Excerpt from:

Internet application layer protocols

By Jacob Palme

Table of contents

1

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.

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:
```

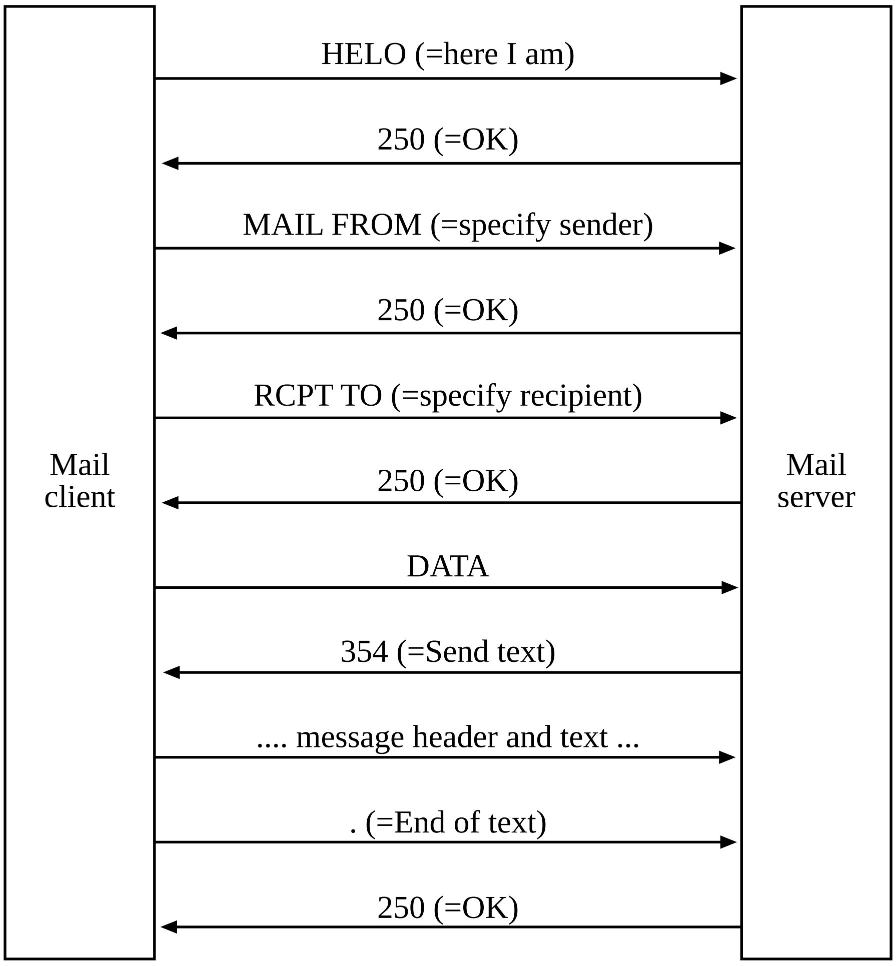


Figure 1: Interactions for sending an e-mail message

```
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

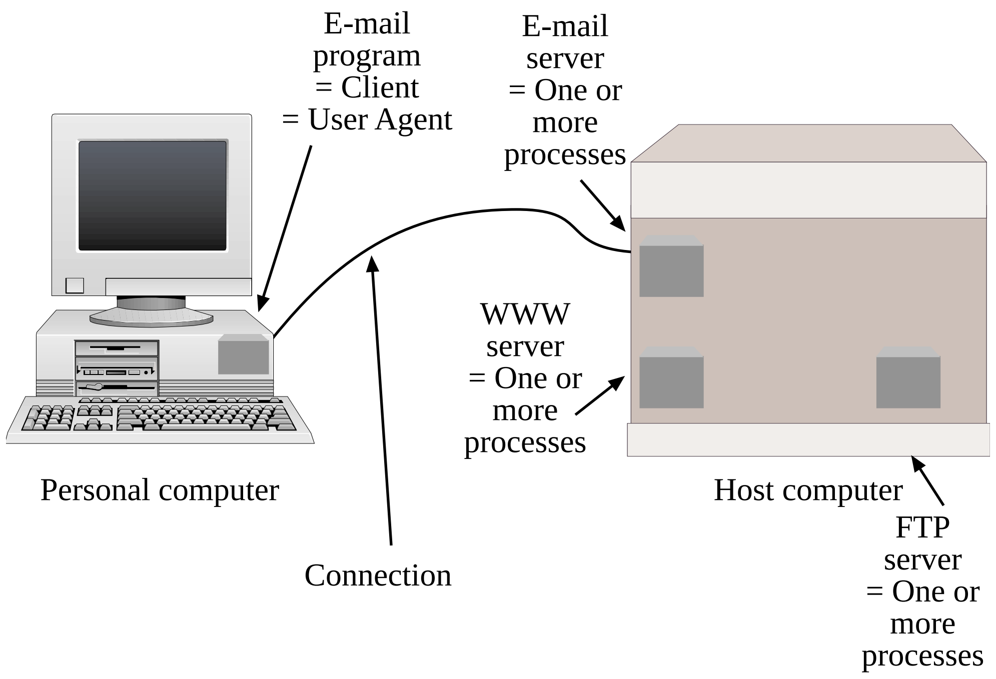

E-mail program = Client = User Agent
E-mail server = One or more processes
WWW server = One or more processes
Personal computer
Host computer
Connection
FTP server = One or more processes

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

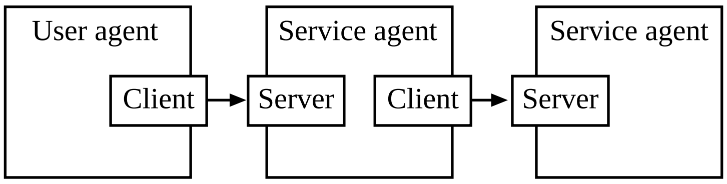


Figure 3: The middle agent in this figure acts as a server to a user agent and as a client to another service agent.

two service agents. The middle service agent acts as a server to the user agent, and as a client to the other user 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:

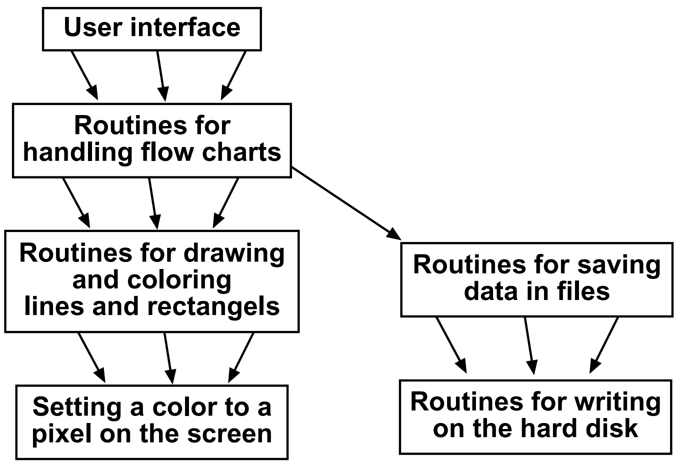

User interface
Routines for handling flow charts
Routines for drawing and coloring lines and rectangels
Routines for saving data in files
Setting a color to a pixel on the screen
Routines for writing on the hard disk

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.

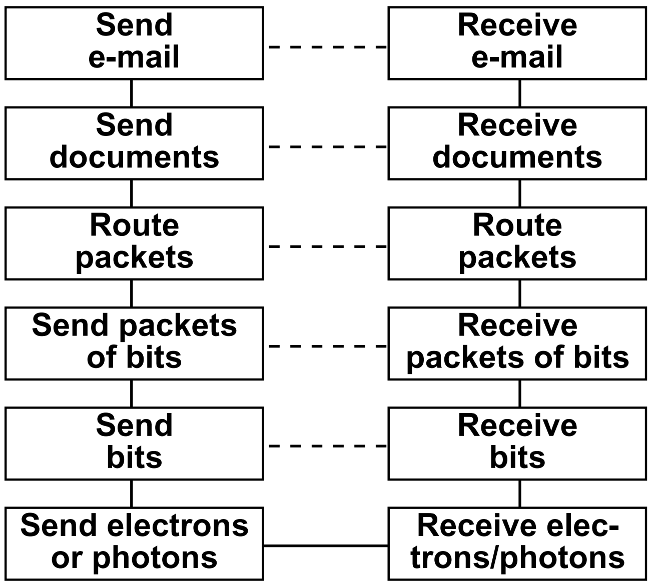


Figure 5: Two stacks of layers, one on each host, communicating with each other.

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.

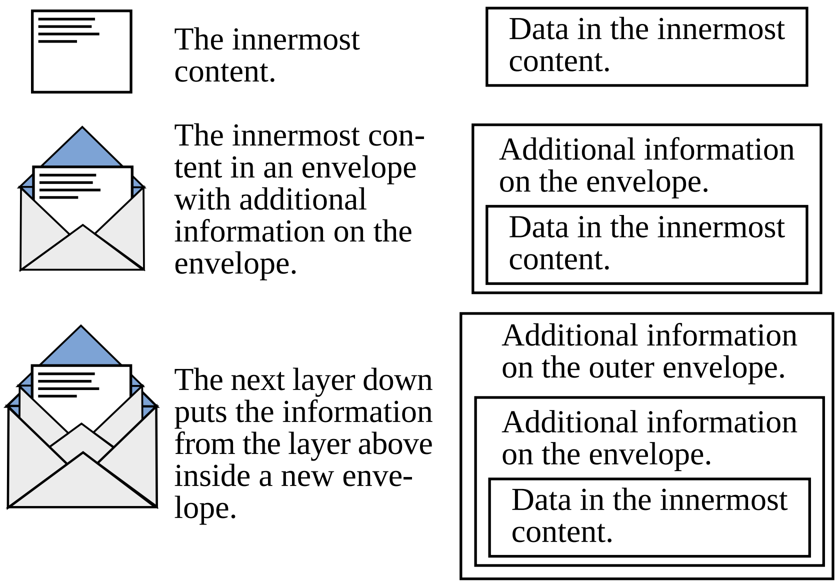

The innermost content.
Data in the innermost content.
The innermost content in an envelope with additional information on the envelope.
Additional information on the envelope.
Data in the innermost content.
The next layer down puts the information from the layer above inside a new envelope.
Additional information on the outer envelope.
Additional information on the envelope.
Data in the innermost content.

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”.

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

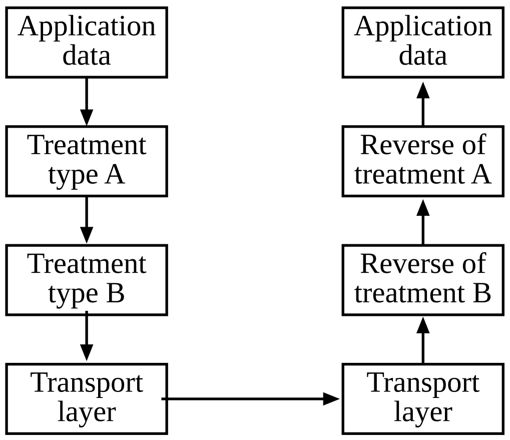


Figure 7: Layering within the application layer (layers below the transport layer omitted)

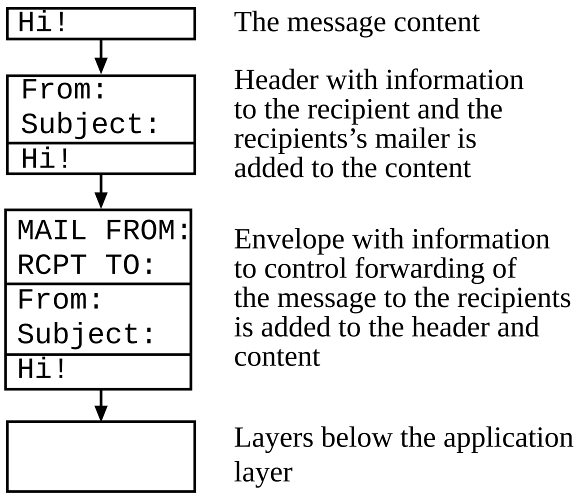

Hi!
The message content
From:
Subject:
Hi!
Header with information to the recipient and the recipients's mailer is added to the content
MAIL FROM:
RCPT TO:
From:
Subject:
Hi!
Envelope with information to control forwarding of the message to the recipients is added to the header and content
Layers below the application layer

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

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

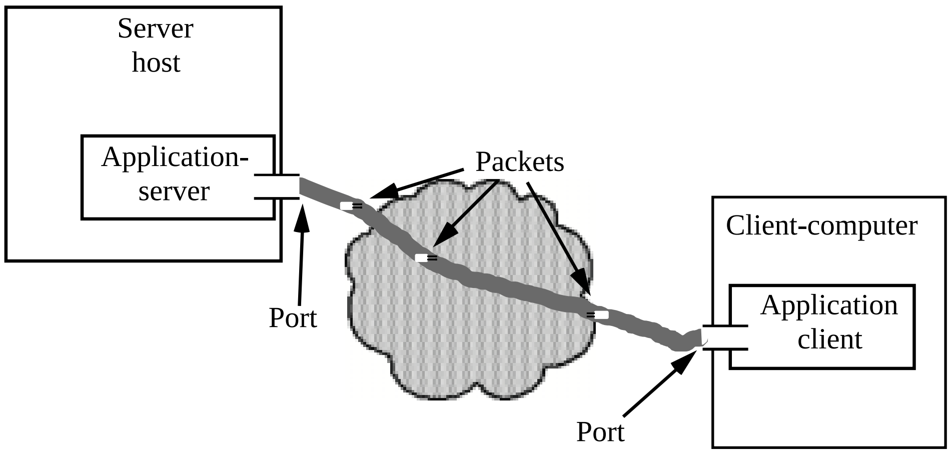


Figure 9: A TCP connection between two computers can be seen as a hose, through which packets of octets can be sent between ports. The port are openings to applications running on the computers.

applications. And the hoses fit into different ports for different appli-ations. 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 proto-col. 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.

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

Table 2: Some common port numbers

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].

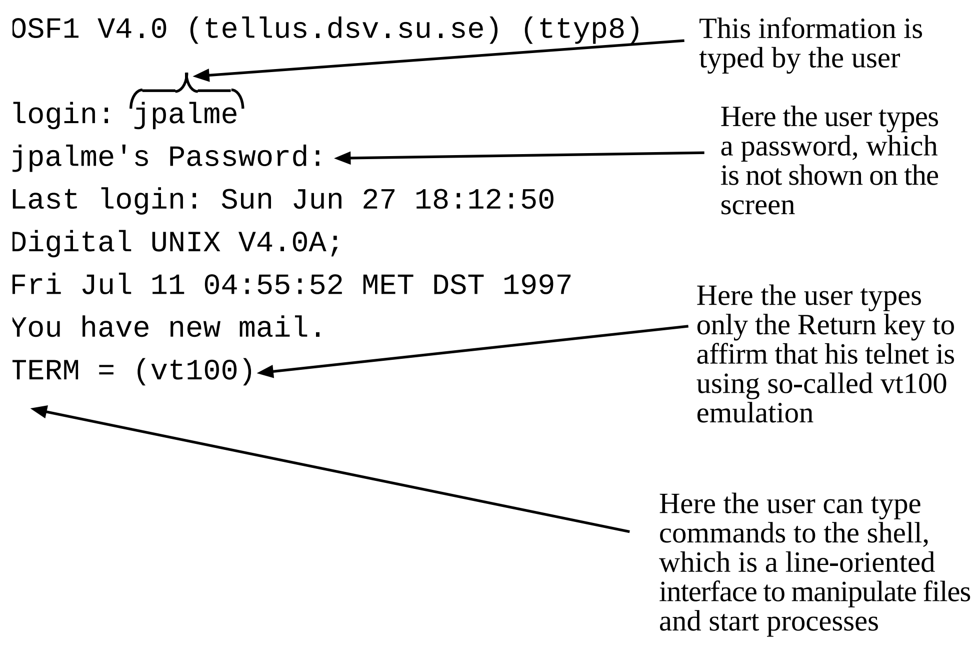
1.6 Telnet: A Simple Application

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.

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



```
OSF1 V4.0 (tellus.dsv.su.se) (ttyp8)

login: jpalme
jpalme's Password:
Last login: Sun Jun 27 18:12:50
Digital UNIX V4.0A;
Fri Jul 11 04:55:52 MET DST 1997
You have new mail.
TERM = (vt100)
```

This information is typed by the user

Here the user types a password, which is not shown on the screen

Here the user types only the Return key to affirm that his telnet is using so-called vt100 emulation

Here the user can type commands to the shell, which is a line-oriented interface to manipulate files and start processes

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

“`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 shows a simple and common architecture, with a single server, and a number of clients which access this server. With this ar-

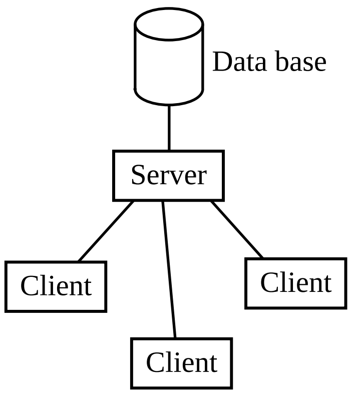


Figure 11: Simple client-server architecture

chitecture, 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.

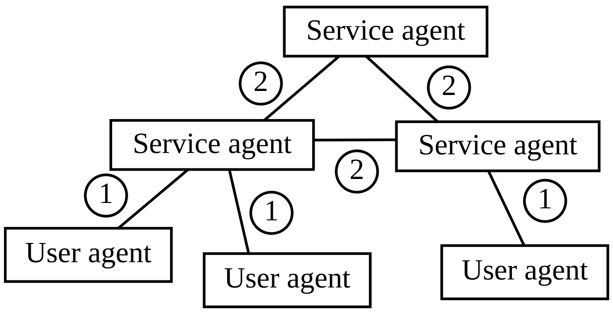


Figure 12: Architecture with several interconnected servers

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 ① and ② in Figure 12. Connections between a user agent and a service agentis labelled ① in the the figure, and connections between two service agentis labelled ② 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

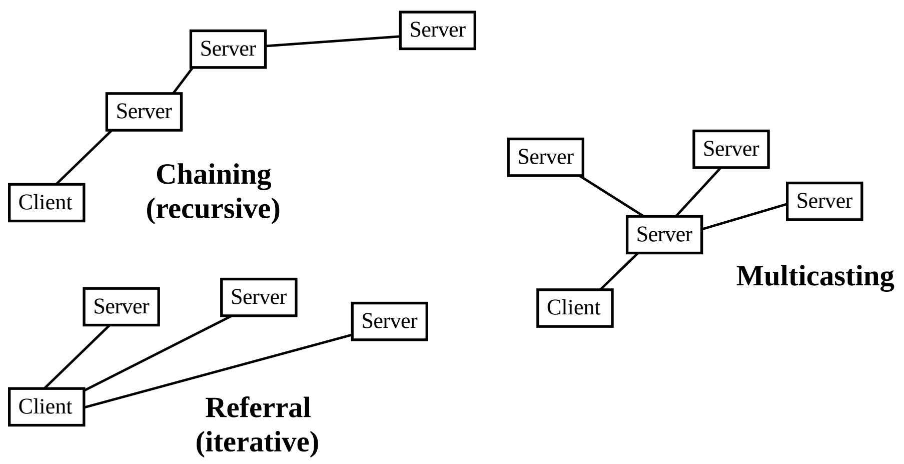


Figure 13: Chaining, Referral and Multicasting

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.

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

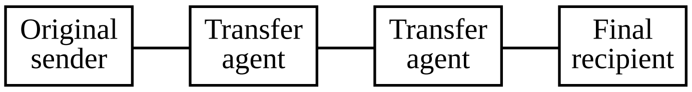


Figure 14: Store-and-forward of e-mail messages

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:

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.

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

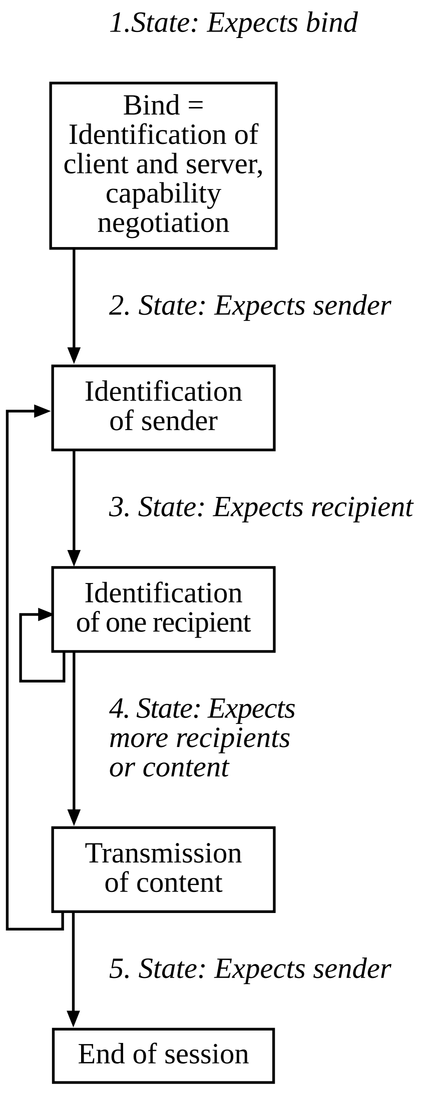

1.State: Expects bind
Bind = Identification of client and server, capability negotiation
2. State: Expects sender
Identification of sender
3. State: Expects recipient
Identification of one recipient
4. State: Expects more recipients or content
Transmission of content
5. State: Expects sender
End of session

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 server expects the client to identify it self. No other commands are allowed in this state.
2. When identification is ready, the server expects the address of the sender.
3. When the sender has been given, the server expects the address of the first recipient.
4. When a recipient address has been given, the server expects either an additional recipient address, or the message content.
5. When message content. has been sent, another message can be sent. This is the same 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.

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

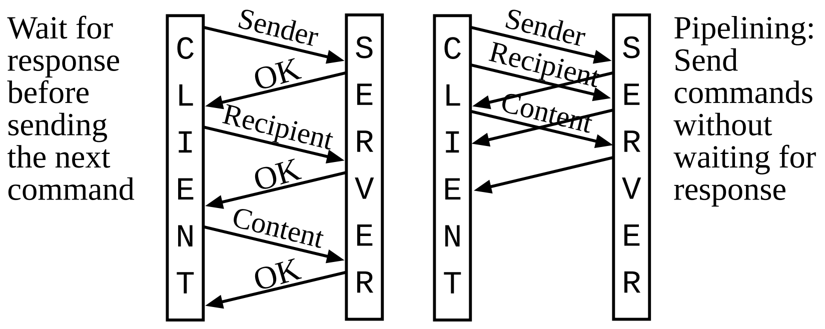

Wait for response before sending the next command
CLIENT
Sender
OK
Recipient
OK
Content
OK
SERVER
CLIENT
Sender
Recipient
Content
SERVER
Pipelining: Send commands without waiting for response

Figure 16: Pipelining reduces the time

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.

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.

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.

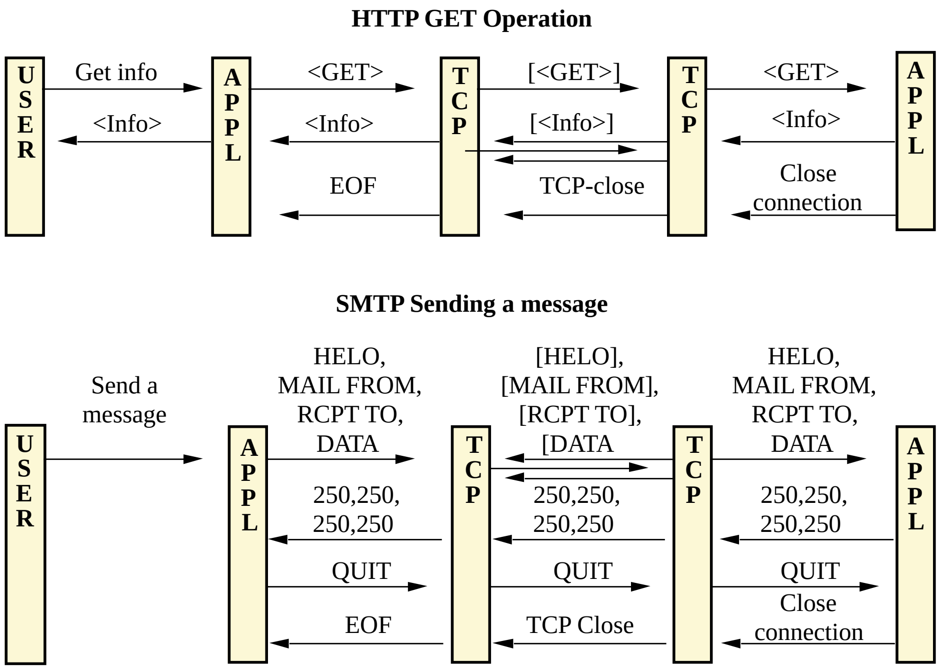


Figure 17: How a connection is ended in HTTP 1.0 and SMTP. In HTTP, the server closes the connection as soon as TCP tells it that all data has been transformed. In SMTP, the client sends a QUIT command, which tells the server to close the connection. Note that no timeout wait is necessary in either case.

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):

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.

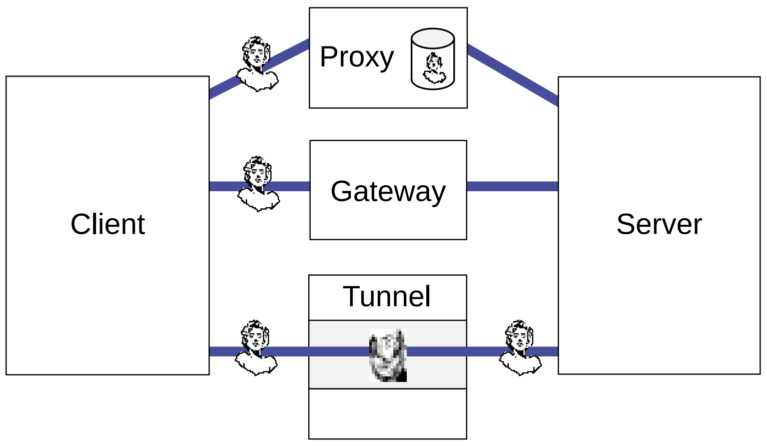


Figure 18: Intermediaries between client and server

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.

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

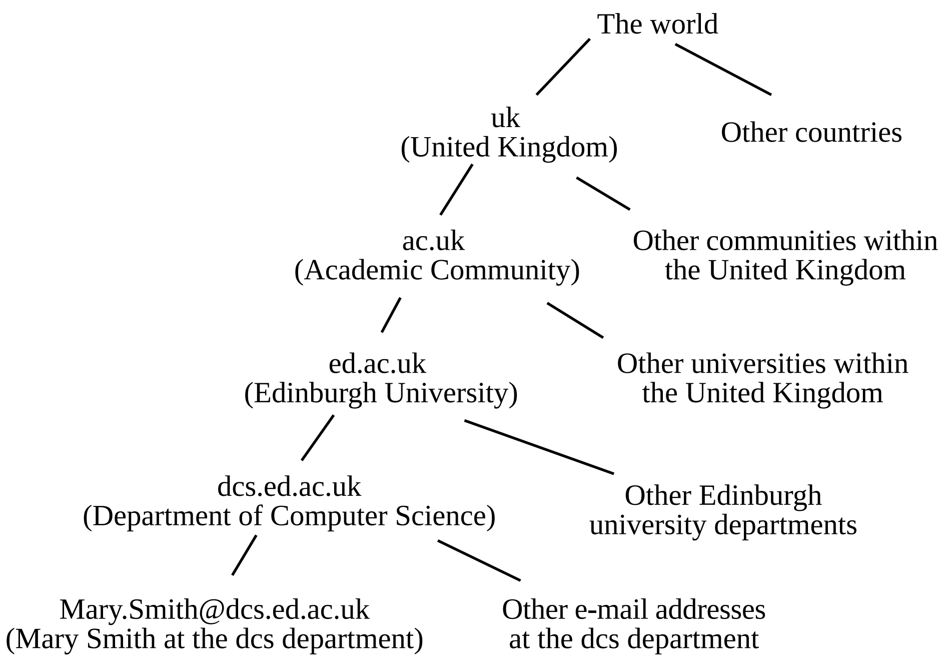

The world
uk (United Kingdom)
Other countries
ac.uk (Academic Community)
Other communities within the United Kingdom
ed.ac.uk (Edinburgh University)
Other universities within the United Kingdom
dcs.ed.ac.uk (Department of Computer Science)
Other Edinburgh university departments
Mary.Smith@dcs.ed.ac.uk (Mary Smith at the dcs department)
Other e-mail addresses at the dcs department

Figure 19: Hierarchical domain 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).

$$\texttt{From: }\underbrace{\texttt{Eliza Clark}}_{\text{User-friendly name}}\ \underbrace{\texttt{<elizac@foo.bar.net>}}_{\text{Globally unique e-mail address}}$$

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 serv-

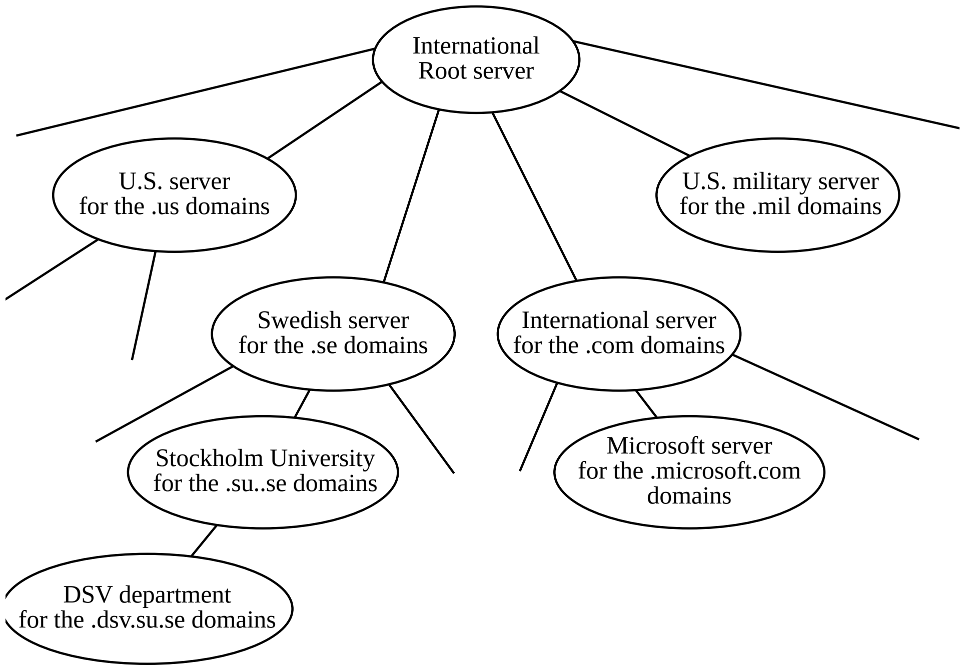

International Root server
U.S. server for the .us domains
U.S. military server for the .mil domains
Swedish server for the .se domains
International server for the .com domains
Stockholm University for the .su..se domains
Microsoft server for the .microsoft.com domains
DSV department for the .dsv.su.se domains

Figure 21: Conceptual structure of the DNS server hierarchy

ers are duplicated more than twice to distribute load and increase reliability.

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):

COM	commercial entities
EDU	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.

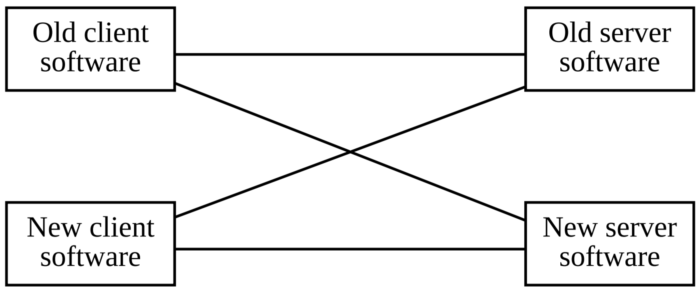


Figure 22: Difficulty of co-working between different versions of protocols

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

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

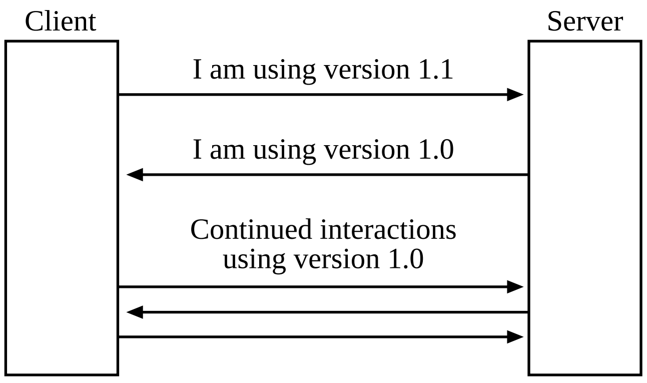


Figure 23: Version number method

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

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 manda-

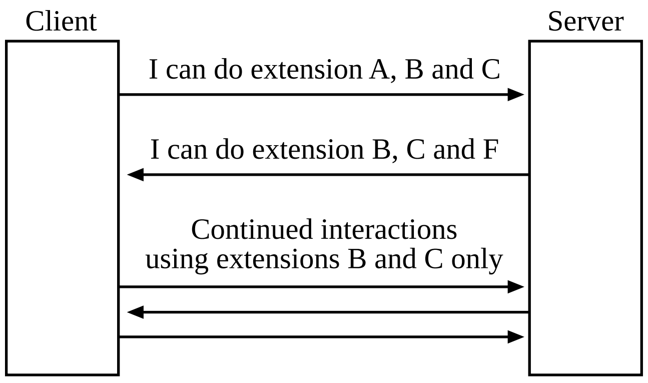

Client
Server
I can do extension A, B and C
I can do extension B, C and F
Continued interactions
using extensions B and C only

Figure 24: Feature selection method

tory 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.)

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 %%%

1.22 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