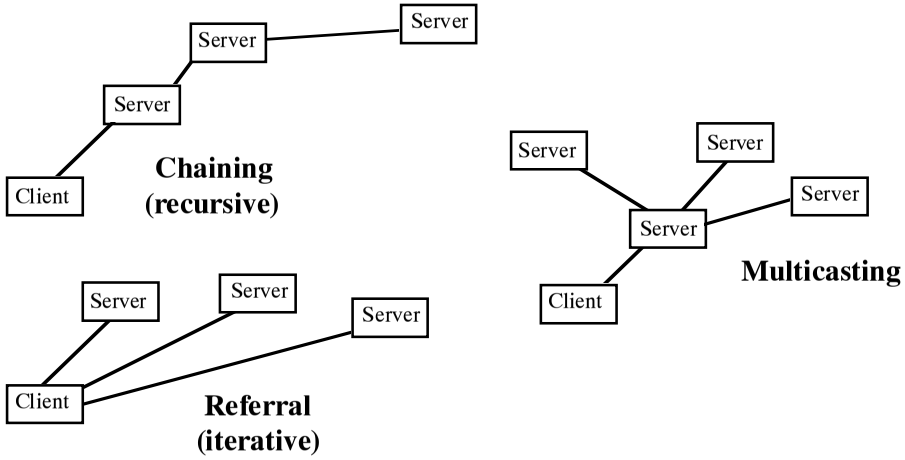


Figure 13: Chaining, Referral and Multicasting

Chaining has an advantage compared to referral. This is that since the information is returned recursively through the servers, a server

can cache the information, so that the next request can be handled from the cache instead of through chaining. Another advantage is that user agents often have slower connections than servers, and it is then more efficient with chaining, because less network operations have to be performed by the user agent through low-bandwidth connections.

An example of a system which uses both chaining and referral is the Internet Domain Name System, the DNS (see page 37). The DNS standards specify that referral is mandatory, chaining is optional. But in reality, chaining is used more often than referral, because of the advantages mentioned above.

1.9 Symmetric and Asymmetric Protocols

Nearly all application layer protocols are asymmetric. By this is meant that the language sent in one direction is different from the language sent in the other direction. Usually, when describing an asymetric protocol, one of the agents is named “client” and one is named “server”. Note that even between two similar agents, such as between two service agents, the protocol can still be assymetric. They are equal, but at a certain moment of time, the protocol is asymetric. For example, when sending e-mail between two service agents, the client is the sender, which can send messages, the server is the recieving agent which can refuse or accept messages.

The e-mail standard for the protocol between two service agents has a TURN command. The TURN command allows the two agents to change roles, so that the client becomes server and the reverse. The TURN command is however not used very much, and it might disappear from the standards in the future. And the TURN command does not make the protocol symmetric, because at a certain moment, the protocol is still asymetric.

The reason for the TURN command was that earlier, many connections were made through dial-up phone connections. Since such a connections are expensive and time-consuming to open, one connection could replace two with the TURN command. Today, however, almost all connections between service agents is through leased lines, so the TURN command is not needed any more.

1.10 Transfer of Responsibility

Protocols are specifications of which statements are sent between two agents, usually a client and a server. Protocols then specify what the client can send, and what the server can respond with. The interaction between client and server may, in some protocols, include several interactions sent back and forward.

An example of a common type of interaction, specified by a protocol, is the transfer of responsibility. This occurs in e-mail, where a store-and-forward technique is often used, as shown in Figure 14. The original sender sends the message

to a local mail server close to the original sender. This local mail server sends the message to a local mail server close to the final recipient. And this server delivers the message to the final reicpient. Sometimes, more than two transfer agents are involved on the route from the original sender to the final recipient. With such a store-and-forward structure, it is very important that the message does not disappear into a “black hole”. At any moment of time, one computer may crash, or the connection between two computers may break. Such occurences should not cause a message to disappear. To ensure this, each agent stores the message on disk in a such a way, that the message will still be there when the agent is restarted after a crash. This means that the responsibility to deliver the message is transferred between the agents. A typical protocol for such a transfer of responsibility may work in the following way:

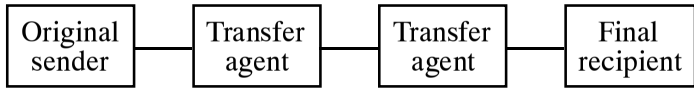


Figure 14: Store-and-forward of e-mail messages

1a. The clients send the message to the server.

1b. The server receives the message and stores it on non-volatile disk.

2a. The server then sends a response to the client that the message has been received.

2b. The client receives this response, and notes that the server has taken over responsibility for this message.

Until step 2b, the client continues to regard the message as unsent. It will thus, if needed, try to send the message once more, until it has received confirmation in step 2b that the server has taken over responsibility for the message. This means that there is very little risk that a message disappears. There is, instead, a small risk that a message is sent more than once.

The transfer of responsibility as described above consists of two interaction steps, one from the client to the server, and one from the server to the client. Some interactions use more than two steps.

1.11 Identification

Another common kind of interaction is identification between two agents. For example, a client can tell the server its name, and the server wants to confirm that this is really the client it claims to be. Identification can consist of the following steps:

1. The client connects to the server and sends its name.

2. The server sends a string of random digits to the client.

3. The client encrypts this string, and send the encrypted string back to the server.

4. The server decrypts the encrypted string, checks that it has received the same random digits it sent in step 2, and tells the client that identification has succeeded.

Since only the right client knows how to encrypt the random string in the right way, the server knows that the client is the one it claims to be. Here, four interactions back and forward between client and server were needed.

1.12 Transactions and Sessions

A session is a series of interactions back and forward between two agents, usually a client and a server. The interactions are performed, one after each other in time. A session often starts with identification (sometimes called “login” or “bind”), after that sometimes comes negotiation of capabilities, and then a number of transactions, and finally the end of the session.

Figure 15 shows the basic steps and state changes when transferring an email message from one agent to another using the SMTP protocol. As seen in the figure, there are a number of states, and in each state, only certain commands are permitted. Such a protocol is called a *stateful* protocol. A protocol which has no states is called a *stateless* protocol.

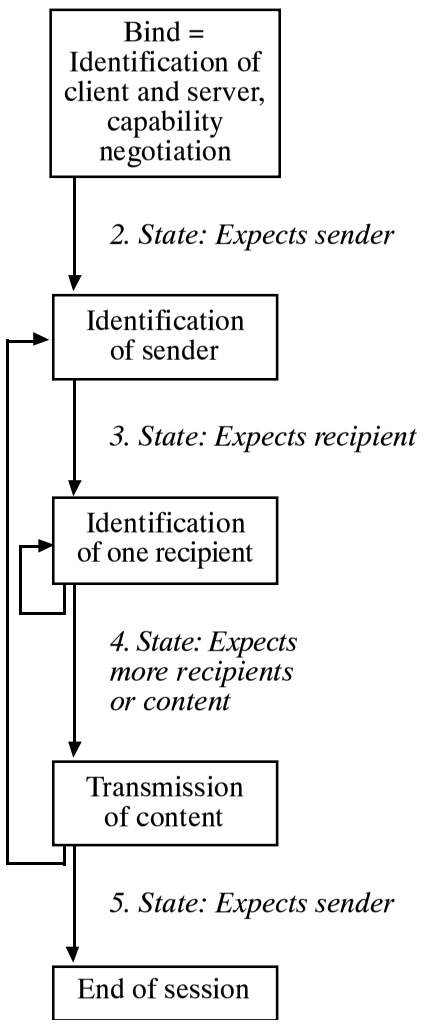


Figure 15: Session with changes of state

This figure shows (somewhat simplified) the changes of state during the transmission of an e-mail message using the SMTP protocol.

1. In the initial state, the serv expects the client to identif it self. No other commands are allowed in this state.
2. When identification is ready, the server expects th address of the sender.
3. When the sender has been given, the server expects th address of the first recipien
4. When a recipient address has been given, the server expects either an additiona recipient address, or the message content.
5. When message content. ha been sent, another message can be sent. This is the sam state as state 2.

Finally, a session ends with a command from the client to end the session. This can be given in any state.

Stateful protocols allow several interactions back and forward during a session. With other protocols, the client sends a transaction, gets a result back, and the connection is then terminated. Such protocols are stateless.

With some protocols, a session is kept open while waiting for the user to look at the results from previous interactions. Such waits can be very long. This is costly for a server, which may have to keep many sessions going at the same time. To reduce this cost, servers will often shut down the session if nothing has been sent for a certain time. This maximum wait time is called a *timeout*. If you have been using FTP, you may have noticed that many FTP servers will terminate the FTP session automatically, if you do not send any commands for a certain time.

To open and close a session will, however, also cost network and computer resources. If several interactions are needed to perform an action, it may be less costly to perform them after each other in the same session. But if the wait times are too long, it may be less costly to abort the session and start a new session.

To keep a session open and wait for a timeout is also costly for a server. It is more efficient, if the client sends a command to abort the session, so that the server need not wait for any timeout.

1.13 Turn-around Time, Pipelining and Windowing

A protocol may use a session with a number of interactions back and forward between client and server. The client has to wait for the response from the server on the previous command, before the client can send the next command. This takes time,

because the turn-around time to send a command and get the response back can be one or several seconds. The total time for a session with many such interactions will then be long.

Here are three methods to reduce these delays:

1. Specify more powerful commands, where more is done in one command, so that fewer interactions are needed.
2. Open several parallel connections. HTTP clients (web browsers) often keep four parallel connections for downloading the different parts of a web page (text, pictures, applets). Too many parallel connections is costly in resources for both the client and server, but with too few connections, dead time may occur when the client is waiting for data from all the connections.
3. In protoocols which use many small interactions, such as SMTP and NNTP, the delay can be used with a method called *pipelining* or *windowing*. Sometimes the word *streaming* is also used for this, although the word streaming also has other uses (see page 29).

By pipelining is meant that the client can send the next command, without waiting for the response from the server on the previous command within the same session. Commands are sent and responses received asynchronously. This is shown in Figure 16. (Note: the OK responses are still sent, they are not shown in the figure to the left to make the figure less complex.) The advantage with this method is that the session is performed faster. The disadvantage is that the error handling, if one of the commands is not accepted by the server, will be more complex. Also, data may have been sent unneccessarily if a previous command is rejected by the server. Pipelining is sometimes called windowing, when the number of commands sent in advance is limited. For example, if the window

size is three, this means that after sending three commands, the client must wait for respone to the first command before sending the fourth command. Pipelining should only be done when the protocol specification explicitly says that pipelining is allowed.

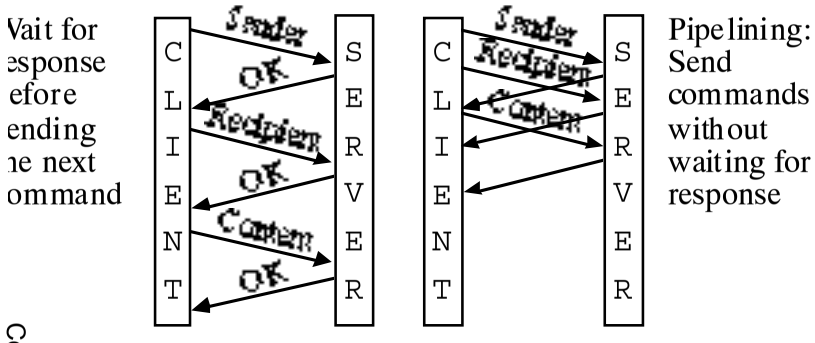


Figure 16: Pipelining reduces the time

TCP itself supports pipelining. A TCP connection is actually two pipes, one in each direction, and the sending process can send more data than the receiving process can handle. TCP will just store the overflow, but can also tell the sending process to make a break in sending if its buffers are full. %%%check if this is true%%%

1.14 Terminating a Connection

A connection can be terminated if one of the two partners closes the connection. For example, the server can close the connection if there has been no activity for a certain timeout period. More reliable and efficient is if the client sends a command to the server, asking for the connection to be closed. Such

commands are usually named EXIT or QUIT. The server then closes the connection. The advantage with this is that both agents know that the connection is closed, and no timeout is necessary.Figure 17 shows how a session is ended for HTTP 1.0 (World Wide Web) and SMTP (e-mail). Note that in both cases, the server knows when to end the connection, and no timeout is needed.

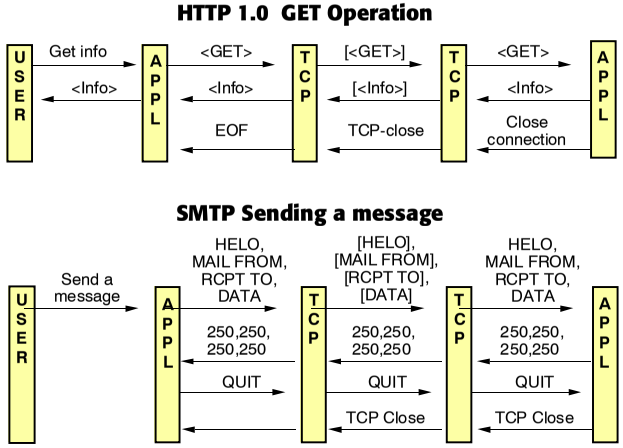


Figure 17: How a connection is ended in HTTP 1.0 and SMTP. In HTTP, the server closes the connection as soon as TCP tells it that all data has been transformed. In SMTP, the client sends a QUIT command, which tells the server to close the connection. Note that no timeout wait is necessary in either case.

1.15 Streaming

By streaming is meant that results are displayed for the user, before all data has been received by the user agent. For example, the top of a web page may be shown, while continuing to download the rest of the page.

The word streaming is also sometimes used in another meaning, to denote what in this book is called pipelining (see page 28). The concepts are connected, since pipelining is a way of assisting streaming.

In order to make streaming work better, the user agent must get a suitable selection of data which gives initial information about a web page. This is one reason why web browsers often open several connections at the same time; they can then download images in the beginning of a large web page, without waiting for the end of the download of the bottom of the web page.

The creator of data can alleviate streaming by sending data in a format where some data can be displayed before all has arrived. For example, many web browsers are not able to show a HTML table before the end of the table. Even if they can display the table earlier, new information later in the table may force them to reformat the table. It is therefore often not good to start a web page with a long table. Instead, one can have two tables, a small table for the first window, an d one or more additional tables for the rest of the web page.

Another method to support streaming, which is used with pictures, is to first send for example every 4th line, then every 2nd line, then the remaining lines. This allows the web browser to show a picture first in a more blurred form, and then sharper and sharper. This method of sending a picture is called *interlacing*.

In some cases, it is not possible to know the full content of a document, while sending it. When, for example, sending simultaneous video and audio, the end of the sending has not yet happended when the beginning is sent. In such a case, recipients often want to see the beginning of the broadcast before its end, and then streaming protocols are needed.

1.16 Intermediaries

Sometimes, the communication between two agents is passed through one or more intermediary computers. There are three main kinds of intermediaries (Figure 18):

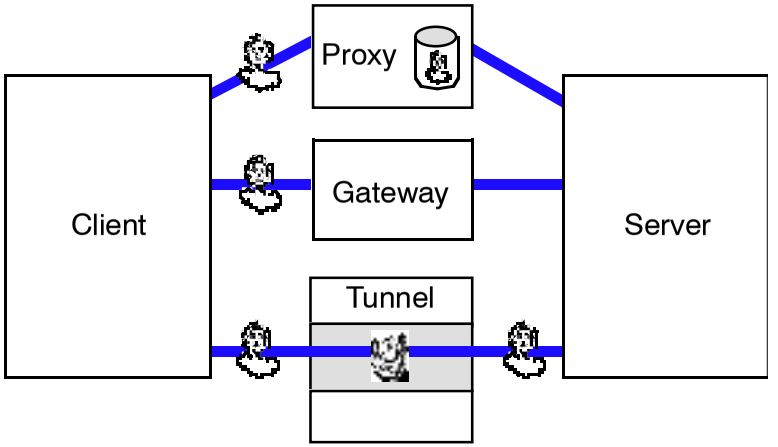


Figure 18: Intermediaries between client and server

Proxy: A server which caches data, so that some requests from the client can be answered directly by the proxy. Proxies also sometimes controls and limits what is allowed for various regulatory or security reasons. Such regulating proxies are called *firewalls*.

Gateway: A gateway usually is placed between two networks with different technology, and may transform the format of data from the format in one of the networks to the format in the other network. For example, an internal e-mail network within a company may use a number of enhanced features, which have to be mapped on the more restricted Internet mail protocols for mail going outside the company.

Tunnel: A tunnel is a way of transferring information between two networks, which both use the same protocol, through some intermediate network which uses another protocol. At the entrance of the tunnel, the data is transformed to a format which is only used for transportation. At the end of the tunnel, data is transformed back again to the original format.

The advantage with tunnels, as compared to gateways, is that no information is lost. With gateways, only the information supported by the protocols at both sides can be conveyed. The advantage with gateways, as compared to tunnels, is that you can reach agents using different protocols at either end.

1.17 Names and Addresses

Unique names are useful in many applications. The two most well-kown unique names on the Internet are WWW addresses (URLs) and e-mail addresses. Both of them are *globally unique*. By this is meant that no two objects on the Internet have the same name. Both are also *addresses*, they can be used to

locate an object. The WWW address is used to find a web page, the e-mail address to send a message to its recipient.

Sometimes, abbreviated names are used which are unique only within a certain domain. For example, if a user has the e-mail address “johnp@honeymelons.com”, then a message inside this company need sometimes only be addressed to “johnp”. It is however advisable to translate “johnp” to “johnp@honeymelons.com” as soon as possible. Otherwise there is a risk that a message with “To: johnp” in the header gets inadvertently sent to the Internet, where the short, non-unique form of the address will not work.

Unique names are used to locate objects, but they are also used to recognize duplicates of the same object, as tools for tracing the transfer of objects, and as tools for handling links between objects.

1.17.1 Domain method of allocating globally unique names

Most unique names are created using some kind of hierarchical structure. The domain name “dcs.ed.ac.uk” represents for example the department of computer science (dcs), within Edinburgh University (ed) within the academic community (ac) within the United Kingdom (uk). And an e-mail address “mary.smith@dcs.ed.ac.uk”

can represent a certain person within this department.

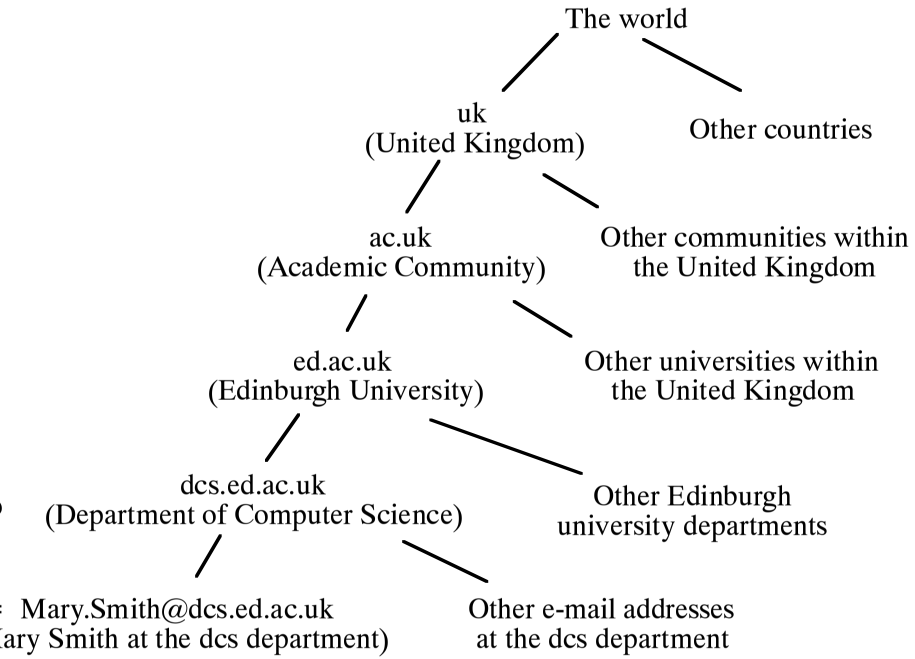


Figure 19: Hierarchical domain names

The main advantage with such hierarchical names is that the creation of new unique names can be decentralised. The “dcs” department at Edinburgh University can itself create new globally unique names, as long as these names end with “dcs.ed.ac.uk”. And Edinburgh university can itself create globally unique names for its departments, as long as the names end with “ed.ac.uk”.

Hierarchical names are also easy to understand for humans and can be used to guide the lookup of a name in a data base. If the names are hierarchical, the data base can be distributed. For example, one server can store names in Edinburgh university, the end of the domain name “ed.ac.uk” can then direct a retrieval to the Edinburgh university name data base. This means that no inefficient multicasting is necessary to look up a domain name.

Ideally, a name should be unique not only in space but also in time. No object should ever get the same name as another object already has had. Schemes which ensure this is however not much used. There is one such scheme called Object Identifiers. It is sometimes used to create unique names for data types and other protocol elements. The Message-IDs of e-mail messages are usually also designed to be unique in both time and space. For object identifiers, the uniqueness is guaranteed by the registration procedure. For e-mail Message-IDs, the time when the message was sent is usually included in the Message-ID, to ensure that it will be unique for an unlimited time in the future.

For ordinary web addresses, however, uniqueness in time is not guaranteed. If a company named Sunshine Flowers obtains a domain name sf.com, and that company goes bankrupt, the name sf.com may be sold to a company named Sexy Feet. And this may be embarrassing to those who have put links to www.sf.com in their web pages.

Some common types of unique names on the Internet are:

Name type	Usage	Example
Domain names	Names of organisations, hosts and servers	dcs.ed.ac.uk, www.dcs.ed.ac.uk
E-mail addresses	Names of e-mail senders and recipients	Mary.Smith@dcs.edu.ac.uk
Message-IDs	Names of e-mail messages. Used to recognize duplicates, and handle reply links between messages.	1999-07-23-131643*Mary.Smith@dcs.edu.ac.uk
IP address	Physical address of a host	[129.215.160.98] (IP addresses are actually 32-bit words in IP4 and 128-bit words in IP6, but they are usually written in the format above)
URL	A combination name which can be used for most kinds of names, but mostly used as addresses for web pages	http://cmc.dsv.su.se/iaps/

1.17.2 Hash Code Method of Creating Globally Unique Names

An alternative to the domain method is to give an object a globally unique name by computing a hash code of the object. There are methods of creating hash codes, which will make it very unprobable that two different objects will have the same hash code. The mostly used method for this is the MD5 [2] algorithm for computing hash codes.

Note that a hash code will only indicate that two objects are identical, not that they are the same object. Before computing a hash code, objects need sometimes be converted to a canonical format. For

example, different platforms indicate line breaks in a text with either only Carriage Return (CR), only Line Feed (LF), or a Carriage Return followed by a Line Feed (CRLF). The hash code may be different if two documents differ in their end-of-line encodings. The algorithm may then specify that all line breaks must be converted to CRLF before computing the hash code.

Important is also which parts of an object is included when computing a hash code. For example, if the hash code is only computed on the body of a message, two messages with different headers may be regarded as identical, which can cause serious problems. Example: A message saying “Oh, I love him” may have a very different meaning if it is a reply to a message saying “Do you like Saddam Hussein” or to a message saying “Do you like Jesus”.

1.17.3 User-friendly Names

Names which are globally unique will never be quite user friendly. We are accustomed to referring to objects with non-unique names like “Eliza Clark” or “The London office”. People will still know what we mean, because of the context in which we use them. Computers are not very good at that capability. Instead, objects on the Internet often have two names, one user-friendly name and one globally unique name. The header of an e-mail message may for example look like in Figure 20

with both a user-friendly name and a globally unique e-mail address. Another example is the title of a web page (user friendly) and its URL (globally unique).

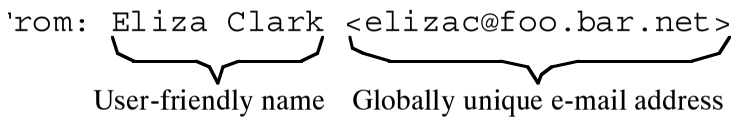


Figure 20: Use of names in e-mail headers

1.18 The Domain Name System

The Domain Name System (DNS) is a distributed data base which converts domain names to IP addresses. For example, you can ask the DNS for the IP address of the domain “dcs.ed.ac.uk” and the DNS will return “[129.215.160.98]”.

The DNS is probably the most commonly used service on the Internet. Whenever a program uses a domain name, for example “http://www.ed.ac.uk/” or “elizac@foo.bar.net” the first step in locating the object is to convert the domain name “www.ed.ac.uk” in the first example, and “foo.bar.net” in the second example, to an IP address. This IP address is then used to physically locate the host when a network connection is established to retrieve a web page or send an e-mail message.

Figure 21 shows a small excerpt from the conceptual structure of the DNS. Conceptually, a search for the IP address of www.dsv.su.se should start at the international root server, and then be

directed from there to the .se server, from there to the .su.se server, and from there to the .dsv.su.se server. In reality, it does not work like that, otherwise the top-level name servers would be severly overloaded. Through caching, most name servers have caches of the addresses of the most commonly used names, including the top-level domains, so that they only have to connect to the top-level domains once or twice a day to verify their cache. Also, all name servers are at least duplicated, so that if one is down, another can replace it. The higher level name servers are duplicated more than twice to distribute load and increase reliability.

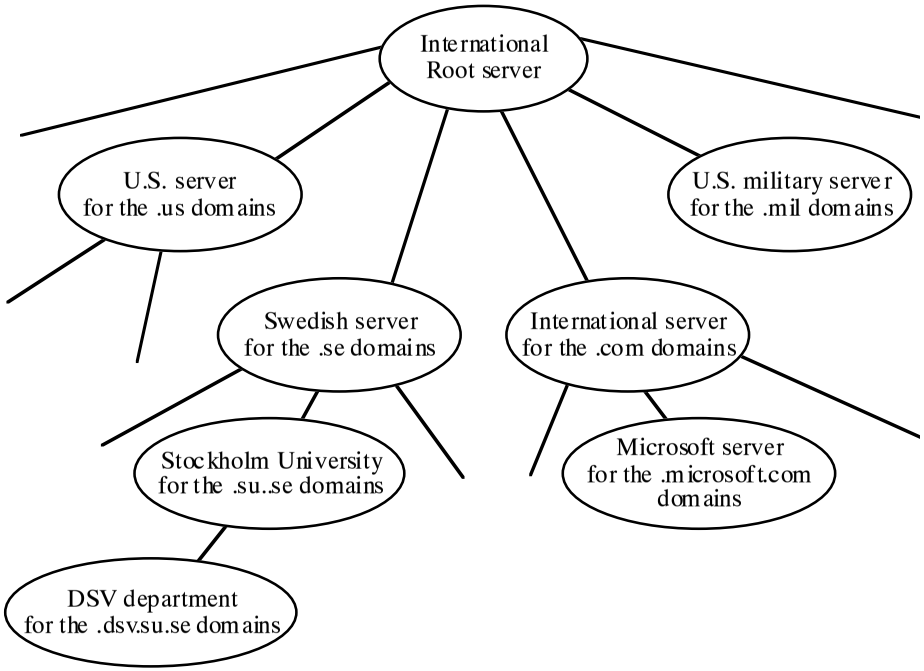


Figure 21: Conceptual structure of the DNS server hierarchy

The DNS uses caching, chaining and referral (See “Chaining, Referral or Multicasting” on page 20).

One peculiar effect has been that small countries with few Internet servers sell their domain name structures to organisations outside their own country. A well-known example is the “.nu” domain, which belongs to the small Island Niue in the Pacific, but its domain names have become popular all over the world, I myself have registered the domain “palme.nu” for my family.

1.18.1 MX records

The DNS actually consists of two data bases stored together. These two data bases are so-called A records, which are used to look up Internet hosts (for example to locate a web page) and MX (Mail Exchange) records, which are used to look up e-mail servers. The reason for this is that companies often have a central e-mail server, to which all e-mail is to be delivered, and which may be different than hosts used for other services.

E-mail is also sometimes sent through store-and-forward. Thus, a look-up in the DNS for an MX record may return a list of hosts in a priority order. The mail should be sent to the first host, but if this host is not available, mail can be sent to the second host. The secondary host will usually forward the mail to the primary host, but it may

also have the capability to deliver the mail to its final recipient.

Since e-mail can be sent through store-and-forward, a server which receives an e-mail message may contact another server and send the message to this server. To find out which host to send the message to, the intermediate server may also use the DNS. There is then a risk that an e-mail message will be sent for ever in a loop back and forward between two servers. If server A finds that B can handle this e-mail address, and server B finds that A can handle this e-mail address, such a loop may occur. To avoid this, the DNS assigns a priority to each MX record. Suppose you look for the MX record for the domain “cs.edu.ac.uk”. You may then be given the following table:

8 mailrelay.ed.ac.uk

5 mailhub.dcs.ed.ac.uk

6 mh2.dcs.ed.ac.uk

A host is not allowed to forward a message from a host with a lower priority number in the DNS to a host with a higher priority number. This will protect against mail forwarding loops.

1.19 Top-level domains

The top-level domains (the last domain in the domain names in DNS) are either two-letter country codes or more-than-two-letter codes representing different kinds of organisations. For example, the domain name “dsv.su.se” has the top level domain “se” which is the country code for

Sweden, and the domain name “ietf.org” has the top level domain “org” which represents non-commercial organizations.

Each country has an organisation responsible for distributing the second-level domain names for that country. It is thus this organisation, which has given Stockholm University the domain name “su.se”.

The country codes are taken from an ISO standard for country codes, the same codes which are used in car registration numbers, international postal codes, etc. There is however one exception. The United Kingdom of Great Britain has the ISO country code “gb” but has he DNS top level domain “uk”. Both are abbreviations of different parts of the full country name. The reason for this is historical. It started that way, and it was then too late to change it. Possibly the difference is also because this country is known for wanting to do things their own way. They drive to the left, while most other European countries drive to the right, and for a long time they insisted in writing the domains in reverse order. Thus, instead of “dcs.ed.ac.uk” they wrote “uk.ac.ed.dcs”. This caused lots of problems, but nowadays they have changed to the same order as the rest of the world uses.

The set of more-than-two letter codes may be extended. Here are the codes decided on when this is was written (July 1999):

)M commercial entities

)U 4-year colleges and universities

NET organizations directly involved in Internet operations, such as network providers and network information centers

ORG miscellaneous organizations that don't fit any other category, such as non-profit groups

GOV United States Federal Government entities

MIL United States military

FIRM Businesses

STORE Online stores and malls

WEB Web-related organizations

ARTS Cultural and entertainment organizations

REC Recreational organizations like park districts

INFO Organizations who are primarily sources of information

NOM Individual, personal domain names

The first seven domains in this list have been in use for many years, the rest have been added recently.

The distribution of domain names ending in these generic domains has been a very controversial political issue. A new organization, ICANN (The Internet Corporation for Assigned Names and Numbers). ICANN will also take over the tasks of registration of other kinds of unique identifiers for Internet, which previously was performed by IANA (Internet Assigned Numbers Authority). When you read this, ICANN has probably already started operation and taken over the tasks of IANA.

1.20 The old versus new problem

Internet protocols are usually implemented on many different computers, which are co-working using the protocols. This can make it difficult to update to new versions of the protocols. Such a

change might require that all the existing agents have to be replaced at exactly the same time. With standard protocols, like those for e-mail and web access, there are millions of agents, and such a replacement of all of them at one time with new versions is very difficult.

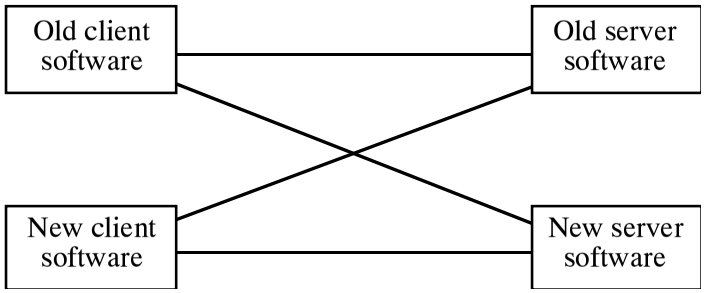


Figure 22: Difficulty of co-working between different versions of protocols

If different companies develop the various agents, this will of course be even more difficult. Some companies may want to support new features in a new version of the protocol, other companies may not want to do this.

It is possible to design a protocol so that it is prepared for future extensions. If this is done in a suitable way, it may be simpler to add new features to a standard with less co-working problems.

Many Internet standards had few, or badly planned, features to support extensions, in their first versions. This forced developers of new versions of the standards to use very messy and untidy solutions.

A horror example is the MIME standard for sending binary attachments in e-mail. Since the old e-mail standards only supported 7-bit characters in e-mail messages, anything else had to be encoded in a 7-bit format.

Here is an overview of good and bad methods of solving these problems.

1.20.1 Version number

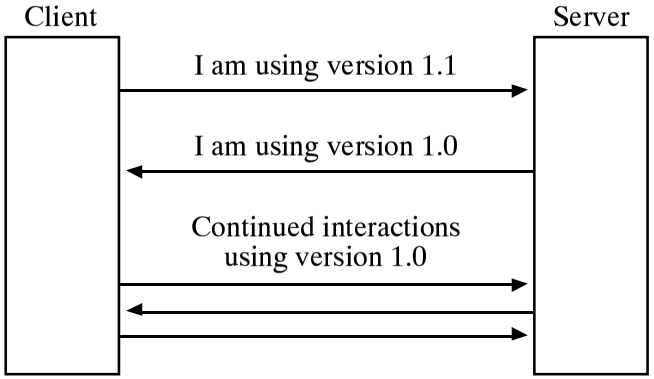


Figure 23: Version number method

With the version number method (see Figure 23), changes in a protocol mean that the protocol is given a new version number. Communication between client and server starts with exchanging version numbers. After this, both agents are expected to adhere to the lowest version any of them used. This means that all agents have to support both the new protocol and many or all older versions, which other agents may be using.

The main disadvantage with the version number method is that if you make two independent extensions, there is no way of indicating that you can use only the first, or only the second extension.

Generally, version numbers are used for major changes. And major changes are difficult to handle. Some protocols have version numbers, but never get any new version number. An example of this is MIME. All MIME e-mail headers must contain the statement

```
MIME-Version: 1.0
```

but there may never be any new version of MIME. This statement is, however, not meaningless. It is a way of saying that this message is in MIME format. So it cannot be removed from the standard.

1.20.2 Feature Selection Method

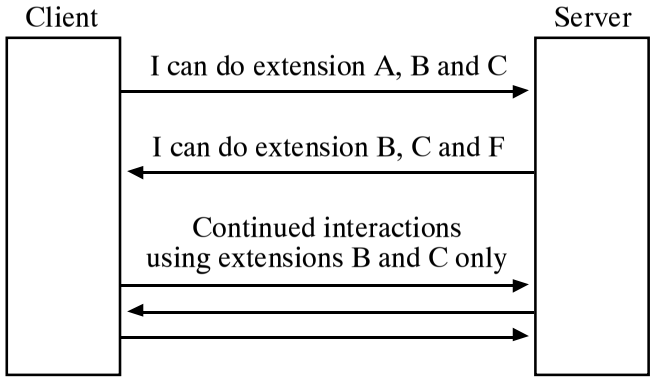


Figure 24: Feature selection method

With the feature selection, interaction starts with the client and the server each listing which features they support. After that execution continues, using only features supported by both (plus certain mandatory basic functions which do not have a feature name).

The advantage with this method is that an agent can choose to support any set of features, and still co-work with another agent which has chosen any other set of features. Co-working will of course work better, if the two agents have more features in common.

Another advantage with feature selection is that standards developing organizations can specify new extensions, without being sure that they will be widely accepted. The implementors on the market can choose whether to support a new feature. There is however a risk with standardizing too many doubptful features, since the more features are implemented by some, but not all vendors, the less weill will their software be able to communicate.

With feature selection, and if the features are given names which are more than one character, it is even possible that different implementors can specify and develop new features, independently of each other. If they do this, however, there is a risk that two implementors will choose the same name for two incompatible features. This issue is discussed more in the next section.

1.20.3 Feature naming

With many methods to allow extensions to standards in an orderly manner, new features are identified by some kind of feature name. There is then a need to avoid two different implementors from defining two different features, but giving them the same name. If they do this, and their applications are to interact, serious problems can occur.

To avoid this problem, there is a need to generate globally unique names for new features. There are two methods commonly used to do this. One method is to use a hierarchical naming space. For example, a feature can be identified by a URL. This URL can refer to a web object which describes this feature. Another well-known method of obtaining globally unique feature tags is the "object identifiers" standardized as part of the ASN.1 standard. Object identifiers have got widespread use even when ASN.1 is not used. The advantage with object identifiers is that they are globally unique in both space and time, and that anyone can obtain a new object identifier. Since object identifiers consist of a series of numbers, not a string of characters, they do not imply any meaning in the choice of words, and there is thus not so large risk of conflicts as there has been with domain names (when one company objects to someone else using the name of that company as a domain name.)

Compendium 4 page 28

Another method is to use a registration authority. Anyone who creates a new feature, registers its name with the registration authority. For Internet IANA (to be taken over by ICANN) is the main organisation for registering such names. IANA has a large number of different feature lists, which it handles for this method. Here are some of the most important feature lists handled by IANA:

Character sets	See page %%
HTTP parameters	HTTP transfer compression methods
IMAP 4 capabilities	Feature selection for IMAP, see page %%
Languages	Tags indicating the language of a document
Media feature tags	Tags indicating features of print and display media, such as paper size, color depth, etc.
Media types	See page %%
Port numbers	See page 12
Transfer encodings	Encodings of binary and 8-bit characters in MIME, see page %%
URL schemes	Values allowed in the initial string (before the ":") in URLs to indicate various access protocols, see page %%

Some features allow anyone, including a company or another standards organisation, to register a value. Other features only accept values defined by official standards. Allowing non-standard features has some problems. Just by registering a feature name, it is given a kind of official backing. Standards organisations often want to make some kind of checking of how reasonable new features are. But if they do such checking, the allowed features become a kind of standard. And this may not be good either, because the new features have not been checked thoroughly enough for an accepted standard.

One solution to this dilemma, which has become more common in recent years, is to precede non-standard features with "vnd." followed by the vendor name. A vendor name as a prefix of a feature clearly says that this is a vendor-specific feature, not a standard feature. Example: "application/vnd.ibm.modcap" is a media type registered by IBM for "Mixed Object Document Content Architecture". Since this method has not always been used, some early vendor-specific formats do not have such names, for example the media type "application/pdf" for the PDF document format used by Adobe Acrobat.

A common practice, explicitly specified in some standards, is to allow non-registered feature tags, if they begin with "X-". This was intended to allow experiments with new feature tags before they are registered. The experience with this method, however, is not good. Some experimental feature tags started out with "X-" and became so widely accepted, that they had to keep the "X-" even when they became so common that they should have been standardized.

1.20.4 Built-in Extension Points

One method of making extensions easier is to provide built-in extension points in a protocol. A built-in extension point is a part of a protocol, where extensions can be added in such a way that old implementations, which do not understand the new extensions, can at least recognize that they are extensions.

Some standards have such extension points built into the standard in specified places. But even standards without such explicitly specified built-in extension points, may still in their practical usage allow extensions at certain points. Two examples:

E-mail header fields

E-mail headers contain standardized fields like "To:", "From:" and "Date:". But anyone can add their own header fields. The standard for e-mail only allows such non-standard header fields if they begin with "X-", but there are common non-standard e-mail header fields which do not begin with "X-", such as "Mailer" or "Return-Receipt-To". Some of these are not much liked by standards making organisations, but they do exist and can be used between agents which support them in the same way.

HTTP request methods

The HTTP protocol is probably the Internet protocol which most often is extended to various special protocols for special needs. One way of extending HTTP is to add your own request methods. HTTP has certain built-in standard request methods, which are indicated in the first line of a HTTP request. Examples of such built-in request methods are GET, HEAD and POST. People who extend HTTP often do this by adding their own request method names. There is, of course, always the risk that the same name you

have chosen for your own request method, gets used by the standards making organisations for a different request method in the future, so this extension method is not very safe. A way to make it safer is to start the new request method with "X-" or "VND." followed by a vendor name. This method of making new methods safer is not (in the case of HTTP) specified in any official standard, but is becoming more and more common practice, for various feature tags.

1.21 Standards Terminology

Certain words have a %%%

1.21 The Standards Making Process

1.21.1 Who makes the standards?

Many different organisations can be involved in making standards. Standards can be developed by:

Developer	Example	Advantages	Disadvantages
Companies	Postscript and PDF were developed by Adobe, RTF was developed by Microsoft.	Fast development, avoids some disadvantages with standards organisations.	May be too specialized for the products of the developer.
Consortiums of companies	Unicode consortium which develops the Unicode character set standard.	If many leading vendors are members, such standards can get widely accepted.	Sometimes two or more competing consortiums develop competing standards.
General standards organisations	Internet Engineering Task Force (IETF), International Standards Organisation (ISO), International Telecommunications Union (ITU), World Wide Web Consortium (W3C).	Many different organisations can contribute their knowledge and experience.	Conflicts are sometimes resolved by ambiguity in the standard or multiple ways of doing the same thing.

1.21.2 Standards stages

General standards organisations have rules which regulate the standards making process. Examples of such rules are stages of progress of standards.

For example, IETF has the following stages:

IETF Draft
: An IETF Draft is a document produced during the work of developing a standard. IETF drafts are not standards and it can be dangerous to develop based on them. Many of them never become standards or are changed much before they become standards. IETF deletes all IETF drafts after usually 6 months, in order to clarify that they are not real standards.

Experimental Standard
: Most standards never get into this stage. It is used when it is unclear if this should become a standard or not, or for standards of interest to very limited communities. If an Experimental Standard is successful, it can progress to a Proposed Standard.

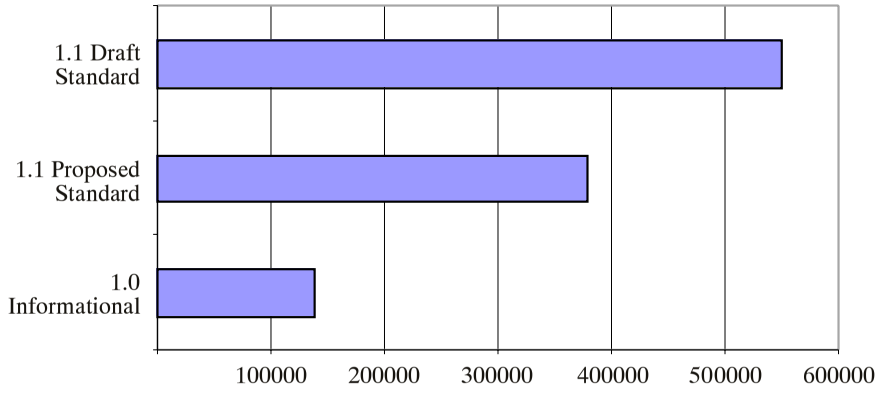
Proposed Standard	Proposed Standards is the first stage of a new standard. The name gives the impression that this is only a proposal, but in reality, proposed standards are often widely implemented. For example, the HTTP standard was first published as Informational in 1996, then as Proposed Standard in 1997, then as Draft Standard in 1999.
Draft Standard	A Proposed Standard can progress to Draft Standard if it is widely implemented. IETF has a rule that any facility of a Proposed Standard which is not supported by two independent co-working implementations has to be removed when the Standard progresses to Draft Standard. This means that standards are often simplified when progressed from Proposed to Draft. For example, the Content-Base HTTP header was removed, when the HTTP/1.1 standard progessed from Proposed to Draft in 1999. This rule is very valuable, since it helps weeding out facilities in standards which are not much used. Such facilities can cause problem in co-working between implementations using them and not using them.
Standard	This is the last stage of a widely supported standard. It usually takes a long time before a standard reaches this stage. HTTP, for example, has not yet (when this is written, February 2001) become a full Standard in spite of its wide acceptance.
Historical Standard	An old standard, which should not be used any more.
Best Current Practice	A recommendation on how to use one or more standards for a particular purpose, or any other documents which need to be developed in the same careful way as a Proposed Standard, but should not itself be a Proposed Standard.
Informational	Other documents of interest to implementors. For example, a summary of information from other standards can be published as an Informational Standard. The reason such a document does not become a Standard, is that if the same thing is specified in more than one standard, there is a risk of contradiction. Advice on how to best implement a standard can also be published as Informational documents. Some Informational documents are not reviewed as carefully as standards are.

IETF has face-to-face meetings three times a year. At these meetings, many different standards group meet. Each group is given 1-6 hours, and there are many parallel sessions during a whole week. Groups are of two kinds, established IETF working groups and BOF (Birds Of a Feather). BOF meetings have the task of deciding if IETF should form a working group. Sometimes, a whole standards is developed during a few succeeding BOFS without ever forming a working group.

Each working group has an e-mail mailing list, through which much of the work is done between IETF meetings. Sometimes, but not commonly, a working group can have a face-to-face meeting between the three regular IETF meetings.

The first version of a new standard is published as an IETF draft. After this, the proposal is usdually discussed in a BOF. Based on the BOF, a charter is developed for the working group. The charter specifies clearly what is to be standardized, and also usually what is not to be included in the standard. The charter also contains a time plan.

IETF tries to develop simple standards on well-known topics, and tries to avoid research and untested ideas. However, some standards tend to become rather large an complex, for example HTTP.



The figure above shows the number of characters in the text of HTTP 1.0, HTTP 1. Proposed and HTTP 1.1 Draft. HTTP 1.1 Draft is a 177 pages large and very comple document. One reason why IETF standard sizes has grown is that the standards work wa dominated by university people in the 1980s, but now people from large commercia companies have largely taken over. Large commercial companies have an interest in larg and complex standards, because this will keep smaller competitors from the market. HTT 1.1 also sometimes describes more than one method to do the same thing, sometimes wit small differences. One reason for this is the need for compatibility with earlier version which means that a new better method is specified together with an older, less good way c doing the same thing.

Work in standards organisations often strives at reaching "consensus", specifications which everyone agrees on. However, trying to reach consensus sometimes results in standards specifying more than one way to do something, in order to make everybody happy. This is of course not good. If the standard specifies two ways of doing the same thing, one implementor may choose only one of the ways, another implementor only another of the ways, and their implementation will then co-work well. Striving to reach consensus also can result in ambiguous statements on controversial issues. These problems with consensus has caused IETF to say that it strives after "rough consensus". By this is meant that most competent people agree, but a few deviants are accepted, especially if the deviants are known to be people who are unwilling to accept aggreements or known to often have different ideas.

IETF is rather restrictive in its use of voting. Straw polls are sometimes made, but in general IETF tries to resolve issues by other ways than voting.

IETF is very open, anyone is allowed to participate in the standards work, and anyone who can afford the cost is allowed to go to the IETF meetings. ISO has another organisation, it is based on national organisations in different countries (for example ANSI in the U.S. and DIN in Germany) who send representatives to the standards meetings and vote on the standards.

An important reason why IETF has been more succesful than ISO in the area of networking standards is that IETF places more stress on implementation experience before finalizing a standard. This means that IETF standards tend to be simpler and have less bugs than ISO standards.

ISO has a different set of stages of standards:

Committee Document	Draft which has been carefully prepared, but is not ready to become a standard yet.
Draft Standard	First version of a standard.
Full Standard	Final version of a standard
Functional Standard	An additional document, which restricts an existing standard and clarifies ambiguous issues. Functional standards are usually developed by other organisations than ISO. The functional standards are needed to counteract the complexity and ambiguity in the original standards.

Because ISO does not test implement as much before publishing a standard, bug corrections (usually named "Addendums") are common.

1.22 OSI versus the Internet

During the 1980s, many people understood that computer networks were going to revolutionize society. Two sets of standards were developed. The International Standards Organisation (ISO) and the International Telecommunications Union (ITU) developed the Open Systems Interconnection (OSI) set of standards. And people at universities and research laboratories developed the Internet. And we all know that the Internet won over the OSI, even if some Internet standards have taken over some functions from OSI.

Some reasons for the success of Internet:

(This is a *very* controversial issue, which many people feel very strongly about. Some of them will strongly claim that one of the reasons listed below is the primary reason for the success of the Internet. Other people will claim that another reason is primary.)

1. Internet standards were in the beginning much simpler and easier to implement. However, during the 1990s, there has been a tendency to make Internet standards more complex.
2. The Internet process of developing standards requires, that all features not implemented by two independent implementations must be removed when a standard is progressed from "proposed" to "draft" standards status. This means that unnecessary or unimplementable features will be removed.
3. The rules for consensus forming are different in the involved standards organisations. IETF requires "rough consensus" which means that most reasonable people agree. ITU requires absolute consensus among ITU members. ISO relies on majority voting. Too strong requirements on consensus can lead to the inclusion, in the standards, of different ways of doing the same things, in order to make everybody happy. But such standards are of course not good, a good standard should avoid different ways of doing the same thing. Otherwise, there is a risk that some implementors will choose one option, other implementors another option, and their products may then not be able to interwork.
4. OSI tried to develop general and universal solutions to cope with all existing and anticipated future usage. This made the standards more logical and wellstructured, but also much more complex.
5. Internet was in the 1980s and the beginning of the 1990s heavily subsidized by the U.S. government.
6. Because Internet started as a university and research network, much valuable and free information was made available to give Internet a good start.
7. OSI was developed by large telecom companies, who wanted to use the standard to dominate the market. Internet, on the other side, has always been very strongly oriented towards letting many different providers develop their own solutions, and letting the market decide which of them would succeed.

When considering this issue, it might be interesting to note that in one country, France, a network with many similarities to Internet was very successful

already in the 1980s. That network was named Minitel. It was somewhat OSI-oriented, but a very simple subset, and it was also subsidized in the beginning by France Telecom. Minitel was also strongly oriented towards letting many different providers put up different services. And similar networks to Minitel, but more oriented towards central control, did not succeed as well as Minitel in other countries, like Germany and the United Kingdom.

Because of this, I personally believe that the bottom-up structure of the Internet (item 7 in the list above) was the most important factor for its success.

7.1 References

[1]	*IANA Port numbers Registry*. http://www.iana.net/numbers.html#P.
[2]	Rivest, R. *The MD5 Message-Digest Algorithm.* Internet RFC 1321, April 1992

A Beginner's Guide to URLs

What's a URL? A URL is a **Uniform Resource Locator**. Think of it as a networked extension of the standard *filename* concept: not only can you point to a file in a directory, but that file and that directory can exist on any machine on the network, can be served via any of several different methods, and might not even be something as simple as a file: URLs can also point to queries, documents stored deep within databases, the results of a *finger* or *archie* command, or whatever.

Since the URL concept really pretty simple ("if it's out there, we can point at it"), this beginner's guide is just a quick walk through some of the more common URL types and should allow you to be creating and understanding URLs in a variety of contexts very quickly.

File URLs

Suppose there is a document called "foobar.txt"; it sits on an anonymous ftp server called "ftp.yoyodyne.com" in directory "/pub/files". The URL for this file is then:

```
file://ftp.yoyodyne.com/pub/files/foobar.txt
```

The toplevel directory of this FTP server is simply:

```
file://ftp.yoyodyne.com/
```

The "pub" directory of this FTP server is then:

```
file://ftp.yoyodyne.com/pub
```

That's all there is to it.

Gopher URLs

Gopher URLs are a little more complicated than file URLs, since Gopher servers are a little tricker to deal with than FTP servers. To visit a particular gopher server (say, the gopher server on gopher.yoyodyne.com), use this URL:

```
gopher://gopher.yoyodyne.com/
```

Some gopher servers may reside on unusual network ports on their host machines. (The default gopher port number is 70.) If you know that the gopher server on the machine "gopher.banzai.edu" is on port 1234 instead of port 70, then the corresponding URL would be:


```
gopher://gopher.banzai.edu:1234/
```

News URLs

To point to a Usenet newsgroup (say, "rec.gardening"), the URL is simply:

```
news:rec.gardening
```

Currently, network clients like NCSA Mosaic don't allow you to specify a news server like you would normally expect (e.g., `news://news.yoyodyne.com/rec.gardening`); this may be coming down the road but in the meantime you will have to specify your local news server via some other method. The most common method is to set the environment variable `NNTPSERVER` to the name of your news server before you start Mosaic.

HTTP URLs

HTTP stands for HyperText Transport Protocol. HTTP servers are commonly used for serving hypertext documents, as HTTP is an extremely low-overhead protocol that capitalizes on the fact that navigation information can be embedded in such documents directly and thus the protocol itself doesn't have to support full navigation features like the FTP and Gopher protocols do.

A file called "foobar.html" on HTTP server "www.yoyodyne.com" in directory "/pub/files" corresponds to this URL:

```
http://www.yoyodyne.com/pub/files/foobar.html
```

The default HTTP network port is 80; if a HTTP server resides on a different network port (say, port 1234 on www.yoyodyne.com), then the URL becomes:

```
http://www.yoyodyne.com:1234/pub/files/foobar.html
```

Partial URLs

Once you are viewing a document located somewhere on the network (say, the document `http://www.yoyodyne.com/pub/afile.html`), you can use a *partial*, or *relative*, URL to point to another file in the same directory, on the same machine, being served by the same server software. For example, if another file exists in that same directory called "anotherfile.html", then `anotherfile.html` is a valid partial URL at that point.

This provides an easy way to build sets of hypertext documents. If a set of hypertext documents are sitting in a common directory, they can refer to one another (i.e., be hyperlinked) by just their filenames -- *however* a reader got to one of the documents, a

jump can be made to any other document in the same directory by merely using the other document's filename as the partial URL at that point. The additional information (access method, hostname, port number, directory name, etc.) will be *assumed* based on the URL used to reach the first document.

Other URLs

Many other URLs are possible, but we've covered the most common ones you might have to construct by hand. At the top of each Mosaic document viewing window is a text field called "Document URL"; if you watch the contents of that as you navigate through information on the network, you'll get to observe how URLs are put together for many different types of information.

The current IETF URL spec is here; more information on URLs can be found here.

Return to the overview

Distributed file systems

Much of the current research in autonomous replication has been heavily influenced by the caching subsystems of distributed file systems. Efficient operation of a large-scale distributed file system depends on caches to reduce the load on file servers; systems like Sun's NFS [26][27] that do not rely on long-term caches cannot support more than a few hundred clients, whereas systems like XFS [35] that distribute server load among clients are designed to scale to many thousands of clients.

The challenge of offering distributed access to a wide-area information system is almost identical to that faced by a large-scale distributed file system; some researchers have proposed making this relation explicit [33] by actually serving files from the Web over a wide-area distributed system like the Andrew File System [28]. Although this idea is still contentious, because it is not yet clear if the semantics of the World Wide Web match those of a distributed file system closely enough to simply implement the one on top of the other, we can nevertheless still learn much from the distributed file system community about building large-scale systems. In the next section we discuss the large-scale distributed file system designed by Blaze as his PhD thesis.

Large-Scale Hierarchical File Systems

The goal of Blaze's thesis [4] was to design a distributed file system that could scale to a very large size, operating across the Internet. He achieves this goal by building an explicit hierarchical system where clients can not only cache items indefinitely but can also serve them to other clients.

Blaze introduces four types of scale that a successful system must address: population, traffic, administrative, and geographic. Each type has its own set of issues. Scaling with population requires a system to cope with growth in the total number of potential clients and implies that a server should not need to store any information about its clients. Scaling with traffic requires the ability to handle the workload generated by all clients. Scaling administratively implies an ability to span autonomous entities. Finally, coping with geographic scale requires the ability to span large distances, with the inherent latency implied therein.

The Web in its current form addresses these by eliminating many of the frills associated with distributed file systems such as caching and write-sharing. Our challenge is adding an optimized caching system to the Web without reducing its ability to scale along these lines.

Before designing his system, Blaze gathered traces from an NFS server to determine optimal cache strategies for his distributed file system. The most important trend that emerged is that files tend to display strong *inertia*, where the same files tend to be opened again in the same mode by the same user. It appears, for example, that keeping a file in a client cache for two hours results in an average hit rate of over 66%.

Blaze also found that most files are overwritten while still very young. If a file is not changed soon after creation, it becomes unlikely that it will be changed. Files therefore rapidly move toward a state of being read-only, and shared-read files are even less likely to be written to. Finally, files opened for reading by other machines are likely to be opened for reading by still others. These results imply that cache sharing will work well for a distributed file system because it is unnecessary to update shared-read files often. We will see that a similar analysis drives the Alex FTP cache as well, and in chapter we will see that these results also apply to the World Wide Web since Web files that are globally popular changes less often than those that are primarily used locally.

Blaze analyzed a variety of different caching schemes. Flat file systems such as NFS, where all clients connect to the same server, inevitably suffer from a bottleneck effect. Eventually the single server is unable to cope with the connected clients. Even with optimal client caching strategies the server must still deal with 8-12% of the original traffic, and as the system grows this will eventually overwhelm any server.

The only solution is to distribute load among several servers. Replicating all files at secondary servers, however, is too expensive, and caching items at secondary servers as requests pass through them creates delays. Furthermore, having an intermediate cache process all results yields surprisingly little gain. Most objects not in the client cache were not in the intermediate cache either. The answer is to look in the caches maintained by other clients: 60-80% of client cache misses were of files already in another cache.

When the load at a server becomes too high, the server stops serving the file directly and only sends out lists of other servers that it knows have cached the file. Clients attempt to first satisfy file requests from their own file caches, then from servers listed in a local name cache, and finally from the file's primary host. If a client receives a list of servers instead of the file, it caches that list in its name cache, and attempts to contact one of those servers instead. We will use a similar method in section to locate replicated files on the Web.

There are several failure modes with this system. When two machines can both talk to a server but not to each other, the server may redirect one to another causing a fault. There are also some consistency issues that arise from computers that are temporarily disconnected from the server and can not receive invalidation messages.

There are several ways in which we apply Blaze's work to our own. We already mentioned that we use his method for locating nearby replicas, and that Web files behave in a similar way to files in a distributed system. We also use his definitions of scalability in section to discuss our own system's scalability.

Alex

One system that bridges the gap between data replication and distributed file systems is Alex [8]. Alex allows a remote FTP server to be mapped into the local file system so that its files can be accessed using traditional file operations.

This is accomplished through an Alex server that communicates with remote FTP sites through FTP, caches files locally, and communicates with the local file system through NFS. When the user wants to access a remote file, Alex downloads the file using FTP and then caches the file locally.

When another user needs the same file and tries to access it through the file system, it is not necessary to FTP it again. The file will simply be taken out of Alex's cache rather than from the original FTP site. This provides a savings in bandwidth and also provides a measure of reliability in the face of network failures. It also saves local disk space because users do not have to make personal copies of the same FTP files; users simply use the files directly off of the Alex server.

Alex's most novel feature is its cache-consistency mechanism. As we have already seen from Blaze's thesis, the older a file is the less likely it is to be changed. Therefore, the older a file is, the less often Alex has to poll to insure that its local copy is up-to-date. To avoid excessive polling, Alex only guarantees that a file is never more than 10% out of date with its reported age. As an example, if a file is 1 month old, then alex will serve the file for up to three days ($10\% \times 30$ days = 3 days) before checking to see if it is still valid.

This is efficient, not only because FTP servers do not have to propagate file updates, but because it also avoids the necessity of modifying FTP servers to add more sophisticated invalidation techniques. In chapter we find that this cache-consistency scheme is applicable to the World Wide Web as well as to FTP. In the next section we examine Web proxies that extend the concepts of Alex to the World Wide Web by caching Web data just as Alex cached FTP data.

World Wide Web proxies

A Web proxy [25] acts as a gateway to the World Wide Web for a campus-sized network or corporation behind a firewall. The proxy accepts a request for a Web documents from a client, retrieves and caches the document, and then makes it available to its client. When another client wants the same document the proxy can serve it from its cache without having to re-request the same document.

The most popular Web browsers support proxies [29] and the HTTP protocol has provisions to help maintain cache consistency. If the proxy wishes to determine if its cached page is up-to-date, a lightweight protocol exists known as the *Conditional GET*. A browser may make a `get` request to the server that includes a timestamp, the *If-Modified-Since* field. If the page has been modified since that time, the server will re-send the page. Otherwise the server will respond with a *Not Modified* message.

Netscape, a company that sells a commercial web-proxy, claims that a 2 or 3 gigabyte Web proxy can support thousands of internal users and can provide a cache hit ratio as high as 65%. Results such as these indicate that web proxies can help alleviate some web scalability concerns, but as we saw above with Blaze, even optimal client caching still does not help distribute server load sufficiently. Web proxies also require human intervention to set up properly, and proxies must still satisfy cache misses from the primary host.

Next: Internet Caching Schemes **Up:** Related Work **Previous:** Related Work

---

James Gwertzman
Wed Apr 12 00:26:11 EDT 1995

Cache Consistency Mechanisms

There are several ways to maintain cache consistency of files placed in a cache: time-to-live fields, active invalidation protocols, and client polling. Time-to-live fields are efficient to set up, but are inaccurate. It is hard to tell in advance how long an object will be valid for; therefore, it is necessary to set the TTL field to a relatively short interval and reload the object frequently to avoid returning stale data. Client polling, as we saw with the Alex system [8], depends on the fact that the older a file is the less likely it is to change, and therefore the less often the client needs to check to see if its copy is stale.

The Alex protocol dynamically adjusts to minimize the amount of stale data returned, but any weak-consistency protocol inevitably results in clients receiving some stale data. We study its potential for cache consistency on the Web in chapter.

Invalidation protocols are required when loose consistency is not sufficient; many distributed file systems rely on invalidation protocols to insure that cached copies do not become stale. Invalidation protocols depend on the server keeping track of cached copies of its data; when a file changes the server notifies the caches that their copies are no longer valid.

Kurt Worrell, as part of his master's thesis [37], examined the use of TTL fields on the World Wide Web and determined through simulation that they are not as effective as an invalidation protocol that allows the primary host to inform caches when objects that they are caching become stale. We challenge this result in chapter.

One flaw with this work is its focus on invalidation to the exclusion of other loose consistency protocols like the Alex polling protocol. Invalidation protocols might be efficient in terms of network bandwidth, but they have several non-trivial costs. For one thing, servers must be rewritten to take advantage of server-directed invalidation. Client-directed protocols such as the Alex polling protocol or TTL fields do not require server modification. For another thing, servers must keep track of where their objects are currently cached. This could be a prohibitive limit to a system's scalability unless caches are organized hierarchically, such that the server only has to keep track of the highest level caches. Worrell assumes this model to make his invalidation protocols feasible, but in doing so presents another potential obstacle to scalability: he also assumes that the *inclusion* principle holds. The inclusion principle states that any time an object is located in a lower level cache, it must also reside in any higher level caches in the hierarchy.

This makes sense for the purposes of an invalidation protocol, since invalidation messages

only need to be passed on to the highest level caches, but could lead to an inefficient use of disk space. All objects cached on lower levels must also be cached on higher levels, but due to the nature of hierarchical caches, this means that one high level cache might potentially need to cache the equivalent of hundreds of low level caches. Otherwise pages may be kicked out of a low level cache simply because a high level cache is full, and it may be impossible to fully utilize space available in low level caches due to a lack of space in a high level cache.

Once again, a client-directed cache consistency mechanism removes this requirement since there is no need to organize caches hierarchically. This also means that caches can be created and taken down quite easily since there is no reason to fit them into a hierarchical tree. Client-directed cache consistency also removes one of the most serious error-modes of invalidation protocols: if a cache is temporarily unavailable then the server must continue trying to reach it; with no other mechanism for invalidation available the cache will not know to invalidate the object unless it hears from the server.

Harvest

The Harvest system [5] was designed to solve the problem of resource location, but also includes a hierarchical caching subsystem. This caching subsystem functions like a Web proxy in that clients request Web objects through the local Harvest cache. If this query results in a cache miss, the local cache in turn queries each of its neighbors, its parent, and the object's primary host. Each neighbor cache and parent returns with a ``hit'' or ``miss'', and if the home site is running an SNTP echo server it returns a ``hit'' as well. The object is then requested from whichever site returns a ``hit'' fastest. If all caches missed, and the primary host is slower than the parent cache, then the local cache will request the object through the parent cache. Otherwise the local cache will request the item from the object's primary host.

This system is still primitive; caches must be configured manually with information about parents and neighbors set in a configuration file. On the other hand, their results show that the Harvest cache is twice as fast at returning data as other widely available Web proxies for most requests.

Next: Resource location **Up:** Related Work **Previous:** Internet Caching Schemes

James Gwertzman

Wed Apr 12 00:26:11 EDT 1995

1.1 Message Handling Overview

1.1.1 Same time and different time communication

Message handling systems (MHS systems) are systems for people to send messages to each other. (Some messages are sometimes sent to or from applications without human intervention, but the main functionality is messaging between people.) The table below gives an overview of some major categories of MHS systems:

	All participants are active at the same time.	Different participants can read and write at different times.
Non-computer equivalent	*Face-to-face meeting, telephone call.*	*Letter, newspaper.*
Two or very few participants.	Simple chat systems, instant messaging systems.	Ordinary e-mail.
Communication within groups of people.	More advanced chat systems.	E-mail with mailing lists. Usenet News, forum systems.

There is no absolute sharp limit. Ordinary e-mail with multiple recipients is often used to communicate in small groups. And some chat systems contain archiving, so that those not present can read the discussion at a later time. There is, however, certain design features which are more important for the different categories.

For *same-time* messaging, it is important that everything said or written is immediately shown to the other participants. Some e-mail systems have an alert facility, where the recipient is reminded by a beep, whenever a new message arrives. Such e-mail systems can be used for almost same-time communication. But most textual same-time communication is done through specially designed chat or instant messaging systems. There are usually separate panes for reading and writing, sometimes the full discussion scrolls in a combined window for all participants, sometimes (only with very few participants) there is a separate window showing what each participant has written.

For *different-time* messaging, the set of messages which has been read and not read at a certain time will be different for each participant. People usually want to read only unseen messages, but they also want to be able to go back and reread old messages. Because of this, such systems have data bases, where messages are stored, and this data base knows what each user has seen and not seen and provides facilities for rapidly scanning unseen messages and searching for old messages. This can either be a personal data base with copies of all the messages for each recipient, or joint data bases, where the text of the messages need only be stored once, even if they have multiple recipients.

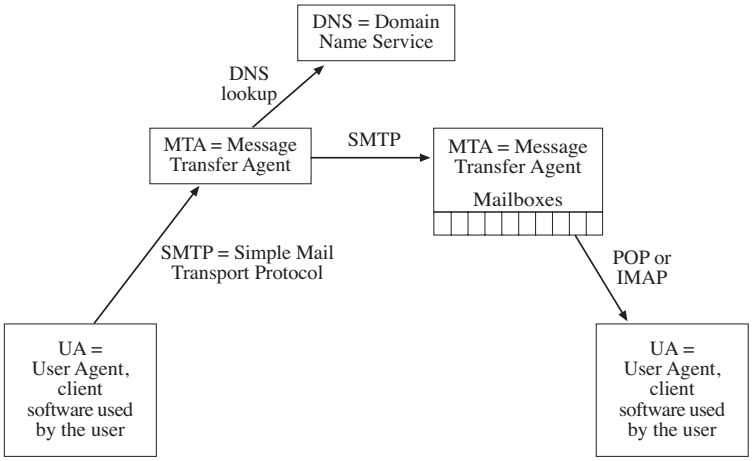
With *different-time* messaging, some people will get more messages than they can comfortably read. It is easier for them to handle the message load, if the software gives them facilities to organize the message data base. Typical such organization tools sort the messages from each discussion group into a separate folder, and sort messages in a thread (messages

which are direct or indirect replies to each other) together in some ways. The application layer standards have special facilities to support such functions.

With *same-time messaging*, it is important that messages are transmitted fast. The store-and-forward methods used by ordinary e-mail is too slow, but some systems for same-time messaging use their own kind of fast store-and-forward transmission. Particular for same-time messaging is that, since messages are written and arrive asynchronously, there can be dead times when the recipient receives nothing. It is, therefore, useful to have a reminder system, perhaps using a sound signal, when a new message arrives. There is also a tendency, in same-time messaging, that the on-going conversation handles more than one topic at the same time. People unaccustomed to chat systems react against this, and sometimes requre “one subject at a time”, just as at face-to-face meetings following an agenda. However, one subject at a time is not the most efficient use of the time. Because people write slower than they read, and because dead times occurs when you are waiting for a reply to a message, the time is used more efficiently if you allow multiple topics in parallel.

Two main variants of the user interface are common in chat systems. One variant has a separate window for each participant, showing what that person has written. The other has a single sequential window all participants of the same chat channel, listing all messages in cronological order. A *channel* is a group of people who are discussing with each other in a chat system.

1.2 Protocols Used for Message Transport



Many application layer protocols are commonly used for message transport. Here are some of the most commonly used:

- The DNS is used to look up the mail host for the recipient, and then in a second step to look up the IP address of the MTAs of the receiving MTA.
- SMTP is used to transport the message to each handling MTA.
- POP or IMAP is used to download the message from the last MTA to the recipient UA.
- MSGFMT (previously known as RFC822) and MIME are used to format the content of a message.
- NNTP for distribution of Usenet News Articles.

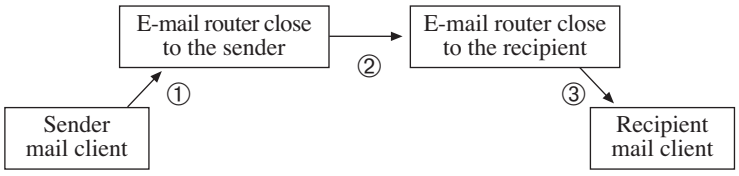
In addition to this, messages may also be conveyed through *usenet news*, which has its own

set of protocols, partially based on the email protocols, but often with their own variations, and messages may be conveyed through *instant messaging* protocols.

Note that the receiving user's mailbox is usually on the last MTA, and not in the personal computer of the recipient. This is important, because personal computers are not always online. Messages can thus be delivered to a mailboxes, even if the actual recipients are travelling, ill, playing games or otherwise not connected to the Internet.

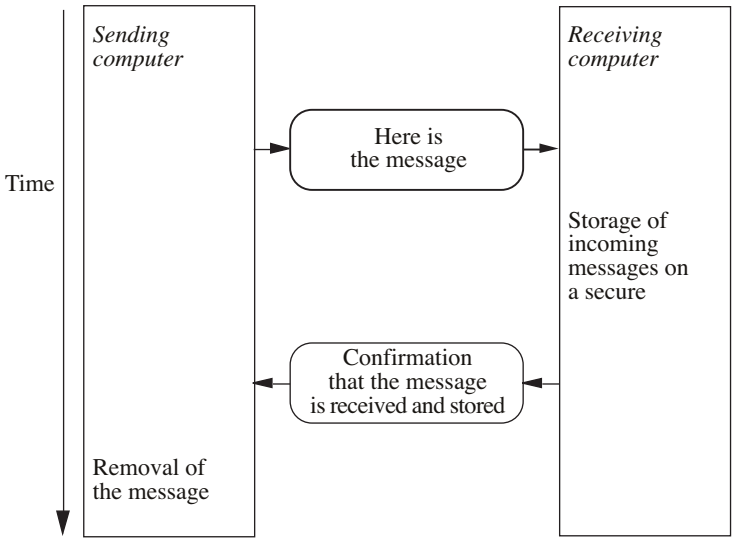
1.3 E-mail: Message Transport

Typical for e-mail and other non-simultaneous message systems is that the messages are copied, often several times, between different mail servers on their way from sender to recipient. The picture below shows a simple and common route which a message may take from sender to recipient:



When senders submit messages, their mail clients connect to nearby e-mail routers. The whole message is then copied ① from sender to mail router. When this copying is ready, this mail router takes over responsibility for delivering the message to its recipients:

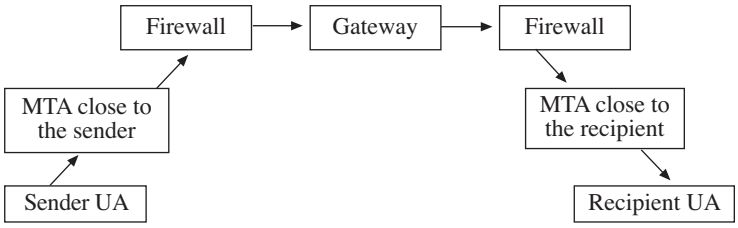
When the receiving computer confirms reciept of the message, the sending computer is no longer involved. In the second step, the mail router close to the sender connects to a mail router close to the recipient. Again, the whole message is copied ②, and responsibility is then transferred to the e-mail router close to the recipient. In the final step, the message is copied ③ to the recipient mail client.



In e-mail terminology, the client of the sender and the recipient are often called *User Agents* (UA), and the e-mail routers are often called *Message Transfer Agents* (MTA).

The transport of a message does not always pass exactly two MTAs as in the figure above.

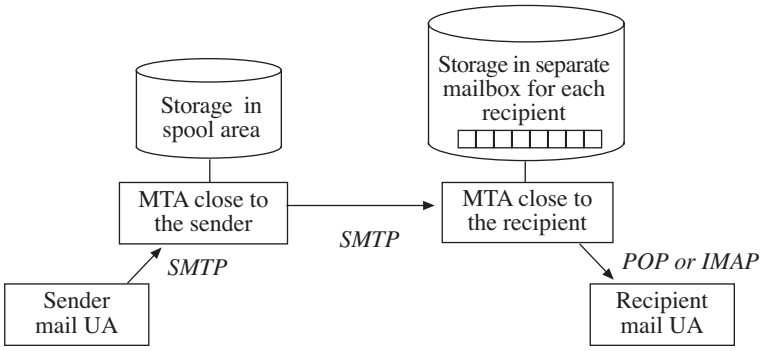
If both sender and recipient work at the same place, only one MTA may be needed. And in a complex networking situation, more than two MTAs can be used, like in the figure below:



A firewall is a security device, and a typical function in it may be to scan for viruses in the message. A gateway is a device which moves messages between two networks using different e-mail protocols, for example from an Internet mail network to an X.400 mail network. (X.400 is a mail protocol developed by the Telecom companies, it is today only used in certain local areas.)

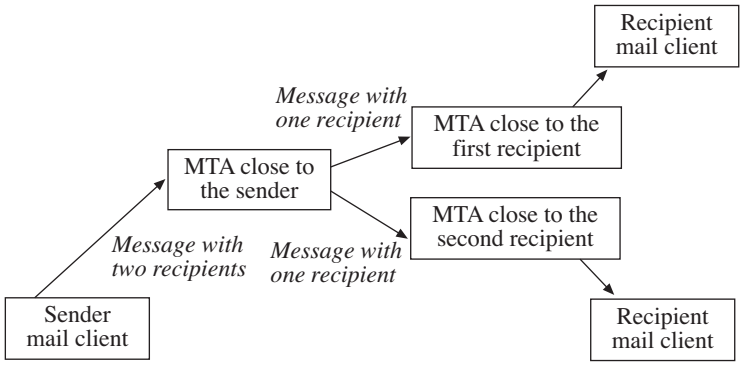
When a message is passed between two agents, a protocol is needed. The most common protocol for passing messages between agents i SMTP. SMTP is almost universally used for all mail transport except the transport from the last MTA to the recipient UA. The reason for this is that SMTP is mainly designed for sender-controlled transmission. In every step between two agents, it is the sending agent which controls the transmission and decides when and where to send a message. This is not suitable for recipient UAs, since they often reside on personal computers, which are not always connected to the network and willing to receive messages. Thus, for the final step from the last MTA to the recipient UA, a protocol is needed where the recipient has more control of when to receive a message. The two most common protocols for this are POP and IMAP. In some cases, the last MTA and the recipient UA run on the same computer or on two computers sharing a common file storage. In that case, the protocol for the last step can just be simply file retrieval. Another important difference between SMTP and the last-step delivery protocols is that identification (usually with a password) is needed to stop messages being retrieved by other than its intended recipients.

An MTA which receives a message will usually store it in a so-called spool area. This is a file storage area for temporarily storing files, and where there is some process, the spool process, which goes through the spool areas and forwards messages from them. An exception to this is the last MTA before delivery to the final recipient UA. That MTA usually stores messages in mailboxes, separate areas for each recipient. This means that when a UA connects to an MTA to download mail, all mail for that particular UA is stored in a special place and can easily be located. See the figure below:

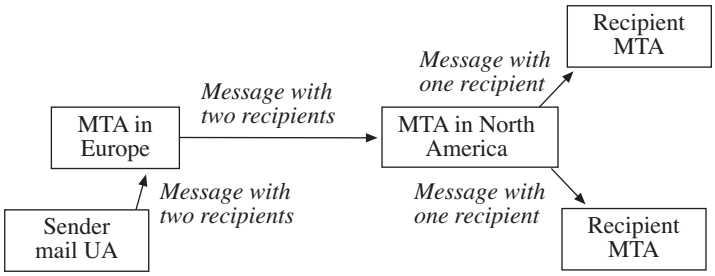


If a message is sent with more than one recipient, the message may be split into separate

copies by one or more MTAs, as shown in the figure below:



The advantage with splitting is that the same message need not be transported more than once before the splitting. This can also be used to reduce transport costs across large distances. If a sender in Europe sends a message to two or more recipients in North America, only one copy might be copied across the expensive Atlantic cables as shown in the figure below:



flame

Here I am flamin giving my person opinion on a controversial issu Other experts hav different opinion this.

A problem with this, however, is that most MTAs are not willing to handle mail, unless either the recipient or the sender is local to the MTA. Thus, the saving shown above requires an agreement with the MTA which splits the message after transport across the Atlantic. This was not always so. In the beginning of the 1990-s, most MTAs were willing to forward mail for any recipient. The reason why this was abolished in the middle of the 1990-s was that spammers used this feature to get foreign MTAs to help them split mail to millions of recipients. Some so-called experts claimed that spamming could be stopped by forbidding splitting of mail by other than the MTA of the sender or the recipient. They enforced their view by implementing a program which scanned all MTAs everywhere, checking that they did not allow foreign splitting, and sending angry letters to non-conforming MTA administrators (postmasters) threatening to stop receiving mail from them unless they stopped splitting. This is an interesting example of how the Internet is regulated in dubious ways by pseudo-police-authorities. Spamming could be counteracted more efficiently using other methods than this. In fact, there is no proof that spamming has in any way been reduced by this method.

1.4 Use of the DNS for mail transport

E-mail has its own special way of using the DNS. The DNS can actually be seen as two different data bases, one for e-mail and one for all other usage. The special records for e-mail are called MX (mail exchange) records.

If an MTA receives a message for a recipient with the e-mail address mary@bar.net, it will connect to the DNS and ask for the MX record for the domain “bar.net”. It may then be told that the MX record for “bar.net” refers to the domain name “mail.bar.net”. It will then make a second ordinary (not MX) DNS lookup to find the IP number for “mail.bar.net,” and it will then forward the message to the SMTP port at this IP number.

A special facility of the DNS, when used for e-mail, is that a lookup for the MX record for a domain (like “bar.net” in the example) may return more than one result. It may return a list of domains. The reason for this is to increase reliability, by having more than one MTA which is willing to handle mail for a particular recipient. An organisation may have one primary mail MTA and another secondary mail MTA to use if the primary mail MTA is not in operation. A common usage of this is that if each department in an organisation has different MTAs, the organisation may have a common master MTA, which is used if one of the department MTAs is out of operation.

There is, however, a risk with this technique. Suppose that an organisation **foo.bar** has three different MTAs, **mail1.foo.bar**, **mail2.foo.bar** and **mail3.foo.bar**. Suppose that a message arrives at **mail3.foo.bar**, but that the malbox of the recipient actually resides in **mail1.foo.bar**. The MTA at **mail3.foo.bar** will then use the DNS to look up the MX record for foo.bar, and may get a list of **mail1.foo.bar**, **mail2.foo.bar** and **mail3.foo.bar**. It might then transfer the message to **mail2.foo.bar**. The MTA at **mail2.foo.bar** will make a new MX record look up, get the same result, and transfer the message back to **mail3.foo.bar**. In this way, an infinite loop may occur with the message being sent back and forward between **mail2.foo.bar** and **mail3.foo.bar**.

To avoid such loops, the DNS has a special facility. It stores, with every MX record, a priority. In the example above, a DNS lookup for **foo.bar** may return the three MTA names **mail1.foo.bar**, **mail2.foo.bar** and **mail3.foo.bar**. But they might each have a priority, for example:

mail1.foo.bar	0
mail2.foo.bar	10
mail3.foo.bar	20

There is then a rule, that an MTA should never transport a message from an MTA with a lower MX record priority value to an MTA with a higher MX record priority value. Thus, **mail2.foo.bar** can never transport the message to **mail3.foo.bar**, and no loop will occur.

There is then a rule, that an MTA should never transport a message from an MTA with a lower MX record priority value to an MTA with a higher MX record priority value. Thus, **mail2.foo.bar** can never transport the message to **mail3.foo.bar**, and no loop will occur.

1.5 Mailbox names

Senders and recipients of e-mail are called *mailboxes*. Note that the recipient mailbox is a box on the last MTA, from which the recipient can download messages. The mailbox is not a storage area on the recipient computer, unless the recipient computer and the last MTA is the same computer (See the figure on page 3).

A mailbox in SMTP usually has two names, a user-friendly name and an e-mail address. For example, my user-friendly name is “Jacob Palme” and my e-mail address is “jpalme@dsv.su.se”. The two names are often written in sequence in the format shown by these examples:

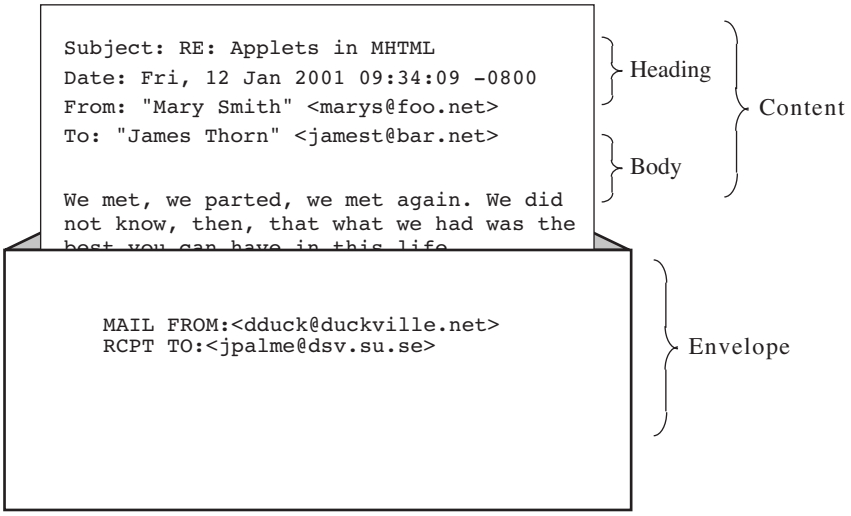
Jacob Palme <jpalme@dsv.su.se>

"Nils B. Frändén" <nfranden@foo.bar>

If the user-friendly name contains certain special characters, including periods followed by spaces and non-latin letters, it must be enclosed in double quotes as shown above. However, this rule is often broken, so it is advisable to write a mail program so that it accepts user-friendly names without double-quotes.

As is shown by the examples above, the user-friendly name is not globally unique – two different people may have the same user-friendly name. The e-mail address must be globally unique (otherwise it would not be possible to know to which mailbox to deliver the mail), it consists of a local mailbox name, the “@” character and a globally unique domain name. (Some servers accept mail with an incomplete domain name, if the address is a local user to the domain served by the server, but this practice is not recommended. It is much better to always use globally unique e-mail addresses, because then the address of a user will be the same independently of where it is sent from.)

1.6 Message format



A message being sent from sender to recipient consists of an *envelope* and a *content*. The content consists of a *message heading* and a *body*.

The *envelope* contains information used or generated during the transport of the message, such as the e-mail address of the recipient and the e-mail address of the sender. The sender's address is needed to be able to send a non-delivery notification if the message cannot be delivered to the recipient.

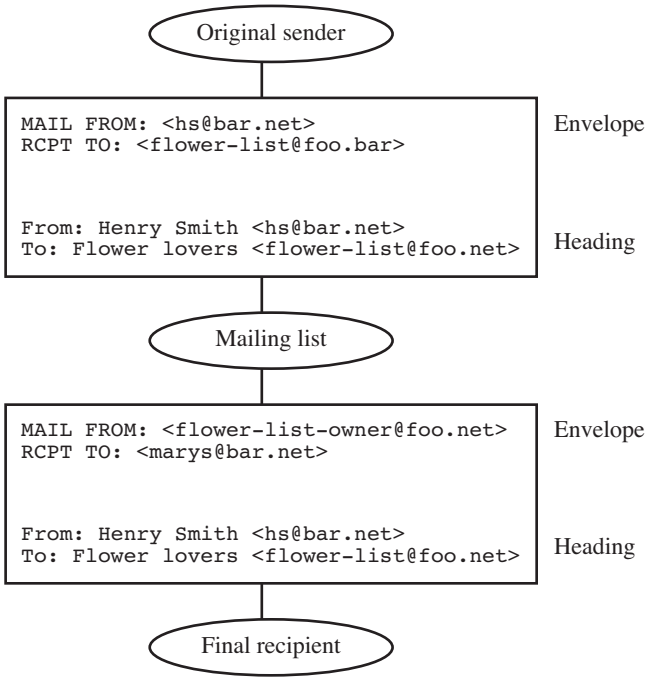
The *content* contains the information which is sent from the original author to the final recipient. It is, in theory, not meant to be changed during transmission. In reality, it is sometimes changed. Certain information which logically belongs to the envelope is in reality

put in the heading. And the content is sometimes transformed, for example between two different encoding formats, such as 8BIT and Quoted-Printable.

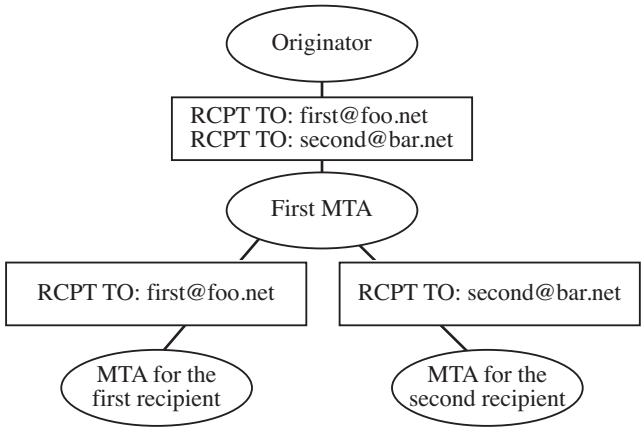
The *body* contains any kind of information, such a text, pictures, sound, video and/or file attachments.

Some information is specified both on the envelope and in the heading, such as the sender and the recipient. This information is, however, not always the same. The envelope shows information pertaining to a particular transmission or step in transmission. The heading shows information created by the sender and intended for the final recipient.

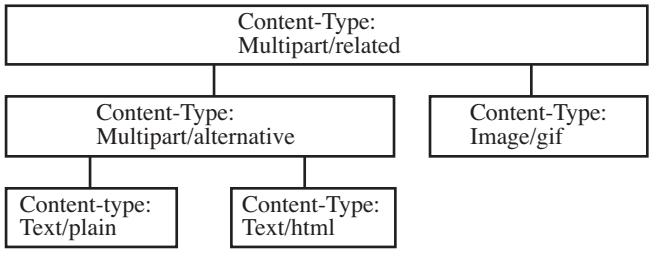
Example 1: If a person sends a message to a mailing list, and the mailing list forwards the message to the members of the list, then the heading contains the original sender in the "From:" field and the name of the mailing list in the "To:" field. The envelope contains different information when transporting the message from the original sender to the mailing list, and when transporting the message from the mailing list to the final recipients, as is shown by the figure below:



Example 2: If a message is split into copies to be delivered to different recipients, then each copy will on the envelope contain the address of the recipient for this copy. A message may, for example, first be sent with two recipient from the sender UA to the first MTA. The first MTA may then split the message into two copies for the two recipients. The envelope will then in the second step of transmission contain only the recipient for this transmission, see the picture below:



The body can consist of several body parts, for example a text and an attachment or a HTML-formatted text and embedded pictures. Each body part can also consist of several body parts. The figure below shows a complex message with body parts inside body parts; this particular structure is very common:



When the body consists of several parts, each part has its own *content-heading*. A content-heading contains information about the content part, such as *content-type* and *encoding method*. The set of header fields allowed in a content-heading is a subset of the set of headers allowed in a message-heading. All the header fields allowed in a content-heading have names beginning with the string Content- such as “Content-Type”, “Content-Language”, etc.

1.7 E-mail: Message Heading

1.7.1 The Internet Message Format

The format of the messages themselves (corresponding to the P2/P22 protocols of X.400) is specified in RFC 822 [17], RFC 1036 [19], RFC 1123 [20], and the multi-purpose Internet mail extension (MIME) standards [21, 22].

The figure below shows an example of a message heading according to the RFC 822 standard. Note that what is shown is not some legible print-out of the heading, but the actual contents, since RFC 822 uses readable IA5 characters in the heading.

```
From comp.protocols.iso.x400-outbound-request@ics.uci.edu Wed
      Nov  4 04:16 MET  1992
Received:   from sunic.sunet.se by heron.dafa.se (16.6/SiteCap-3.0)
      id AA09605; Wed, 4 Nov 92 04:16:24 +0100
Received:   from ics.uci.edu by sunic.sunet.se (5.65c8/1.28)
      id AA25221; Wed, 4 Nov 1992 04:15:03 +0100
```

```
Received:   from ics.uci.edu by q2.ics.uci.edu id aa14794; 3 Nov 92
      13:47 PST
Received:   from USENET by q2.ics.uci.edu id aa14789; 3 Nov 92 13:46
      PST
From: "Paul.Rarey" <Paul.Rarey@ssf-sys.dhl.com>
Subject:    Re: X400 address
Message-ID: <921103134349.16483@maverick.ssf-sys.DHL.COM>
Encoding:   35 TEXT, 12 TEXT SIGNATURE
X-Mailer:   Poste 2.0
Date: 3 Nov 92 21:46:13 GMT
To:   mhsnews@ics.uci.edu,
      Piet Beertema <mcvax!cwi.nl!piet@uunet.uu.net>
cc:   ifip65@ics.uci.edu

... Text of the message ...
```

Example of an RFC 822 message heading.

User names in RFC822 can have both a user-friendly name and an e-mail address (see page 8).

RFC 822 specifies a heading format, which is intended to be readable for both humans and computers. It is used for communication between the sender and the recipient and is not used during transmission. The most important fields in Internet mail headings are:

Received: Contains a trace of the hosts through which a message has passed. It is mainly used when investigating problems with the mail transfer. Such a trace could be used for loop control (as in X.400) but in practice usually is not. Many UA softwares suppress these lines when showing the header to their users.

From: The e-mail address of the author of the message.

To:, Cc: and Bcc: The recipient(s). Note that this is not the recipient(s) to be used when delivering the mail (those are given in the SMTP protocol). This is information to be read by the recipient(s) or their mail program. It can even contain names which are not proper e-mail addresses.

Message-ID: A globally unique identity code for the current message, which can be used in the “Reply-To” field to eliminate duplicates of the same message arriving via different routes.

Reply-To: Used to indicate that personal replies to a message are to be sent to someone else than the name in the “From” field. Note that the name in this field should not be used for “group replies,” that is, replies intended for the group of recipients or the mailing list that received the In-Reply-To message.

Followup-To: Similar to “Reply-To” but used for group replies. The value must be a usenet newsgroup name, not an e-mail address. (This is not part of RFC 822 it is defined in the Usenet News standard, RFC 1036 [19].)

In-Reply-To: Used to coordinate replies with the original message. An intelligent mail software can use this field to help the user find the original message of a reply.

References: Similar to “In-Reply-To.” It is used mainly in Usenet News to indicate references between postings to newsgroups.

Date: The date, time, and time zone when the message was sent.

Subject:: A title line describing the topic of the message.

The RFC 822 heading often contains additional fields not defined in the RFC 822 standard. Some of them may be defined in other standards, while others are not standardized at all. In the heading abpve, the fields “Encoding” and “X-Mailer” are not part of RFC 822.

The RFC 822 heading ends with a blank line, which marks the beginning of the text of the message.

1.7.2 References between Messages and Global Message IDs

There is often a need to let a message refer to previously sent messages: for example, in replies. It is an important facility for a recipient of a reply to be able to easily ask his electronic mail system for the message to which the reply replies. It is also useful for users of electronic mail systems to be able to browse through whole conversations. In order to be able to transfer a reply link between two messages which are sent at different times, it is useful if every message has a globally unique identifying code. This is a code which no other message has or is going to get.

E-mail has some kinds of identifying codes in the heading and sometimes also on the envelope. The code on the envelope, however, is shortlived, and long-lived references between messages are handled using the identifying code in the heading. A globally unique identifying code in the heading is usually produced by combining two subfields. One subfield is a valid electronic mail name (usually of the originator, but some other valid name can also be used) plus an identifying code which is unique relative to this electronic mail name. Since electronic mail names and addresses must be unique (otherwise it would not be possible to deliver a message to its recipient), this will then produce a globally unique identifying code for the message itself. Often, these parts are separated with an asterisk. Example: **200107081059*john@foo.bar**. The part before the asterisk is usually an encoding of the date and time when the message was sent in some format. Another variant is to use a sequential numbering of all messages sent by john@foo.bar. A disadvantage with a sequential number is that the last used number need not be stored, and if this stored number is lost, duplicate Message-IDs will not be created. Therefore, the date/time method is usually better.

E-mail has three different kinds of references between messages, all indicated by use of the Message-ID:

In-Reply-To: for ordinary replies.

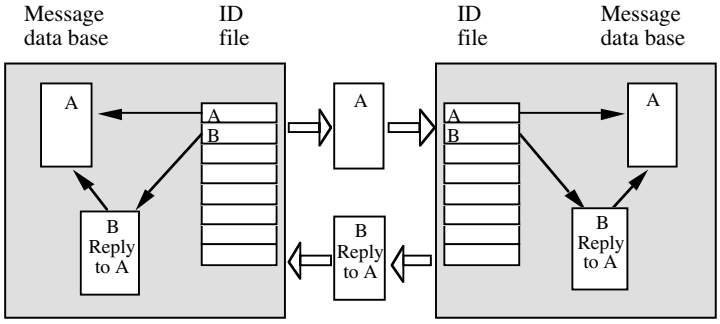
References: for other kind of references.

Supersedes: when the new message is a replacement of an old message, for example, a new version of a text under development or containing a copy of the original message with an error corrected. Supersedes is not yet standardized, but is commonly used in Usenet News, but not so often in e-mail.

A message cannot have an *In-Reply-To* reference to more than one previous message, while

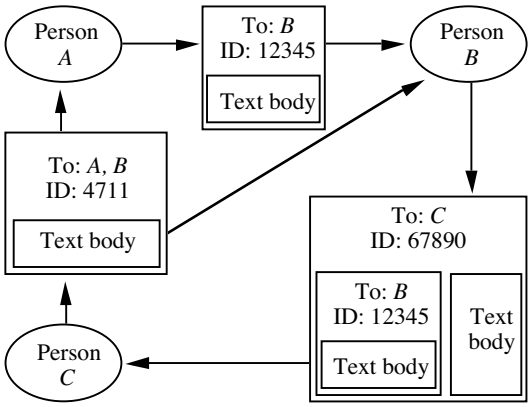
the number of messages it is *References:* or *Supersedes:* can be more than one. A consequence of this is that a conversation which is created using the *In-Reply-To* field will always have a tree structure, something which is not true for conversations created by *References:* and *Supersedes:*. Another consequence is that two different conversations can be merged into one if the *References:* and *Supersedes:* fields are used to create conversations. Such merging is not possible if only the *In-Reply-To* field is used.

The figure below shows how the Message-ID can be used to connect a reply to the original message, using a data base of Message-ID-s with pointers to the messages they refer to in the mailbox data base of both the sender and the recipient.



Use of Message-ID to correlate replies with the original message.

When you create a reply link on a message which contains forwarded body parts, you should carefully consider to which heading the reply link should refer. This is illustrated in the figure below.



Problem with references to the wrong Message-ID.

Person *A* writes a message to person *B*. *B* forwards the message to *C*, using the forwarding mechanism of including the whole text of the original message as a body part of the forwarded message. *C* then writes a reply to the original message and sends the reply to both *A* and *B*. It is very important for *C* to think carefully of whether to use the ID of the original message (12345, in the example) or the ID of the forwarding message (67890, in the example). This is important because *A* has never seen the forwarding message. This means that a reference in the reply, saying that it is a reply to 67890, will not be understood by *A*. Good message systems should be designed to protect users from such mistakes.

Of course, the intention of *C* may be to reply also or mainly to the text body which *B*

added when the message was forwarded to *C*. But if that is the case, *C* should either send the reply only to *B* or forward the whole message on to *A*, since a reply sent to *A* on the forwarded body part in 67890 will otherwise not be understood by *A*.

A person will often get several copies of the same message, forwarded by different users and distribution lists. It is an important service to the user that his electronic mail system be able to correlate these copies, so that the person will not have to read the texts more than once and so that the person can find all comments, appendices, and recipients of the message. Such correlation should also be done when the recipient receives the same message both directly and included as forwarded body parts in other messages. The globally unique Message-ID is the code which is used to provide such correlations. Note the use of the word "correlate," not the word "merge." Since all copies of the same message are not always identical, information can be destroyed if they are merged carelessly.

1.8 E-mail: Message Body

1.8.1 Types in Internet Mail

1.8.1.1 MIME Introduction

The Internet mail standards used before 1992 could handle only text in the 7-bit ASCII alphabet and did not allow the additional body-part type facilities available in X.400. However, in 1992, the Internet mail standards added facilities in an extension called MIME. MIME is defined in RFC 2045 [21] to RFC 2049 [22]. A good tutorial on MIME is given in [25]. There is an FAQ[2] on MIME [44] which includes a list of known MIME implementations.

MIME allows Internet mail to contain the following

- Multiple objects in one message.
- Unlimited line length and message length.
- Character sets other than IA5 (7-bit ASCII). In particular, the ISO 8859-1 and the ISO 10646 (Unicode) character set is important.
- Binary and application-specific files.
- Diagrams, pictures, voice, video, and multimedia in messages.
- References to files, which can be retrieved automatically through the net when the recipient wants to read the message. The contents of the file is thus not transported with the message.

---

2 FAQ (frequently asked question) is a document containing answers to often asked question in Internet discussion groups. There are many FAQs on various topics, and they often contain a lot of useful information.

With MIME, it is possible to have several body parts representing the same information in different formats. The recipient or his client software can then choose the version which it is best capable of displaying. For example, a recipient with an IA5-terminal can see a message in this format, while a recipient with an ISO 8859-1 terminal can see a better version of the message.

1.8.1.2 MIME Encoding of Data

There was a problem when MIME was defined: it was essential that existing e-mail systems be able to handle MIME messages in a reasonable way without modification. However, some previous Internet e-mail systems could not handle messages with 8-bit characters in the text or with unlimited line length, binary information, etc. in the body.

Because of this, MIME encodes the new body parts in a format which looks like 7-bit ASCII to old mail software. This is not a very neat solution, but it was necessary since no extension facility for body parts was included in the old Internet mail standards. MIME encodes additional information in either of two ways so that it looks like 7-bit ASCII. These two methods are called *base64* and *quoted-printable*.

Base64 uses four ASCII characters to represent three 8-bit bytes. The 24 bits of the three 8-bit bytes are split into four groups of 6 bits, and each such bit is encoded in a character set which has 64 different values. These 64 values are those characters in 7-bit ASCII which are least likely to be corrupted. (This is more fully described in the chapter about coding.)

Quoted-Printable represents those bytes, which have an exact correspondences in 7-bit ASCII, with their 7-bit ASCII values. Other characters are encoded as an "=" character, followed by two digits with the hexadecimal value of the byte. An exception is the "=" character itself, which is coded as "=3D."

1.8.1.3 Example of a Complete MIME-Encoded Message

The figure below shows a complete example of a MIME message as it is transmitted (not as it is shown to the user). This message contains two body parts. Each of the body parts contains the same text in ISO 8859-1.

```
Test message containing 8-bit characters.
AE=Ä, OE=Ö.
```

The two lines above are transmitted in both body parts in the MIME message below, but the text is encoded as quoted-printable in the first body part and as base64 in the second body part.

```
Return-Path: <jpalme@ester.dsv.su.se>
Date: Sun, 26 Sep 1993 18:49:01 +0100 (MET)
From: Jacob Palme DSV <jpalme@dsv.su.se>
Subject: A MIME message
To: Lars Enderin <larse@dialog.se>
Message-Id: <3.85.9309261822.A27024-0200000@ester>
Mime-Version: 1.0
Content-Type: MULTIPART/ALTERNATIVE; BOUNDARY="1430317162"

--1430317162
Content-Type: TEXT/PLAIN; CHARSET=ISO 8859-1
Content-Transfer-Encoding: QUOTED-PRINTABLE

Test message containing 8-bit characters.
AE=3D=80, OE=3D=85.
```

```
--1430317162
Content-Type: TEXT/PLAIN; CHARSET=ISO 8859-1
Content-Transfer-Encoding: BASE64

VGVzdCBtZXNzYWdlIGNvbnRhaW5pbmcgOC1iaXQgY2hhcmFjdGVycy4KQUU9
gCwgT0U9hS4K
--1430317162--
```

A complete MIME message.

As can be seen from the example in the figure above, quoted-printable has an advantage over base64 because a recipient whose mail system is not MIME compatible will still be able to understand much of the content, especially the ordinary English text. In the example, quoted-printable is also shorter: only 61 characters are required compared to 73 characters for base64. Base64 is, however, more suitable for pure binary text, such as a graphic bit image or an encoded sound.

In addition to base64 and quoted-printable, MIME also allows the encodings *7bit*, *8bit* and *binary*. These are not really encodings, since they represent uncoded data. Because of this, 8bit and binary should not be transmitted via mailers which are not MIME-compatible. To transport such data uncoded, extensions to the SMTP standard for message transport are also needed.

1.8.1.4 MIME Heading Extensions

MIME defines the following extended fields to the RFC 822 heading:

`Content-Type:`	Specifies the type and subtype of the data in the body.
`Content-Type: TEXT`	Specifies textual information in several different character sets.
`Content-Type:`	Specifies that the body contains more than one body part. Each body part can have additional heading fields at the beginning of the body part.
`Content-Type:APPLICATION`	For application-specific or binary data. Of special importance is `APPLICATION/POSTSCRIPT`.
`Content-Type: MESSAGE`	For an encapsulated mail message.
`Content-Type: IMAGE`	For a bitmapped picture using, for example, the graphics interchange format (GIF) or joint photographic experts group (JPEG) formats.
`Content-Type: AUDIO`	For sound. (Note that the word “voice” is used in X.400. However, X.400 voice is probably not intended to be restricted only to spoken sounds.)
`Content-Type: VIDEO`	For video using, for example, the motion picture experts group (MPEG) format. The video may, but need not, contain a sound track.
`Content-Transfer-Encoding:`	Specifies how the data is encoded. Encoding type may also be given in the content-type heading field.
`MIME-version:`	To indicate that MIME is used and which MIME version is used.
`Content-ID:`	ID code for the content. Can be used to allow one body part to refer to another body part in another message or for references between the body parts in a message.
`Content-Description:`	Contains a textual description of the content.

IANA (Internet Assigned Numbers Authority), the internet global registration authority, maintains a register of MIME content types. IANA requires a publicly available description of

the new format, or, alternatively, that public domain viewers for the new format are available. A consequence of this is that some commercial companies have been unwilling to to register the formats of their products. In such cases, a Content-Type EIGHTBIT has to be used instead. The disadvantage with this is that the receiving client software cannot automatically start a viewer for the new format when messages in that format arrives.

1.8.1.5 Multipart messages in MIME

The MULTIPART attribute in a MIME heading specifies that the body contains more than one body part. Each body part can have additional heading fields at the beginning of the body part, including recursive use of the multipart attribute to include MULTIPARTs as parts in higher level MULTIPARTs.

Of special interest are

`MULTIPART/MIXED`	for several parts of different types;
`MULTIPART/ALTERNATIVE`	for the same information in more than one encoding;
`MULTIPART/PARALLEL`	as mixed, but to be viewed at the same time (for example, voice in parallel with a drawing); and
`MULTIPART/DIGEST`	to indicate that this is a collection of several messages written by different authors.
`MULTIPART/RELATED`	several files, which are to be combined into one document or set of related documents. Used, for example, to send HTML, SGML or XML with pictures, applets, etc., as separate files.

1.8.1.6 The Multipart/Related Content Type

The Multipart/related content type is designed when you are sending several files, which are related by URL-links. It is used, for example, to send HTML, SGML and XML with embedded pictures or applets as separate files.

Each file is a separate body parts. Each body part is labelled by either Content-ID or Content-Location. The URL referring to the body part from another body part, is of the URL type "cid:" to refer to a Content-ID, or can be any kind of URL (absolute or relative) to refer to a Content-Location with the same content.

Example (abbreviated):

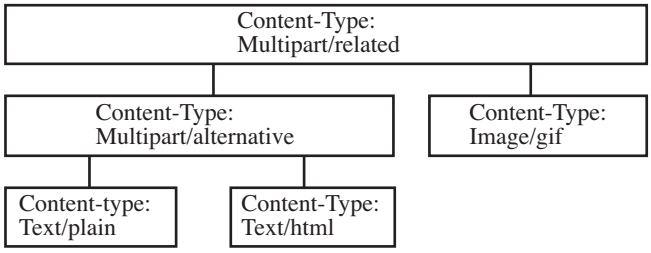
`Content-type: Multipart/related`	The compound object of the HTML text and the embedded message.

`Content-Type: Text/html`
 	
`<IMG SRC="cid:1*foo@bar.net>`
 	
`<IMG SRC="picture.gif">`
	The main text in HTL format.
	
	Link to an embedded image using a "cid:" type URL.
	Link to an embedded image using a relative URL.

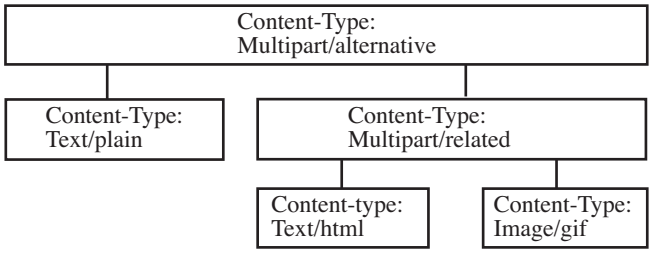
`Content-Type: Image/gif` `Content-ID: 1*foo@bar.net`	The first embedded image, identified by a Content-ID.
`Content-Type: Image/gif` `Content-Location: picture.gif`	The second embedded image, identified by a Content-Locdation URL.

Since some mailers do not support this, messages are usually sent using multipart/alternative, with plain text in the first branch and HTML in the second branch. This can be done in two ways:

1.8.1.6.1.1 With the multipart/alternative inside the multipart/related:



1.8.1.6.1.2 With the multipart/alternative outside the multipart/related:



Some mailers send messages using each of these methods, so a good mailer will have to be able to receive messages in both formats.

1.8.1.7 The Message Content Type

The MESSAGE content type is used to forward a message within another message. For example, you can forward a message you have received, as one body part, with your own comment as another body part. MESSAGE has the following subtypes:

- MESSAGE/RFC822 when you encapsulate a normal Internet mail message, formatted according to the RFC822 or MIME formats.
- MESSAGE/PARTIAL if this is one part of a large message, which you have split into several smaller messages during transport. Information is in this format included to help the receiving mail client merge the parts together into a complete message again.
- MESSAGE/EXTERNAL-BODY; ACCESS-TYPE= to send, instead of a message, just a reference to where the message can be retrieved. ACCESS-TYPE can indicate various ways of retrieving the actual content, like various variants of FTP etc.

1.8.1.8 Mime Heading Character Sets

MIME allows extended character sets also in message headings. The encoding is rather ugly and some implementations do not support it.

1.8.2 In-line or attachments

In the user interface, a MIME message can either be displayed as a sequence of text passages and images, or MIME can be used to send attachments to a primary textual message. The attachments are then not viewed unless the user explicitly asks for them. In practice, the latter

usage has been most common. RFC 1806 specifies an e-mail header which can be used to indicate, separately for each body parts, whether this body part is to be shown inline or as an attachment. The format is `Content-Disposition:` followed by either the word `inline` or the word `attachment`. A `file name` parameter can also be included, indicating a recommended file name if the attachment is stored in a file.

Note that if the `Content-Type: Multipart/related` is used to send a document with inline objects like images, then the techniques described in the MHTML standard should be used. `Content-Disposition` can be used, but should be ignored by mailers which understand `Content-Type: Multipart/related`.

1.8.3 Sending HTML in e-mail

MIME was planned in order to allow messages to contain pictures, formatted text, sound, etc. In practice, MIME has mostly been used to send such information as attachments. The success of the World Wide Web (WWW) means that the HTML (Hypertext Markup Language) has become a very successful format for documents with formatted text and inline graphics. There are two ways of sending HTML in e-mail:

(1) Use the message/external-body MIME body part, which means that the e-mail message only contains the URL (Uniform Resource Locator) of the actual message, and the recipient mailer will then use this URL to retreieve the message from the net just like viewing an ordinary web page.

(2) Send the HTML text within the message, using the Content-Type Text/HTML.

The HTML format often uses more than one file to convey a message. For example, graphics, frames and applets are usually stored in separate files, which are combined by the web browser when displaying the document. Thus, there is a need to send HTML as a set of related body parts which together form the document. This is done using the `Content-Type: Multipart/related`. The main message will then reference the other parts either through their Content-IDs or through their URLs. If URLs are used, the other body parts are given a heading `Content-Location` which indicates the URL of this body part.

The MHTML standard is not only useful for sending HTML in e-mail. It can also be used for archiving of web pages. By archiving web pages in the MHTML format, all information in a web page can be archived in a single file, instead of separare archiving of the HTML text and the in-line images and other in-line data.

1.9 SMTP – The Protocol for Message Transport

The Internet protocol for message transport is SMTP (except for the transmission from the final MTA to the recipient UA, where other protocols are usually used). SMTP is a protocol for the transmission of a message from the original UA to the first MTA and from one MTA to another MTA. SMTP is thus a protocol for transmission one step on the way from originator to recipient. However, some information, such as the e-mail address of the sender,

is usually conveyed from one SMTP step to the next SMTP step.

Since SMTP is used to control the transmission, it specifies most of the envelope information. Some envelope information is however specified in the message heading.

SMTP is a session-oriented protocol. By that is meant that a connection is established between two agents, and then a series of interactions takes place between the two agents. Here is an example of a simple SMTP session:

Sending agent command	*Responding agent response*
Opens a connection to port 25 of the receiving host	Listens for connections on port 25, acknowledges new connections
`HELO dsv.su.se`	`250 nexor.co.uk`
`MAIL FROM: <jpalme@dsv.su.se>`	`250 OK`
`RCPT TO: <j.onions@nexor.co.uk>`	`250 OK`
`RCPT TO: <seb@nexor.co.uk>`	`250 OK`
`DATA`	`354 Start mail input`
... the lines of text in the message ...	
`.`	`250 OK`
`QUIT`	`221 nexor.co.uk service closing`

An SMTP session can include the transmission of more than one message. In that case, a new MAIL FROM command comes at the place of the QUIT command in the figure above.

The basic SMTP commands are:

"HELO" opens an SMTP session.

"MAIL FROM:" starts the sending of a new message by specifying the sender. Its value is an e-mail address enclosed in angle brackets.

"RCPT TO:" is repeated once for every recipient. In the original SMTP protocol, an acknowledgement from the server (the 250 response code) was required after every recipient, before the next recipient address could be sent. SMTP has later on been extended with an optional facility to send several recipients in sequence.

"QUIT" indicates the end of an SMTP session, a server receiving a QUIT command should close the connection.

"DATA" signifies the start of the body of the message. The server response "354" indicates that the server expects more. After DATA, the body is sent. According to the original SMTP standard, the body must be a number of text lines, and finished by a line containing a single period. Line breaks in SMTP must always be a carriage return character plus a linefeed in sequence (CRLF) and the end of the body is thus signified by CRLF, a single period, and a second CRLF. In order to allow the sending of messages containing lines with only a single period, all lines which start with a period have an extra period sent at the beginning of the line.

Thus, the following text:	**Is in SMTP sent as:**

`The next sentence will start with a period:`
`.line starting with a period`
`The next sentence will contain a single period`
`.`
	`The next sentence will start with a period`
	`..line starting with a period`
	`The next sentence will contain a single period`
	`..`
	`.`

In addition to the HELO, MAIL FROM:, RCPT TO:, DATA and QUIT command, the

original SMTP protocol contained some additional commands. Most of these are either not used or not used very much today. Examples of these commands are:

VRFY to ask the server to verify that it has a mailbox with a certain e-mail address. A server may also give a positive response if it is willing to deliver a message to this recipient, even though the recipient mailbox is not on the same host as the server. The parameter to VRFY can be an abbreviated name, the response should be the full e-mail address, if the abbreviation is non-ambiguous.

TURN is a command which within a single SMTP session switches the roles between sender and recipient, so that the agent which started the SMTP session can receive mail from the MTA it connected to. TURN is useful if connection times are long, such as for dial-up modem connections. The increasing use of fixed lines has reduced the needs for the TURN command.

The most common digits in the SMTP reply codes are:

1.9.1.1.1.1 First digit:

1 Positive preliminary (not used in SMTP)
2 Success
3 Ready but requires additional info
4 Transient failure
5 Permanent negative

1.9.1.1.1.2 Second digit:

0 Syntax (error)
1 Information requested in reply
2 Transport service problem
5 Application-specific problem

1.9.1.1.1.3 Examples of reply codes to the MAIL FROM command:

250 Originator accepted
452 Out of local storage
500 Command syntax error

A number of proposals for extensions to SMTP have been developed. These extensions allow the sending of delivery-report requests, binary and 8-bit data, and an SMTP sender can check if a very large message can be received before sending it. These extensions are only to be used by extended SMTP servers, so there is also a protocol for two SMTP servers to query each other's capabilities at the start of an SMTP session. This protocol thus provides an extension mechanism to SMTP and is called ESTMP (Extended Simple Mail Transfer Protocol). The method of protocol extension used by ESTMP is that the server gives the client a list of which ESTMP extensions it supports, and the client must then only use those extensions which the server says that it supports. This method has many advantages compared to the method of indicating protocol level (like HTTP version 0.9 or 1.0 or 1.1) because a server can choose to provide certain extensions without having to support other less useful extensions.

In plain ESTMP, a session is started by the client sending the HELO command. A client which supports ESTMP send the EHLO command instead of the HELO command. The EHLO command only indicates that the client understands ESTMP, not that the client is capable of handling any particular ESMTP extension.

The table below lists the most important ESMTP extensions:

Service extension	Keyword	Parameters	Verb	RFC
Send	SEND	none	SEND	821, 1869
Send or Mail	SOML	none	SOML	821, 1869
Send and Mail	SAML	none	SAML	821, 1869
Expand	EXPN	none	EXPN	821, 1869
Help	HELP	none	HELP	821, 1869
Turn	TURN	none	TURN	821, 1869
Pipelining	PIPELINING	none	none	1854
Message size declaration	SIZE	adds optional parameter size-value ::= 1*20DIGIT no of octets	none	1870
Checkpoint/ restart	CHECKPOINT	adds optional parameter TRANSID to MAIL FROM command	none	1845
Large and binary MIME messages	CHUNKING	BDAT is used instead of DATA, and takes as parameter packet length and last packet indication	BDAT	1830
8bit-MIMEtransport	8BITMIME	adds optional parameter BODY to MAIL FROM, values 7BIT and 8BITMIME	none	1652
Delivery Status Notification Extension	DSN	adds optional parameters NOTIFY and ORCPT to RCPT command and RET and ENVID to the MAIL command	none	1891, 1892, 1894

Here are three different examples of ESTMP capability negotiations:

(1)
Only mandatory SMTP commands provided

```
S: <wait for connection on TCP port 25>
C: <open connection to server>
S: 220 dbc.mtview.ca.us SMTP service ready
C: EHLO ymir.claremont.edu
S: 250 dbc.mtview.ca.us says hello
```

(2)

```
S: <wait for connection on TCP port 25>
C: <open connection to server>
S: 220 dbc.mtview.ca.us SMTP service ready
C: EHLO ymir.claremont.edu
S: 250-dbc.mtview.ca.us says hello
```

Basic optional services: EXPN, HELP;

```
S: 250-EXPN
S: 250-HELP
```

Standard service extension: 8BITMIME;

```
S: 250-8BITMIME
```

Unregistered services: XONE and XVBR

```
S: 250-XONE
S: 250 XVRB
```

(3)
ESTMP not supported

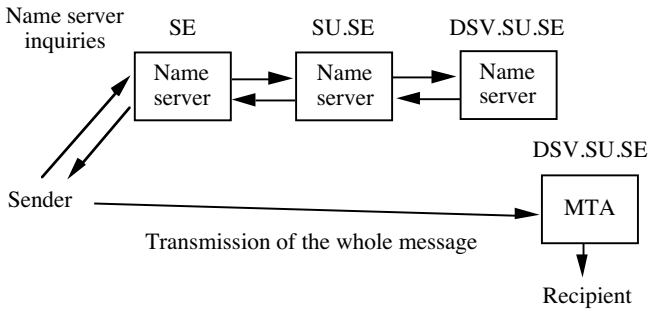
```
S: wait for connection on TCP port 25>
C: <open connection to server>
S: 220 dbc.mtview.ca.us SMTP service ready
C: EHLO ymir.claremont.edu
S: 500 Command not recognized: EHLO
```

1.9.2 SMTP command pipelining

SMTP often requires many interactions back and forward between client and server. For example, when a message is to be sent to many recipients, the client is expected to send a RCPT TO for each recipient and wait for the response from the server before sending the next RCPT TO. This will make the protocol slow, since interactions across large network distances often cause a delay of one or more seconds.

In practice, many SMTP clients ignore this rule, and send everything without waiting for reply codes, and then gets all the reply codes asynchronously. This is not correct, but usually works, since TCP will provide storage on the net of the data sent in advance. An ESMTP extension specified in RFC 1854 allows a server to indicate that it is capable ot such pipelining.

1.9.3 Use of domain addresses for routing



Name server look-up followed by direct transmission.

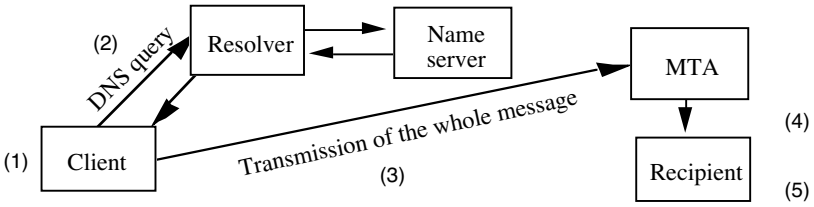
The figure above shows how a series of name servers that know host addresses in domains successively closer to the recipient can be used to find the network address of the recipient MTA host. The sender can then transmit directly to the recipient MTA. Of course, the name servers for SU.SE and DSV.SU.SE might be combined into one name server. And the name server for SE can cache names and addresses, so that after a second inquiry for the address of DSV.SU.SE, the SE name server can answer directly without forwarding the query to SU.SE. The technique of keeping names passing through a name server is called *caching*. Cached addresses should only be kept for a limited time, since they may otherwise become out-of-date.

The technique described in the figure above, where each successive name server connects to another name server, is called *chaining*. Another also commonly used technique is called *referral*. With referral, the searcher will successively connect to a series of name servers until the right one is found.

1.9.4 Routing and Use of Name Servers in Internet

The methods for routing mail messages using name servers in Internet is described in RFC

974 [18], RFC 1101 [23], RFC 1123 [20], and RFC 1348 [24].



Use of name servers for Internet mail routing.

E-mail handling in the Internet is usually done in the following stages. (The numbers refers to numbers in the figure above.)

(1) The originator edits and submits his message for mailing.

(2) For each recipient, the name server facility is used to find the IP-address of the host which is most suitable for delivering mail to the recipient.

(3) The mail is then forwarded to the hosts serving each recipient. The SMTP protocol is used for this.

(4) The host described in (3) can be the host closest to the recipient or an intermediate host which forwards the message, for example, a gateway to another network with a different mail standard. If it is an intermediate host, this host will then use SMTP or some other protocol (like X.400 P1) to forward the message to the next host in a chain leading to the recipient. If the message is sent to a distribution list, the host handling the list will expand the list and forward the message to the members of the list. Often, incoming messages from the Internet to an Intranet are first routed to a firewall process, which will check the messages for viruses, before they are re-routed internally.

(5) Finally, the recipient reads the mail. The mail system of the recipient can use the machine-readable information in the message heading to aid the user in providing facilities like:

- (a) Finding the message to which the current message is a reply.
- (b) Finding messages from a certain sender, or messages which arrived via a certain distribution list.
- (c) Finding messages written between certain dates.

Name servers for routing in the internet are called DNS servers. A DNS server takes as input an Internet domain address, such as EIES2.NJIT.EDU. There are many DNS name servers, all responsible for only part of the DNS tree. This means that sometimes the first name server contacted by a resolver may not have the information requested. The information can then be found by using either *chaining* or *referral*. The Internet DNS allows both chaining and referral. Every DNS server must support referral. Support for chaining is optional.

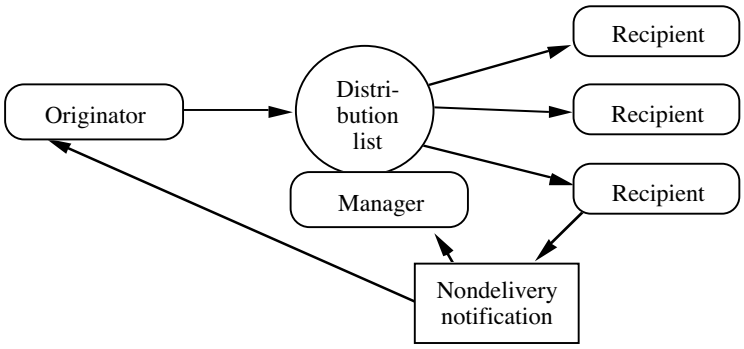
1.9.5 Delivery Status Notifications

A good electronic mail system should always (unless the sender explicitly relinquishes this requirement) inform the sender if a message cannot be delivered. But sometimes a fault, such as a disk crash, can cause messages to disappear without any notification to the sender (such situations are often called *black holes*). However, if the sender has requested a delivery

notification, the fact that it has not arrived within a reasonable time can indicate to the sender that the message has been lost, even when no nondelivery notification appears.

It is, of course, advantageous to the sender if his mail software automatically recognizes incoming delivery and nondelivery notifications and handles them in suitable ways. The sender may prefer not to be informed every time such a notification arrives, but to store them so that later the sender can ask the system to provide a report about which recipients have received their messages. Notifications usually indicate that a message has reached the mailbox of the recipient, but some systems also provide notifications when a message has been read by the recipient, or, rather, when it has been shown on the screen of the recipient.

Nondelivery notifications are often difficult to understand if you are not an expert on electronic mail protocols. Getting a nondelivery notification from some recipient to whom you never sent any message can be especially confusing. What happens is that you send a message to a mailing list, and there is a delivery problem in delivering the message to one of the members of this list. For large mailing lists, such delivery reports should, of course be sent to the manager of the list, not to the originator of the message. However, it is not uncommon for mailers to mistakenly send the nondelivery notifications to the originator.



Sending error reports to mailing list manager or to the originator

The Internet mail delivery notification functionally is called Delivery Status Notifications (DSNs), and it is specified in four RFCs:

RFC 1891: SMTP service extension, 31 pages

RFC 1892: Multipart/report content-type, 4 pages

RFC 1893: Enhanced status codes, 15 pages

RFC 1894: Delivery Status Notification format, 31 pages

Requests for delivery status notifications is sent via SMTP, using optional parameters to the RCPT TO and MAIL FROM commands:

SMTP command	Optional parameter	Description
`RCPT TO`	`NOTIFY`	Values: NEVER = do not send any DSNs. SUCCESS = send a DSN if the message was successfully delivered to the recipient mailbox FAILURE = send a DSN if the message could not be delivered to the recipient mailbox DELAY = indicate willingness to accept notifications if delivery is delayed but may succeed later on
`RCPT TO`	`ORCPT`	Original sender-specified recipient address (needed to allow recpient of notifications to correlate notifications with original recipients, since some gateways will rewrite the recipient e-mail addresses before delivery)

SMTP command	Optional parameter	Description
MAIL FROM	RET	Values: FULL = return full text of the message with the DSN HDRS = return only headers of the message with the DSN If neither FULL nor HDRS is indicated, this means that no return of either headers or body is requested.
MAIL FROM	ENVID	Transaction ID to be returned with notification, so that the sender can find which message sending caused the failture. (Message-ID cannot be used for this, since sometimes the same message is sent more than once with the same Message-ID).

While the requests for DSNs are sent via SMTP, the DSNs themselves are specially formatted MIME messages. A DSN is sent as a MIME message with the Content-Type Multipart/Report. A Multipart/Report consists of two mandatory and one optional part:

Part 1 (mandatory) is a human readable message explaining in natural language the cause of the error. This part is mainly intended for recipients whose e-mail clients do not recognize the special format of Part 2 of a DSN.

Part 2 (mandatory) contains a machine parsable account of the reported event, with a special MIME type called Message/Delivery-status.

Part 3 (optional) contains the original message either full or only its headers.

Here is an example of a complete delivery status report:

```
Date: Thu, 7 Jul 1994 17:16:05 -0400
From: Mail Delivery Subsystem <MAILER-DAEMON@CS.UTK.EDU>
Message-Id: <199407072116.RAA14128@CS.UTK.EDU>
Subject: Returned mail: Cannot send message for 5 days
To: <owner-info-mime@cs.utk.edu>
MIME-Version: 1.0
Content-Type: multipart/report; report-type=delivery-status;
      boundary="RAA14128.773615765/CS.UTK.EDU"

--RAA14128.773615765/CS.UTK.EDU Part 1, in “human-readable” (?)
format:

The original message was received at Sat, 2 Jul 1994 17:10:28
-0400
from root@localhost

   ----- The following addresses had delivery problems -----
<louisl@larry.slip.umd.edu>  (unrecoverable error)

   ----- Transcript of session follows -----
<louisl@larry.slip.umd.edu>... Deferred: Connection timed out
      with larry.slip.umd.edu.
Message could not be delivered for 5 days
Message will be deleted from queue

--RAA14128.773615765/CS.UTK.EDU Part 2, in machine-parsable
format:
content-type: message/delivery-status

Reporting-MTA: dns; cs.utk.edu

Original-Recipient: rfc822;louisl@larry.slip.umd.edu
Final-Recipient: rfc822;louisl@larry.slip.umd.edu
Action: failed
Status: 4.0.0
```

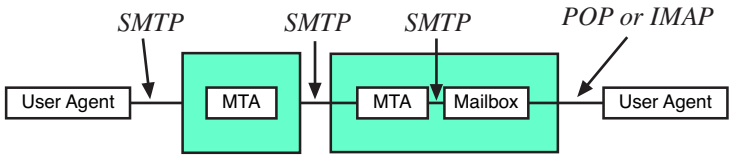
```
Diagnostic-Code: smtp; 426 connection timed out
Last-Attempt-Date: Thu, 7 Jul 1994 17:15:49 -0400

--RAA14128.773615765/CS.UTK.EDU
content-type: message/rfc822    Part 3, return of original
message:

[original message goes here]
--RAA14128.773615765/CS.UTK.EDU--
```

Here are some of the fields which can occur in a Delivery Status Report:

`per-message-fields =`	Fields which apply to the whole message.
`[ original-envelope-id-field CRLF ]`	Envelope identifier from request.
`reporting-mta-field CRLF`	MTA which attempted to perform the delivery or relay.
`[ dsn-gateway-field CRLF ]`	Name of gateway which transformed foreign delivery report.
`[ received-from-mta-field CRLF ]`	MTA from which the message was received.
`[ arrival-date-field CRLF ]`	Arrival date to reporting MTA.
`*( extension-field CRLF )`	
`per-recipient-fields =`	Fields which apply to only one recipient at a time.
`[ original-recipient-field CRLF ]`	Original recipient when sent.
`final-recipient-field CRLF`	Final recipient to whom delivery status is reported.
`action-field CRLF`	failed, delyaed, delivered, relayed (to non-DSA environment), expanded.
`status-field CRLF`	Status code (RFC 1893) (DIGIT "." 1*DIGIT "." 1*3DIGIT
`[ remote-mta-field CRLF ]`	Name of MTA which reported to reporting MTA.
`[ diagnostic-code-field CRLF ]`	Sometimes less preciste diagnostic code from remote MTA.
`[ last-attempt-date-field CRLF ]`	Time of last delivery attempt.
`[ final-log-id-field CRLF ]`	Log entry in final MTA logs.
`[ will-retry-until-field CRLF ]`	Time when delivery attempts will stop.
`*( extension-field CRLF )`	

1.10 Message delivery

When a user runs an e-mail client package on his personal computer, this client needs a protocol to talk to a server, corresponding to the P3 and P7 protocols in X.400. The model behind these protocols, like in X.400, is that mail messages are stored in a server, to be downloaded to the e-mail client at request of the user, as is shown by this picture:



Model behind the POP and IMAP protocols.

In the Internet, the two most important such protocols are:

- Post Office Protocol (POP) [9, 12], a protocol for fast downloading of mail to client software, where the client stores and handles the mail in the personal computer, corresponding to P3 in X.400.
- Interactive Mail Access Protocol (IMAP) [8], a protocol for cases where the user wants to store his messages in the server, and wants to be able to manipulate this storage from client software on his personal computer. IMAP is a more complex protocol than POP.

Commands in POP and IMAP are textual strings, just like commands in SMTP. Here is a list of the most important commands in POP:

USER	Client identifies mailbox to be downloaded
PASS	Password
STAT	Get number of messages and size of mailbox
LIST N	Return size of message N
LAST	Get highest message number accessed
RETR N	Retrieve a full message
TOP N M	Retrieve only headers and the first N lines
DELE N	Delete message
QUIT	Release service
NOOP	See if POP server is functioning
RPOP	Insecure authentication

IMAP is a more sophisticated protocol than POP. In IMAP, a server can send messages to the client without a request from the client, and several transactions between server and client can go on in parallel. This can be used to reduce the wait time for the users. Each message in an IMAP mailbox has a set of properties, which can be retrieved one or more than one at a time. Examples of properties are a seen flag and a deleted flag on the message. In IMAP, to delete a message you first set the delete flag on the message, and then perform the expunge command. IMAP also has a capability for searching for messages in the mailbox stored in the server.

1.11 Mailing lists

1.11.1 Expansion of nested mailing lists

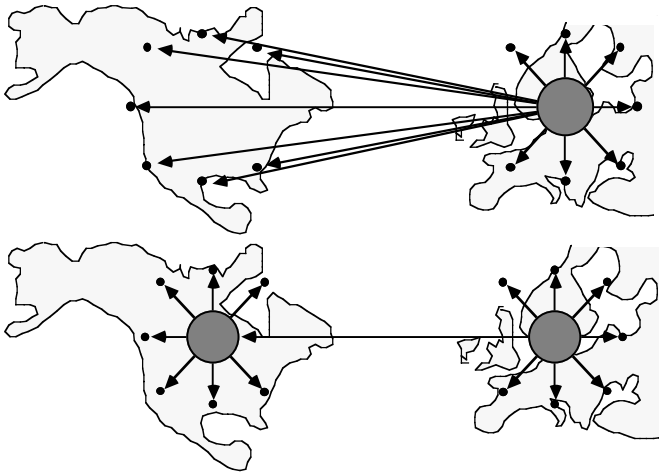
Nested mailing lists occur when one list is a member of another list. If, for example, list B is a member of list A, then a message sent to list A will be distributed to all members of both list A and B. A message sent directly to list B, however, will not reach the members of list A, unless A also is a member of B.

There are two techniques for handling mailing lists:

- *Sender UA expansion*. The mailbox software (user agent software) of the sender finds the list of the members of the list and sends it directly to them. If the lists are nested, the sender UA will successively and recursively find the lists of members of the sublists, all the way to the final recipient. With this method, the message will thus be converted to an ordinary multirecipient list. The message will also have ordinary users, and not lists, as recipients, before the message leaves the sender UA .

- *Expansion at the list location*. The sender's UA sends the list to the list-expansion agent or to the domain which is responsible for the list. The list is then expanded at this location. Expansion means the replacement, on the envelope (not in the message heading) of the name of the list with the names of the members of the list. With nested lists, the message is then forwarded to the domains of the sublists, which perform the secondary expansion, and so on if there are sublists to the sublists.

Consider a person who is a member two lists. With the second method, this person will probably receive two copies of the same message. With the first method, such duplicates can be eliminated during the expansion. In spite of this, expansion at the list location (the second method) is most common. With two nested lists, one for the European and one for the American members, the message need only be sent to one single recipient when crossing the Atlantic. If the whole list is expanded by the sending UA, at least 100 recipient have to be listed when crossing the Atlantic, which will make the transfer more expensive. See the figure below.



Advantage of using nested mailing lists (bottom) vs. nonnested (top).

Other advantages with expansion at the list is that it is easier, to support distributed control of the membership of a group—each sublist can have its own management. This can, of course, be a disadvantage when central control of the membership is required.

1.11.2 Loop control for nested mailing lists

If two lists are directly or indirectly members of each other, there is a risk that the same message will be looped back and forward indefinitely between the lists. There are several different techniques for avoiding this:

(1) Full expansion by the originating UA.

(2a) A trace list of all the mailing lists passed is put on the envelope of the message. A mailing list can then refuse to accept, incoming messages, that have the name of the mailing list itself as part of the trace list on the envelope of the message.

(2b) Using a variant of method (2a), each mailing list will instead refuse to send a message to another mailing list which is included in the trace list of the message.

(3) The registration system for mailing lists is designed in such a way that no list will ever be a member, directly or indirectly, of itself.

(4a) Each mailing list stores the message-IDs of all messages passing through the list. When the same message returns once more to the list, the list checks the message-ID of the message and stops the loop if a message with the same ID has already passed through the list. (Message-ID is also known under the term Message-ID.)

(4b) This is a variant of method (4a), where a checksum of the content of the message is used instead of the message-ID.

An advantage of method (2b) is that it saves some unnecessary transmission. However, method (2b) is not as reliable as method (2a), because the same electronic mail address can have different forms, and therefore the comparison used in method (2b) may not work. It is easier for a list expander to recognize a name which it itself puts on an envelope than a name which some other list expander puts on an envelope.

Comparing methods (4a) and (4b), method (4b), use of a checksum, has the advantage that it caters for systems where the message-ID is corrupted (something which is not unusual), but has as a disadvantage that two different messages may accidentally get the same checksum. A further problem with the message-ID is that two messages with the same message-ID may not always be identical. Finally, some mail systems do not generate globally unique message-IDs on the messages they produce.

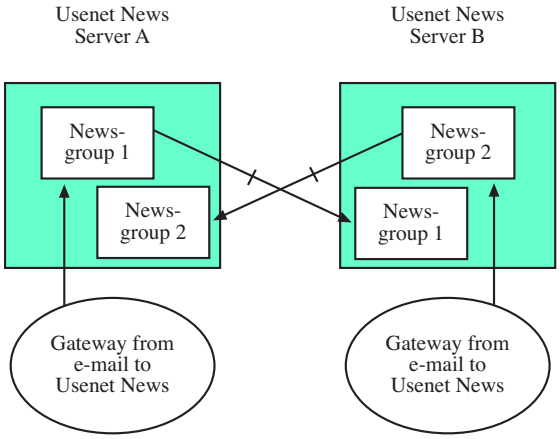
Method (4a) is often used by computer conferencing systems, sometimes combined with the other methods. For example, Usenet News mainly uses method (4a).

Programs that send messages through gateways from Internet e-mail to Usenet News will assign Message-IDs to messages lacking such IDs. This can cause the same message to get different Message-IDs by different gateways.

The Listserv software has very powerful methods for avoiding loops, primarily based on method (4b).

In practice, many existing mailing lists have no loop control mechanism except that the people who manage the list manually try to avoid producing loops.

Usenet News uses Message-IDs as its main loop control mechanism. This is, in Usenet News, implemented in a way which will sometimes cause multi-group messages to only appear in some of the groups, to which it was sent. The figure below explains how this can happen:



The original message is sent by e-mail to two e-mail mailing lists,which are both gatewayed from e-mail to Usenet News through two gateways. One of the gateways enters one of the mailing lists to Newsgroup 1 through Usenet News server A. The other gateway ensters the other mailing list to Newsgroup 2 through Usenet News server B.

When the Newsgroup 1 version of this message is to be moved from news server A to news server B, the loop control in news server B will refuce to accept the message, since it already has the message in Newsgroup 2. This means that subscribers to Newsgroup 1, but not Newsgroup 2, in Usenet News server B, will incorrectly never see this message. In the same way, subscribers to only Newsgroup 2 in News server A will never see the message.

I have suggested that the Usenet News standards should be modified to remove this problem. But the experts in IETF say that this is not possible, we will have to live with this problem. This is not the first time when the technical experts in IETF have refused user-friendly changes to standards.

1.11.3 Management of large mailing lists

Management of large mailing lists is not easy. The manager will every day receive nondelivery notifications for messages from the list which cannot be delivered, and requests for addition and removal of members from the list. It is sometimes difficult to identify the item in the list of members which such a notification or request refers to. Sometimes, they are actually members of a sub-mailing list. The problems of managing large mailing lists is more fully discussed in [3].

1.11.4 How You Become a Member of a Mailing list

1.11.4.1 The LISTSERV Way

Suppose you want to become a member of a mailing list, whose e-mail address is listname@host.bit.net. You then write an e-mail message addressed to listserv@host.bit.net and write in this message a single text line with the text:

```
SUB listname Your Own Name
```

```
SUB listname Your Own Name
```

For information about other commands you can give, such as how to unsubscribe from a list, download archived messages from the list, etc., write a message to the same recipient, listserv@host.bit.net, with the single word HELP in the text of your message. For more information about Listserv.

1.11.4.2 The -Request Way

Suppose you want to become a member of a mailing list, whose e-mail address is listname@host.bit.net. You then write a message to listname-request@host.bit.net and write in the text of the message something like “Please add me as a member of this mailing list.” The e-mail address with the name of a mailing list with “-request” added to it can also be used for other communication with the list manager.

1.11.4.3 Subscription through web pages

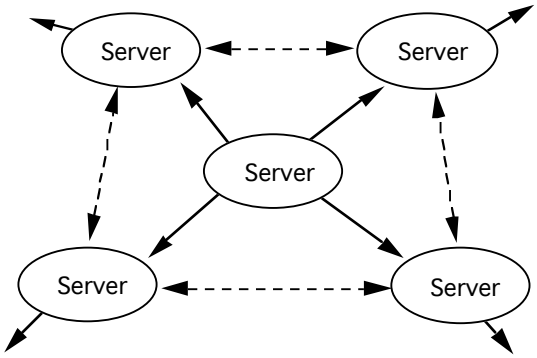
A third method is to subscribe to mailing lists through web pages. However, there is usually no very secure identification of a person who accesses a web page. Because of this, a convention has developed that if a person subscribes to a mailing list through a web page, then the list server will first send an e-mail to the user, asking him/her to confirm the subscription. Not until this confirmation is received, will the subscription be entered.

1.12 Usenet News

Usenet News is a distributed asynchronous forum system. Forums in Usenet News are called newsgroups, and messages are called articles.

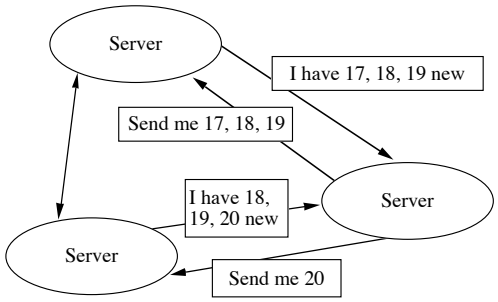
1.12.1 Distribution of news

The basic principle of Usenet News is that a local server handles most of the functionality. Usenet News standardizes two variants of the NNTP protocols: One for communication between adjacent servers, one for communication between a client and a server. Each server can download as much as it wants of what is available on any of the adjacent servers. Loop control is handled both by a trace list and a list of the Message-IDs of received messages stored by each server, so that the server can reject the same message coming back again. The procedure for distribution of news can be compared to pouring water onto a flat surface; the water flows out in all directions as shown in the figure below.



"Pouring water" principle of Usenet News distribution.

The figure below shows how new articles are forwarded from server to server in Usenet News. A server tells its adjacent servers which items it offers, the server requests those it has not already got via another route.



This figure shows how new articles are forwarded from server to server in Usenet News.

Information about a user, such as how much this user has seen, is stored in the client. The server need not even know which users are using it. There are many different user-interface softwares for Usenet News, Some of them, of course, do not provide all the available functions.

1.12.2 Cancel and Supersedes

In addition, Usenet News provides an interesting functionality which restricts communication to only those members of a newsgroup who work in the same organization or live in the same area or country. This functionality, however, is not used very much, and its existence is controversial, since it means that different users will get different views of the same newsgroup.

Usenet news has a *cancel* command, which can delete messages already sent out. Only the author of the cancelled message and the local newsserver administrator is allowed to cancel a message. Since, however, it is very easy to fake your identity, this command poses an obvious security risk, and the command is known to have been used to cancel messages for political reasons. The command is also used (not quite appropriate) by cancelbots, robots (= automatic programs) which cancel obvious spams by identifying messages with the same content sent to many disparate newsgroups. Usenet news also often has a Supersedes header field, which refers from a new message to an old message. This header usually cancels the old message.

There is also a *supersedes* header, which is used when a new message is sent to replace the previous version of this article. Supersedes is similar to cancel in that it causes a real deletion of the message being replaced. This is different from, for example, the obsoletes header of X.400, which only marks a new message as a replacement, but is not meant to cause the previous version to be physically deleted. Obsoltes thus is similar to the In-reply-To and References headers in e-mail.

The most important restriction of Usenet News is that closed groups are not well supported. The only kind of closed groups which are common in Usenet News are groups which are restricted to one or a selected set of news servers. Such groups will then be open to anyone with an account in these news servers, but closed to everyone else.

1.12.3 The Network News Transfer Protocol (NNTP)

The Usenet News protocol is called network news transfer protocol (NNTP) and is specified in RFC 977 [46]. The standard for the format of Usenet News articles is specified in RFC 1036 [19].

The table below lists the most common NNTP commands:

`article [<Message-ID> | <Number>]`	Return text of designated article. If no parameter is given, the next article is returned. The current article pointer is put at the fetched article.
`body [<Message-ID>| <Number>]`	As article, but only returns body
`group <newsgroup>`	Go to the designated newsgroup
`head [<Message-ID> | <Number>]`	As article, but only returns head
`help`	Lists available commands
`ihave <messageID>`	Informs the server of an available article. The server can then ask for the article or refuse it.
`last`	Sets current article pointer to last message available, return the number and Message-ID.
`list [active | newsgroups | distributions | schema]`	Returns a list of valid newsgroups in the format: *group last first*
`newgroups <yymmdd hhmmss> ["GMT"] [<distributions>]`	List newgroups created since a certain datetime. "distributions" can be e.g. *alt* to only get newsgroups in the *alt* category.
`newnews <newsgroups> <yymmdd hhmmss> ["GMT"] [<distributions>]`	List Message-ID of articles posted to one or more newsgroups after a specific time. *newsgroups* can be. e.g. net.*.unix to match more than one newsgroups. *distributions* checks for articles which also has this other newsgroup as recipient.
`next`	Current article pointer is advanced. Returns number and Message-ID of current article.
`post`	Submit a new article from a client.
`slave`	Tells the server that this is not a user client, it is a slave server. (May give priority treatment.)
`stat [<Message-ID> | <Number>]`	As article, but only returns Message-ID. Used to set the current article pointer.

Note that the same protocol, NNTP, is used for communication both between a client and a server, and between two servers, but that sometimes different commands are used. Thus, when a user client submits a new message, the `post` command is used, but when a server

sends a new message to another server, it usually uses `ihave` to give this information, and the receiving server then requests the message, if it has not already received it from another server.

Most of the message headers are the same in Usenet News and in e-mail, but there are some differences as shown by this table:

`Newsgroups:`	Comma-separated list of newsgroups to which this article belongs. Example of newsgroup format: alt.sex.fetisches.feet. *Should never occur in e-mail.* Use "Posted-To:" instead!
`Subject:`	Add four characters "`Re.` " for replies. Do not change subject in replies.
`Message-ID:`	Mandatory in Network News, and must be globally unique.
`Path:`	Path to reach the current system, e.g. abc.foo.net!xyz!localhost. E-mail path format also permitted. Compare to `Received:` and `Return-Path` in e-mail.
`Reply-To:`	In news: Where replies to the author should be sent. In e-mail: Ambiguous.
`Followup-To:`	Where replies to newsgroup(s) should be sent.
`Expires:`	Suggested expiration date.
`References:`	Message-ID-s of previous areticles in the same thread. Should always contain first and last article in thread. Compare to e-mail: Usually only immediately preceeding messages..
`Control:`	Not used in e-mail. Communication with servers. Body or subject contains command. Subject begins with "cmsg".
	`cancel` Delete physically a previously sent article.
	`ihave` Host telling another host of available new articles.
	`sendme` Host asking for articles from another host.
	`newgroup` Name of new group, plus optional word moderated.
	`rmgroup` Remove a newsgroup. Requires approved.
	`sendsys` Send the sys file, listing neighbours and newsgroups to be sent to each neighbour.
	`version` Version of software wanted in reply.
	`checkgroups` List of newsgroups and descriptions, used to check if list is correct.
`Distribution:`	Not used in e-mail. Limits distribution to certain geographical/organizational area. Example: Distribution: se, no.
`Organization:`	Of sender.
`Keywords:`	For filtering.
`Summary:`	Brief summary.
`Approved:`	Required for message to moderated group. Added by the moderator, contains his e-mail address. Also required for certain control messages.
`Lines:`	Count of lines of the message body.
`Xref:`	Numbers of this message in other newsgroups. Only for local usage in one server. Example: Xref: swnet.risk:456 swnet.sunet:897

There is a problem with the `newsgroups` header in a message sent via both Usenet News and e-mail. Different systems use this header in two different ways, in the mail version of such messages:

(a) To indicate that this message has also been sent via Usenet news to the indicated newsgroups.

(b) To indicate that this is a personal reply, sent only via e-mail, to a message posted on the indicated newsgroups.

Because of this problem, it is better to use the `Posted-To` header in e-mail to indicate that a message has also been sent to certain newsgroups, and e-mail recipients should ignore any `newsgroups` heading in an e-mail message.

MIME is not used as much in Usenet News as in e-mail. Instead of BASE64, an older method named UUENCODING is often used in Usenet News to include binary attachments. Also, because of message size restrictions, large attachments are very often split into several messages in Usenet News. This also occurs in e-mail, but is more frequent in Usenet News, since some Usenet News servers try to save space by not accepting articles above a certain size limit. Both MIME and Usenet News have methods of indicating how a client can automatically combine parts into a complete message or attachments.

In addition to NNTP, also other protocols are sometimes used for communication between Usenet news servers.

1.12.4 News Control in Usenet News

An important functionality in all asycnchornous (not same time) message systems is the news control functionaltity. This functionality will help the user find which messages are new and not yet read by this user. Most message systems handle this functionality together with the storage of messages. This means that if messages are stored in a server, the server also knows for each user, what that user has read andn ot read. And if messages are stored in the user's personal computer, then news information is also stored there.

Usenet News, however, does this in a different way. Messages are usually stored in the server, but the server does not know which messages each individual user has read and not read. The information on what a user has read is stored in a very compact format in the user agent software. The most common format for this file is the one shown by this example:

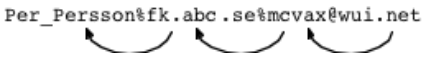
```
254-290
300-312
350-
```

The example above says that this user has unseen articles number 254-290, 300-312 and all articles numbered higher than 350. This format is very compact, because there are usually long sequences of articles which are read and unread. Storing a single bit for each message is then not optimal.

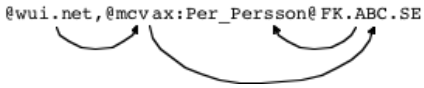
1.13 Relative addresses

Domain addresses are *absolute* addresses. An absolute address is the same address for a certain recipient, irrespective of where the message is sent from. Another type of address is called a *relative* address. A relative address indicates one or more relay stations on the route to the recipients. This would be roughly equivalent to indicating the postal mail address of a person as: “First to London, then from London by train to Nottingham, then by truck to the University, then by mailman from the University to the Computer Science department.”

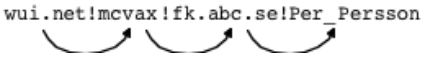
There are three commonly used formats for relative electronic mail addresses. All the three addresses in the figure below indicate the same relative route but in a different printed format



This format is sometimes used by gateways to the Internet, although it is not officially part of the Internet mail standards.



This format is sometimes used in the Internet mail standards.

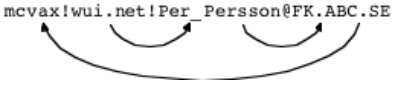


This format has been used much, and is still sometimes used, in Usenet News.

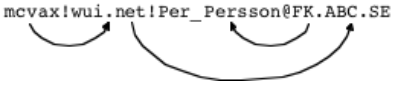
Combinations of several of these formats in one address may occur sometimes. For example, an address of the format:

```
MCVAX!FK.ABC.SE!Per_Persson@WUI
```

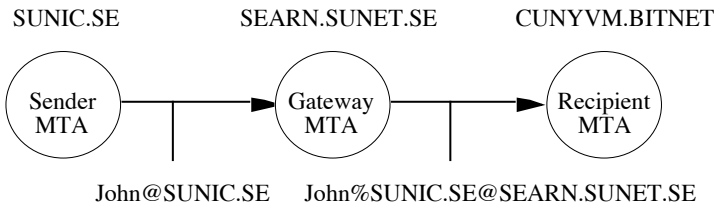
may according to some practices be interpreted as:



but may according to Usenet News practice be interpreted as:



The format using percent (%) signs is not part of the official Internet mail standards. This format will occur sometimes often when a message passes a gateway between two nets. The figure below shows how gateways can create relative addresses.



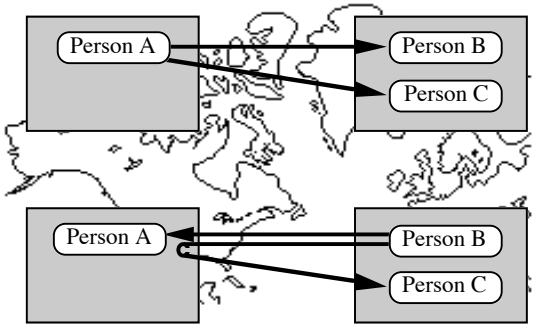
A message from a sender **John@sunic.se** passes a gateway from Internet to Bitnet. The address of the sender is given as **John@sunic.se** before the gateway, but the gateway translates this into **John%sunic.se@searn.sunet.se** when transmitting the message to Bitnet. Thus, the gateway takes the original sender address, replaces the @ with a %, and appends @ followed by the name of the gateway.

In Bitnet, when a reply is sent from the recipient to the sender, the reply is first sent to **John%sunic.se@searn.sunet.se**. Bitnet views this address as a user named **John%sunic.se** at a host named **searn.sunet.se**. Thus, from a Bitnet viewpoint, the whole Internet is in this case seen as local names in **searn.sunet.se**. When the reply reaches searn.sunet.se, this MTA will strip out **@searn.sunet.se** and look at the rest, **John%sunic.se**. It will translate this back to **John@sunic.se** and forward the reply to the original sender.

This example shows that neither the Internet nor the Bitnet will actually view this address as relative. In Bitnet, the address is an absolute address of a user named **John%sunic.se** in the host searn.sunet.se. Only the gateway itself recognizes the translation between % and @.

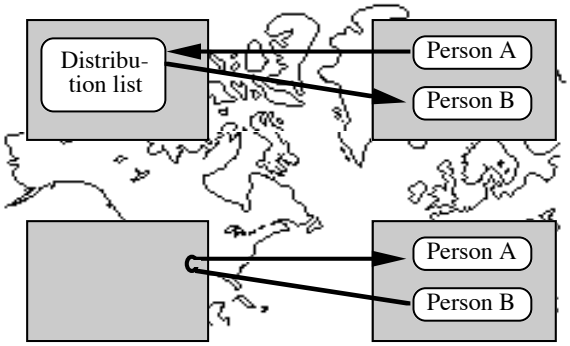
Thus, fundamentalists can proudly claim that “our messaging standard does not use any relative addressing” even though relative addresses are sometimes transported in their net.

There are many disadvantages of relative addresses. You cannot print your address on your business card in the same way for all recipients. You have to indicate a different address depending on to whom you are giving your address. There may also be problems with sending a return message, as shown by the figure below:



Suppose that a person A in America sends a message to two recipients B and C in Europe and that this message uses relative addressing via some American gateway. This means that when B gets the message, the relative address to C is given via this American gateway, so that if B sends a reply to C, this reply must be routed unnecessarily twice across the Atlantic. There are two disadvantages of this. Firstly, it means unnecessary costs and delays, as was shown in the example above. Secondly, the American net may refuse to transmit messages from a European sender to another European recipient, since the American net may have to pay for part of this unnecessary transmission.

A problem similar to that in Figure «#techniques».«#relativeinefficiency» can occur when distribution lists are used, as in Figure «#techniques».«#relative2inefficiency».



In this example, a person A in Europe sends a message to a distribution list in America. The letter reaches a recipient B in Europe via the list. If relative addressing is used, a reply from B to A may have to be transferred via a gateway in America.

Everyone agrees that relative addressing is bad. Why then does it crop up again and again? The reason is that electronic mail is handled by different electronic mail networks, connected via gateways. Since each net has limited knowledge of the internal structure of another net, it can often only address a recipient in another net via a gateway between the nets, and the inclusion of such gateways into addresses makes them relative. The increasing dominance of Internet as a single universal network has however led to relative addressing occuring less and less often.

Internet Message Access Protocol Version 4

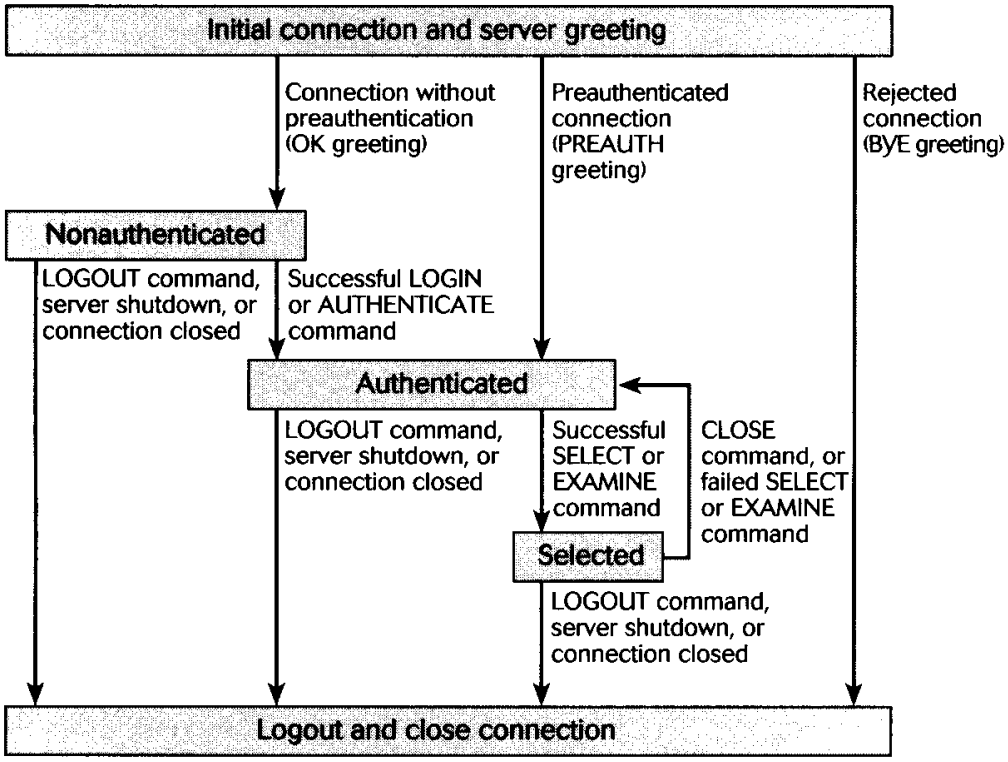


Figure 9-3 *The lAMP state machine* as *described in RFC 2060.*

Internet Message Access Protocol (IMAP) is more suitable when the e-mail client *soft*ware is running on a laptop computer. With IMAP a user can selectively download messages or even just parts of messages. This feature is highly useful when accessing e-mail over a slow telephone line from a laptop computer. RFC 2060 describes IMAP in terms of a state machine. For each state, the client can issue a limited number of IMAP commands to the e-mail server. Some of these commands can cause a transition into another state, wherein a different set of commands would he applicable. Figure 9-3 shows the IMAP state machine as described in the IMAP RFC.

With IMAP, the client can download selected messages from a mail server and then disconnect. In the disconnected state, the client can modify any or all of the messages. In order to allow synchronization, IMAP assigns each message a unique identifier that is valid across any IMAP sessions that the client might establish.

IMAP commands consist of an identifier and a command name, followed by parameters, if any exist. The server returns a tag in its response. The tag allows the client to associate the response with a command. This is necessary because IMAP allows the client to issue multiple commands without waiting for a response. Obviously, the client needs to ensure that each tag is either unique or, if the client uses a pool of tags, that the pool is large enough so that a particular tag is not reused before the server response containing that tag is received. The server can generate responses that are not associated with any particular outstanding command. These are called *untagged responses*, and they use the special asterisk (*) tag.

IMAP results include the following:

- OK indicates successful completion of a previously sent command. OK can also indicate that additional information has been included when sent as an untagged response.
- NO indicates unsuccessful completion of a previous command when tagged; it is used to convey warnings when untagged.
- BAD indicates a bad command or argument when tagged; it indicates a serious protocol problem when untagged.
- PREAUTH is always untagged; it indicates that there is no need for a LOGIN command.

Offprint from Dilip C. Naik: *Internet Standards and Protocols*, Microsoft Press 1998, ISBN 1-57231-692-6

· BYF is always untagged; it indicates that the server is ending the session.

IMAP commands are summarized in the following table. Parameters in square brackets are optional.

Table 9-3 Summary of IMAP Commands

Command	State	Parameters	Description	Result
NOOP	All	None	Used as a "keep alive" to reset timers on server	OK, BAD
CAPABILITY	All	None	Server responds with an untagged response indi cating its capabilities; the only capability defined for now is IMAP 4	OK, BAD
LOGOUT	All	None	Client indicates end of session; server responds with an untagged BYE response	OK, BAD
AUTHENTICATE authenti-	OK, NO,	Nonauthenticated mechanism name	Authentication cation mechanism; defined authentication mechanisms are Kerberos v4, S/KEY, and GSSAPI (described in Chapter 6)	Client indicates BAD
LOGIN	Nonauthenticated	User ID, password	Provides clear text user ID and password; not as secure as the AUTHENTICATE command	OK, NO, BAD
CREATE	Authenticated	Mailbox name	Creates a mailbox	OK, NO, BAD
DELETE	Authenticated	Mailbox name	Deletes a mailbox	OK, NO, BAD
SELECT	Authenticated	Mailbox name	Selects a mailbox for which the client will issue further commands; server responds with untagged responses detailing information about the mailbox as well as the response to the SELECT command	OK, NO, BAD
EXAMINE	Authenticated	Mailbox name	Same as the SELECT command except that the mailbox box is opened in read-only mode	OK, NO, BAD
RENAME	Authenticated	Old mailbox name, new mailbox name	Renames a designated mailbox	OK, NO BAD
SUBSCRIBE	Authenticated	Mailbox name	Adds the given mailbox to the list of subscribed mailboxes	OK, NO, BAD
UNSUBSCRIBE	Authenticated	Mailbox name	"Undoes'" the SUBSCRIBE command; removes a given mailbox from the sub scribed list	OK, NO, BAD
APPEND	Authenticated	Mailbox [flags] [date-time] message	Appends a designated message to the specified mailbox; other parameters (if present) are appended to the message	OK, NO, BAD

Command	State	Parameters	Description	Result
LIST	Authenticated	Context, mailbox	Allows the user to list a subset of available names; the context argument pro vides the server with addi tional context	OK, NO, BAD
LSUB	Authenticated	Context, mailbox	Same as LIST command except that the server limits its response to mailboxes that subscribed via the SUBSCRIBE command	OK, NO, BAD
STATUS	Authenticated	Mailbox	Requests the status of the named mailbox	OK, NO, BAD
FETCH	Selected	Message set, message item name	Retrieves data pertinent to a message; data could be parts of a message or the whole message	OK, NO, BAD
STORE	Selected	Message set, message item name, message item data	Changes data associated with specified message; by default the changed value is returned in an untagged response	OK, NO, BAD
CHECK	Selected	None	Requests implementation-dependent checkpointing of the currently selected mailbox; the resulting mail box state is stored to disk	OK, BAD
EXPUNGE	Selected	None	Deletes all messages marked for deletion	OK, NO, BAD
SEARCH	Selected	[character set], fsearch criteria]	Searches mailbox for mes-sages that match given cri-teria	OK, NO, BAD
COPY	Selected	Message set, mailbox name	Copies specified messages into specified mailbox	OK, NO, BAD
UID	Selected	Command name, command arguments	Indicates specified com-mand (COPY, FETCH, STORE, or SEARCH) that should be executed with unique identifiers, rather than message sequence numbers, as input	OK, NO, BAD
X	Selected	Implementation defined	Experimental implementation-dependent command	OK, NO, BAD
CLOSE	Selected	None	Deletes messages marked for deletion; causes a tran-sition to authenticated state	OK, NO, BAD

IMAP is more complex than POP3 and is harder to implement. IMAP also puts more demands on storage resources at the e-mail server, since old messages might accumulate on the server. On the other hand, it is more likely that a decent backup strategy is in place at the e-mail server, thus protecting data integrity. Also, IMAP allows the possibility of implementing groupware applications, since server-side mail processing as well as shared mailboxes are supported features. (A shared mailbox is a mailbox that can be accessed by multiple recipients.) IMAP also allows for searches to be implemented on e-mail still residing on the server, without downloading the mail to the client.

When TCP is used, the IMAP server listens on port 143.

At this writing, both Microsoft and Netscape have committed to shipping IMAP e-mail client and server software. SunSoft, ICL, and NetManage also have IMAP e-mail software implementations.

IMAP is described in RFCs 2060,1731, and 1730.

Pretty Good Privacy

Pretty Good Privacy (PGP) leverages private key cryptography, public key cryptography, and message digests (all described in Chapter 5, "Cryptography and Security Basics"). PGP ensures message confidentiality (only the intended recipient can decrypt and read the message) and message authentication (the recipient can be sure of the identity of the sender). But keep in mind that, strictly speaking, PGP is used to ensure privacy of files, not just privacy of e-mail.

To ensure confidentiality, PGP encrypts a message with a randomly generated 128-bit key using the International Data Encryption Algorithm (IDEA) symmetric block cipher. This key is then placed in a message header, which is encrypted using the RSA public key cipher. The e-mail recipient's public key is used as the key for the RSA cipher. The recipient decrypts the header using the recipient's private key to recover the randomly generated key that's embedded in the message.

To ensure authentication, a technique called digital signatures (see Chapter 5) is used. The message is processed using MD5, which produces a 128-bit hash. The hash is then encrypted using RSA, with the sender's private key being the input key for RSA. This encrypted hash is prefixed to the message. The recipient extracts this encrypted hash and decrypts it using the sender's public key. The recipient also decrypts the message and independently computes the MD5 hash of the message. The computed MD5 hash should be identical to the decrypted hash.

PGP provides for three different key lengths:

- A "casual" option that has 384-bit keys
- A "commercial" option that has 512-bit keys
- A "military" option that has 1024-bit keys

PGP stores a user 5 public and private key in files called *keyrings* on a computer disk. These keys are stored in an encrypted form. Each user has to provide a password to access the keys. The MD5 message digest algorithm is used to produce a digest of the password phrase. This message digest is used as the key to encrypt the keys with the IDEA cipher. PGP has a crucial weakness that is common to public key cryptography systems: how to ensure that a public key is correct and belongs to a particular user.

The digital signature produced by PGP and the encrypted message can have the most significant bit of a byte set. These messages may or may not be successfully transmitted through e-mail servers because some servers assume that the most significant bit is always reset. PGP provides a scheme of mapping every three characters into four characters, with the most significant bit clear.

PGP is available on a wide variety of platforms, including Microsoft Windows, DOS, UNIX, Macintosh, and VMS. Both freeware and commercial versions of PGP are available. Some versions do not meet U.S. export requirements. An international version of PGP, called PGP5I, was created by printing PGP5 code in a book and then exporting the book (but not the code). The code was then scanned into a computer. This export version meets U.S. government requirements.

PGP is described in RFCs 2015 and 1991. See http://www.pgp.com for more details.

News and Usenet

Usenet allows users to have discussions in public forums called *newsgroups.* A message that's sent to the newsgroup is forwarded to every user who subscribes to the newsgroup. Usenet is the name given to a loosely coupled network of computers that exchange e-mail messages. The messages are tagged with particular subjects or headers. These headers identify the newsgroup to which the message belongs. The message is called an *article.* This exchange of articles among servers as well as between clients and servers is accomplished using the Network News Transfer Protocol (NNTP), described in the next section.

The major difference between Usenet and a list server is in the way a user accesses and receives information. With Usenet, the user has to fire up a news reader and download messages. The subscriber can look at article headers and then retrieve the full text for only selected articles. With a list server, however, the subscriber receives e-mail directly in his or her mailbox.

Newsgroups are not centrally administered. They are essentially self-policing mechanisms. Newsgroups are named according to a particular convention: names consist of multiple components, each separated by a dot. The first component indicates the category of the newsgroup (for example, *soc,* as used in soc.culture.india). The major newsgroup categories (identIfied by the leading component) are shown in Table 9-5.

Table 9-5 Malor Newsgroup Categories

Group Name	Description
biz	Dedicated to business-related issues
comp	Related to computers, computer software, networking, and other computer-related issues
sci	Dedicated to science-related subjects
soc	Dedicated to social and cultural issues
talk	Provided for lengthy discussions and debates
news	Dedicated to discussion of news and information
rec	Dedicated to recreational activities such as the arts, hobbies, and so on
misc	Associated with miscellaneous issues that do not fit any of the other major categories
alt	Limited distribution category whose content has a wider latitude than other groups

For more details, see RFC 1036.

Network News Transfer Protocol

Network News Transfer Protocol (NNTP) defines a standard for

- Clients to post news articles to servers
- Clients to retrieve and read news articles from servers
- Articles to be exchanged between news servers

NNTP is similar to SMTP. For example, NNTP messages must use the ASCII character set, and NNTP commands are in the form of simple ASCII text. Commands are ASCII lines with the command name followed by optional parameters that are specific to the command involved. Each command is terminated by ASCII CR and LF characters. Like SMTP, the NNTh responses consist of a three-digit code followed by ASCII text. The three-digit code can be used to drive a protocol state machine, whereas the text can be displayed for human consumption. Responses are terminated by the same CR/LF sequence. When a command results in the server sending back more data, such as an article body, each line is terminated by the CR/LF sequence. The end of the response is indicated by a line with a single dot (.) followed by a CR/LE Just as in SMTP, dot stuffing is employed; any line that begins with a dot will have the dot doubled, then a CR/LF appended at the end for sending, and the extra dot stripped by the receiving server.

NNTP responses consist of three digits. The first digit conveys information about the success or failure of the message. Table 9-6 describes the semantics of the first digit of the NNTP response code.

Table 9-6
First-Digit NNTP Response Code Semantics

lxx	Informative message
2xx	Command OK
3xx	Command OK so far; send the rest
4xx	Command was correct but could not be performed
5xx	Command incorrect or unimplemented; serious error

The second digit in the response code indicates the function response category. Table 9-7 summarizes the semantics for the second digit of the NNTP response code.

Table 9-7 Second-Digit NNTP Response Code Semantics

x0x	Connection, setup, and miscellaneous messages
xix	Newsgroup selection
x2x	Article selection
x3x	Distribution functions
x4x	Posting
x8x	Nonstandard implementation-specific extensions
x9x	Debugging output

NNTP provides 15 commands that a client can issue to the server.

Note that an NNTP server might occasionally act as a client when NNTP messages are being relayed. Table 9-8 summarizes the NNTP commands. Square brackets identify optional parameters. Angle brackets, where shown, are syntactically required.

Table 9-8 Summary of NNTP Commands

Command	Arguments	Description
ARTICLE	<msg-id>	Retrieves the text of the message specified by *msg-id*
ARTICLE	*[nnnn]*	Retrieves the text of the message specified as article index number *nnnn*
BODY	<msg-id>	Retrieves the body associated with the message specified by *msg-id*
GROUP	GroupName	Selects *GroupName* as the current group and returns the numbers of the first and last articles, as well as an esti mate of the number of articles in the group
HEAD	<msg-id>	Retrieves the header associated with the message specified by msg-id
HEAD	[nnn]	Retrieves the header associated with the message specified as article index number nnnn
HELP	none	Provides an ASCII text response detail ing commands implemented by the server
IFIAVE	<msg-id>	Client informs the server that it has the message specified by msg-id; if the server sends a positive response, the client transfers the entire message; if the server sends a negative response, the client must not send the message
LAST	none	Moves the current article pointer back to the previous article within the same newsgroup
LIST	none	Returns a list of group names; for each group the msg-id of the first and last messages is returned, with an indica tion of whether posting to that group is permitted
NEWSGROUPS	date, time, [GMTI, [<dist>]	Returns a list of newsgroups created since specified date and time; time is in server's time zone but can be optionally specified as GMT; dist is optional and used to restrict the list of newsgroups returned

NEWNEWS	newsgroups, date, time	Returns message IDs of new messages in specified group since specified date and time
NEXT	none	Moves current article pointer to next article in current newsgroup
POST	none	Used to post new messages to server; server can respond positively or negatively
QUIT	none	Terminates the session
SLAVE	none	Informs the server that the client is another slave server
STAT	msg-id	Similar to ARTICLE except that no text is returned; typically used to position current article pointer
XOVER	none	Not described in RFC 977; de facto standard command that retrieves header information for a group of articles using a single command

NNTP is documented in RFC 977.