The Hypertext Transfer Protocol in the World-Wide Web Library of common code

Henrik Frystyk Nielsen

August 10, 1994

Preface

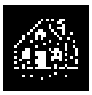
This paper is an attempt to document my work as a technical student at CERN , "the European Laboratory for Particle Physics" from February 1 1994 to August 1 1994. It is, however, also an attempt to visualize the potential features of the information exchange system called the World-Wide Web that I have been working on during this period. The document serves as my master thesis from Aalborg University, Denmark, Institute of Electronic Systems, Department of Communication Technology

During my stay, my main task has been to maintain and develop the World-Wide Web Library of Common Code and the Line Mode Browser. As the library is a part of the CERN HTTP Server a part of my time has been devoted to this application as well. However, this thesis reflects my work in designing, developing and testing software modules that provide the interface between the World-Wide Web applications and the Internet. Especially attention has been put into the design of the Hypertext Transfer Protocol (HTTP) client and how it has been extended to provide a multi-threaded interface to the World-Wide Web applications.

The documentation is organized into several hypertext documents which gives the reader the possibility of reading it in an unknown number of ways which makes it fundamentally different from what any paper version can provide. All internal references are made as hyperlinks so that any jump back, or forth, in the documentation is immediately accessible using whatever means are available within the World-Wide Web client used. All external references available on the Internet are also accessible through hyperlinks. This means that the reader is not limited within this particular information but at any instant can jump to any other information provider on the Internet using the World-Wide Web. Another characteristic is, that unlike a paper version, this documentation is constantly changing as the Web itself is constantly changing. This is an inherent consequence of using a global information system with thousands of highly independent users and information providers.

The documentation is organized in a tree structure in which this document serves as the top node or the root document for further browsing. The following list is a "list of contents" that indicates the overall structure of the tree and a proposed way of traversing it.

10. Multi Threaded Clients
11. Client Interface for Posting
12. Summary
13. References



However, any other path might be just as good, and I hope that when you are reading it you will find your own way through. All documents related to this work have a "home link" symbolized by the little icon to the left that points back to this page.

The thesis is also available in a Postscript version for A4 and Postscript version for 8.5"x11". It has been generated from the HTML version using the Line Mode Browser as a filter.

Finally, I would like to thank my supervisor at CERN, Tim Berners-Lee and Ari Luotonen for their great support and source of inspiration throughout the project.

Abstract

The subject of this thesis is to introduce the World-Wide Web as a generic model for exchanging information on the Internet using a set of Presentation Layer protocols with the Hypertext Transfer Protocol as the most important. The description is related to the lower layers of the Internet TCP/IP protocol stack.

The software developed and maintained by the World-Wide Web team at CERN is described with attention put on the World-Wide Web Library of Common Code and an updated control flow chart of the library architecture.

A multi-threaded, interruptable I/O Hypertext Transfer Protocol client based on an event driven loop has been designed and implemented as a state machine. Furthermore, a design for how World-Wide Web Clients interface to the library when posting a data object to one or more remote servers is designed and implemented.

The code is currently available in an alpha version (August 1994), but will officially be released later this summer.

Synopsis

Emnet for denne rapport er at beskrive World-Wide Web som en generel model for hvorledes information kan udveksles via Internetet. Modellen anvender et sæt af Presentation Layer protokoller hvoraf Hypertext Transfer Protocol er den vigtigste. Beskrivelsen er relateret til de nederste lag af TCP/IP protokol stakken, som anvendes på Internettet.

Endvidere er software implementationen af World-Wide web modellen, som er udviklet og vedligeholdt af World-Wide Web team p CERN beskrevet. Særlig vægt er lagt på beskrivelsen af World-Wide Web Library of Common Code og et nyrevideret kontrol flowdiagram af den anvendte arkitektur i biblioteket.

Rapporten beskriver designet og implementeringen af en multi-threaded Hypertext Transfer Protocol klient med interruptbar I/O. Designed er udformet som en state machine baseret på en hændelsesdrevet løkke. Endelig beskrives designet og implementeringen af et interface mellem en World-Wide Web klient og Library of Common Code for når en klient vil sende data over til en fjern server (post).

Den udviklede software kode er i øjeblikket tilgængeligt i en alfa version (august 1994), men en officiel version vil blive frigivet senere denne sommer.

Contents

Chapter 1

Introduction to the Internet

This chapter gives an overview of the Internet. It presents the history and the basic model of the Internet, but it is not an attempt to describe the Internet in all its detail which would be out of scope at this place. Please see Douglas Comer for an excellent description of the Internet. The structure of the document is as follows:

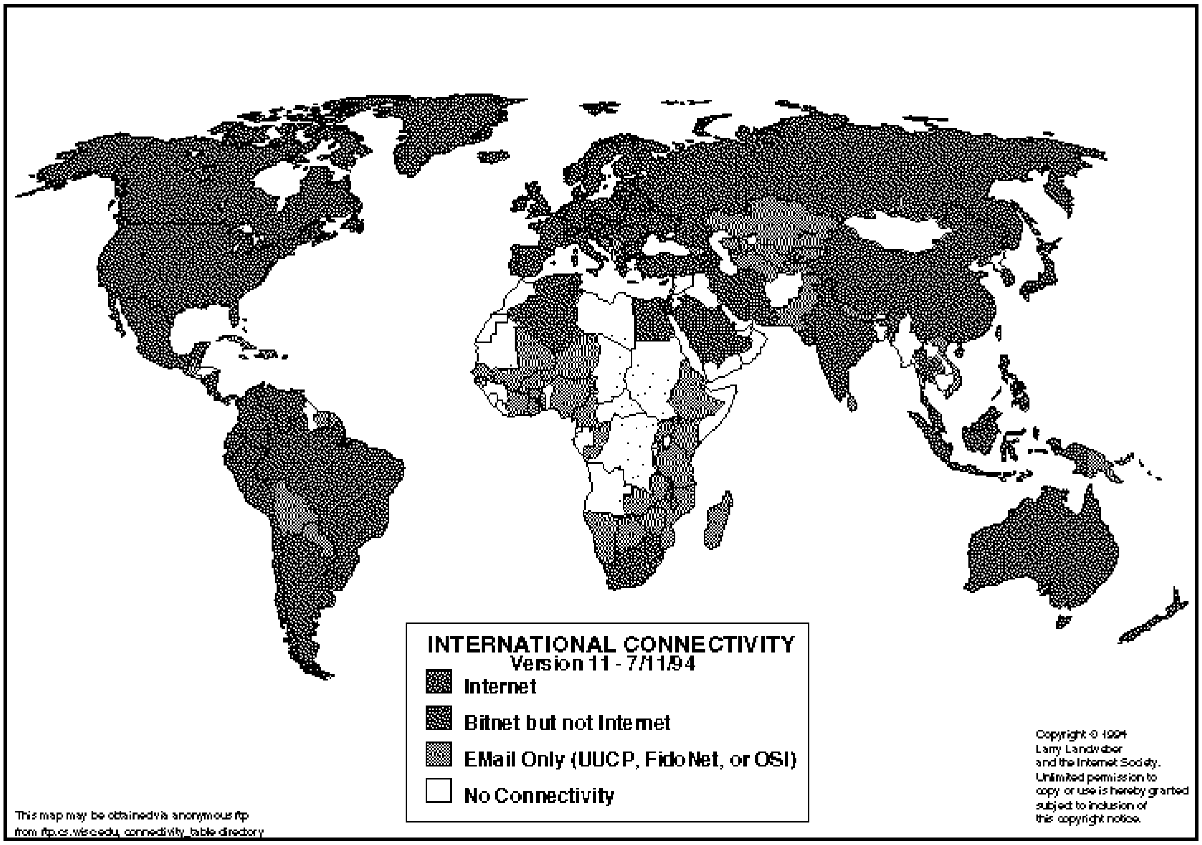
1. Architectural Model
2. Address Scheme
3. Domain Name Server
4. Gateways and Routing

In the late 1960s the American Defense Advanced Projects Research Agency, ARPA (later DARPA) started a research project on the subject of computer networks. One of the first results of this project was an experimental four node network starting in 1969. Later the network expanded to include several military installations and scientific research centers. In the mid 1970s work began towards the Internet with the architecture and protocols taking their current form around 1978-79.

The Internet as we know it today started around 1980 when DARPA started to use the TCP/IP protocol stack on all installations connected to the DARPA Internet. The transition ended in the beginning of 1983 when TCP/IP became the only protocol stack allowed on the Internet. This is still the current situation on the Internet, but now it has grown to several thousands of nodes and millions of users. The countries connected to the Internet are illustrated in the figure below.

In 1988, DARPA decided that the experiment of ARPANET was complete and started to dismantle the ARPANET that, until then, was the backbone of the Internet. However at the same time, the American National Science Foundation established the NFSNET which then became the new backbone network with a capacity (1992) of 45 MBPS.

Many Internet organizations other than DARPA have an important influence on the further development of the Internet. A few of them are mentioned below:



The Internet Activities Board (IAB) This organization was created in 1983 in order to guide the evolution of the Internet development. It now has two major components: Internet Engeneering Task Force and the Internet Research Task Force.

Internet Engeneering Task Force (IETF) The IETF is the protocol engeneering, development, and standardization branch of the Internet Architecture Board (IAB). IETF manages the Request for Comments (RFC) documents

Internet Research Task Force (IRTF) The IRTF is the research and development branch of IAB. They do research in new network technologies and

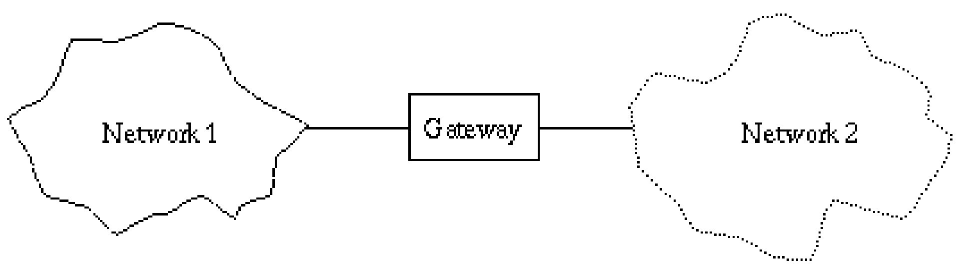
InterNIC Information Services The InterNIC is a collaboration project of three organizations: General Atomics, AT&T, and Network Solutions, Inc. Their goal is to make networking and network information more easily accessible to researchers, educators and the general public. They work together with the Network Information Centers (NICs) located throughout the Internet.

1.1 Architectural Model

The term "Internet" is a generalization that covers thousands of interconnected networks around the world based on very different technologies. The networks differ in almost any possible network specific parameter such as transmission medium, geographical size , number of nodes, transmission speed, throughput, reliability etc. The only reason why this generalization is possible is because the Internet is based on an abstraction that is independent of the physical hardware. In short, it represents a homogeneous interface to its users in spite of the heterogeneous hardware that it is based on.

The diversity among networks connected to the Internet is partly due to an evolution of technology resulting in new networks having higher reliability, better throughput etc. However, there will (at least for a long time) exist a need for fundamental different network architectures as no network technology today can supply a solution that covers all aspects of internetworking.

This section introduces the basic architecture of how the Internet is organized. The description starts at a certain abstraction level that does not include a description of the underlying physical network technologies such as Ethernet, Token Ring, FDDI etc. These are all described in Computer Networks. The basic idea of an internet is to provide the possibility of transporting data from one network to another through a connection in a way that both parties agree on and understand. The connection between the two consists of a gateway computer that is physically or logically connected to both networks (logically in the case of a cordless network). The situation between two networks looks like:



Each *cloud* is a network with an arbitrary number of connected nodes. The gateway between them serve as the only way of exchanging data directly between the two networks. Later in this chapter it is described how two hosts can communicate even though they are not connected directly but must go through intermediate networks.

1.2 Address Scheme

In order to reference any node as a unique point on the Internet, a global two dimensional 32-bit integer address space has been defined which gives a maximum number of 4G connected nodes on the Internet. The first element is a netid and the second is a nodeid, that is:

```
address = (netid, hostid)
```

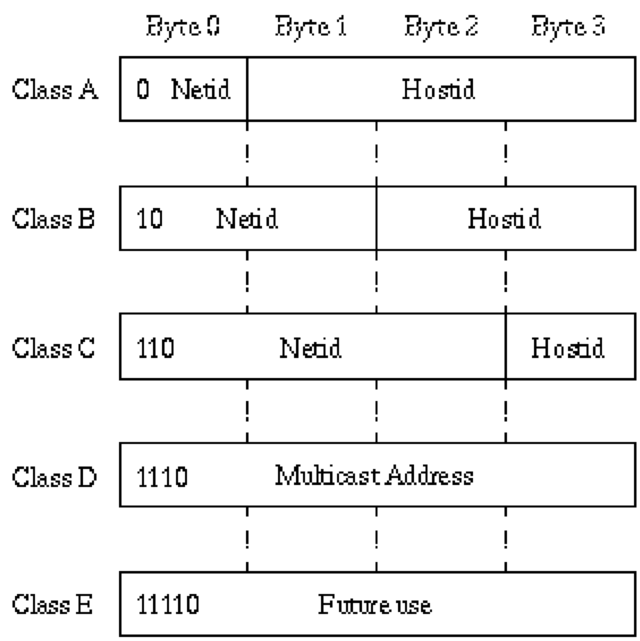
A common notation for specifying an Internet address is by using four fields of decimal integer numbers ranging from 0 to 255 separated by decimal points, e.g.:

```
128.141.201.214
```

which is the IP-address of the World-Wide Web info server at CERN.

1.2.1 Address Classes

In order to provide IP-addresses which suit both large networks with millions of hosts and small networks with a few hundred hosts, the netid part and the hostid part can occupy a varying part of the IP-address. The number of possible nodes on a network, being the amount of bits assigned to the hostid, categorizes the address space into 5 classes:



The definition of the classes is as follows:

Class A This class has a 1 byte netid and a 3 byte hostid. As networks in this category are characterized by having a 0 as the first bit in the address, the maximum number of networks is 128. However, as 24 bits are available for the hostid, each network can contain 16M connections. A network can be categorized by the first fields in the address and for a Class A network the value of first field is in the range 0-127.

Class B Class B networks have 2 bytes for the netid, but as they are required to start with the bit combination 10b, the maximum number of networks is 16K. The number of connected nodes is 64K and the value of the first field ranges from 128-191 and the second from 1-254.

Class C This class is for small networks with a maximum number of nodes limited to 256. This class is characterized by having the leading bit pattern 110b which leaves the maximum number of networks to 2M. The value of the first field is from 192-233, the second from 0-255, and the third from 1-254.

Class D Class D networks are networks without the possibility of addressing any individual node. All 32 bites are used by the netid and hence any reference to the network is automatically a broadcast message to all the connected hosts. The characteristic leading bit pattern for this class is 1110b.

Class E This is currently not in use but reserved for future use. However, the characteristic leading bit pattern for this class is defined as 11110b.

From this description it can be seen that the IP-address given above is a Class B network with the possibility of 64K nodes.

An interesting thing to note about having the IP-address containing information of the network is that a gateway as a consequence of being connected to two networks must also have two IP-addresses in order to be accessible from both sides. This is the reason for not referring to a number of hosts but nodes or connections to the network. In the Gateways and Routing it is described how the current addressing scheme influences the routing algorithms used on the Internet.

1.2.2 Physical Addresses

It is important to note that Internet addresses are an abstraction from the addresses in a physical network implementation like Ethernet. They assure that the same addressing scheme can be used in every part of the Internet regardless of the implementation of the underlying physical network. In order to do this, a binding must exist between the IP-address and the physical address. Dependent on the physical network addressing scheme, this binding can either be static or dynamic. An example of the latter is the Ethernet addressing scheme that is a 48-bit integer. As it is not possible to map 48 bit into a 32-bit IP-address without loosing information, the binding must be determined dynamically. The Addressing Resolution Protocol (ARP) is specially designed for binding Ethernet addresses dynamically to IP-addresses but can be used for other schemes as well.

1.2.3 Subnetworks

As will be explained in the section Gateways and Routing, Internet routing between gateways is based on the netid part of the IP-address. In the past few years a very large number of small networks with only a few hundred nodes have been connected to the Internet. Having so many netids makes the routing procedure complicated and time consuming. One solution to this is to introduce a subnet addressing scheme where a single IP-address spans a set of physical networks. This scheme can also be used to divide a large number of nodes into logical groups within the same network.

The scheme is standardized and described in the RFC IP Subnet Extension. The idea is basically to use three coordinates in the IP-address instead of two, that is:

```
address = (netid, subnetid, nodeid)
```

However, the subnetid only has a special meaning "behind" the front subnet gateway. The rest of the Internet can not see it and treats the subnetid and the nodeid as the hostid. Only the gateways indicated in the figure need to know of the subnets and can then make the routing accordingly.

Furthermore, the subnet hierarchy does not have to be symmetric. This is indicated in the figure where subnet 3 and 4 are subnets of subnet 2, whereas subnet 1 does not have any subnets.

A 32-bit subnet mask for each level in the subnet hierarchy is required in order to make the gateway routing possible between the subnets. This mask specifies what part of the IP-address is the subnetid and what part is the nodeid by simple boolean AND'ing.

1.2.4 Special Addresses

One advantage of having the network encoded as a part of the IP-address is that it is possible to refer to the network as well as individual hosts. Three special cases have been specifically allocated for exploiting this feature:

Broadcast Messages

It is possible to generate as broadcast message to all nodes on a network by specifying the netid and letting the hostid be all 1s. However, there is is no guarantee that the physical actually supports broadcast messages, so the feature is only an indicator. It is not possible to make a broadcast message to the whole Internet in one operation. This is to prevent the network from flooding the Internet with global broadcast messages.

This network

Situations might appear where a host on a network does not know the netid of the network that it is connected to. This happens every time a host without stationary memory wants to get on to the net. However, the host *does* know its physical address which is sufficient for communicating locally within the network. In this situation it sets the netid to 0 and sends out a broadcast message on the local network. Two Internet protocols are available for doing this:

Reverse AddressResolution Protocol (RARP) This protocol is adapted from the Address Resolution Protocol that is especially created to resolve 48-bit physical Ethernet addresses into 32-bit IP-addresses. Only a dedicated RARP server on the network will answer the reply by filling out the netid and send it back to the requester. In case the main RARP server is down a backup RARP can be chosen to perform the job.

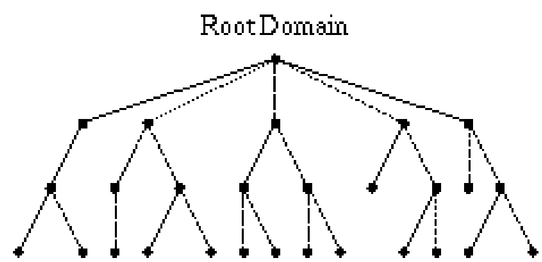
Internet ControlMessage Protocol (ICMP) This is a generic low level error and information protocol that can be used for sending error and information messages between any host gateway (also from gateway to gateway and host to host). It also has the possibility of sending out a simple *information request* message, and this can be used to obtain the netid of the network. In this situation, the gateways on the local network will respond to the request with an information message having the right netid.

Local Host

By convention the Class A address 127.0.0.1 is known as a *loopback address* for the local host. This address provides the possibility of accessing resources local to your own system. On Unix platforms, this is defined in the **/etc/hosts** system file.

1.3 Domain Name Server

This section is an introduction to the Internet Domain Name Service (DNS). See DNS and Bind for a complete description of the service. The DNS is build on top of a distributed database where every data record is indexed by a name that is a part of the *Domain Name Space*. The index itself is a hierarchically organized tree structure as illustrated in the following figure:



where the top node is called the *root domain* with the null label (empty string) but referenced as a single dot. Each node in the tree is labeled with a name consisting of at most 63 characters taken from the set of

- letters from A-Z (case insensitive)
- digits from 0-9
- hyphen

The advantage of having a hierarchical structure of the name space is that administration of the space can be delegated to different organizations without any risk for name collision. This is very important as the size of the DNS database is foreseen to be proportional to the number of users on the Internet as the database not only can contain information about hosts but also about personal mail addresses.

The structure shown above is very similar to the Unix file system. The most important difference is that a record in the DNS database is indexed from the bottom of the tree and up whereas a Unix file is indexed from the top of the tree, e.g.:

info.cern.ch The *info* is the host name and the *cern.ch* is the domain name.

/usr/local/bin/emacs *emacs* is the file name and */usr /local/bin* is the path

Another similarity is *aliases* that are pointers to the official host name in the DNS database. In the Unix file system it is implemented as (soft) links.

DNS is a client-server based application consisting of the Domain Name Servers and the resolvers. A server contains information about some segment of the DNS database and makes it available to clients or resolvers. Resolvers are often just software libraries that is linked into any Internet program by default.

In the next section it is described what happens when a host has more than one physical connection to the Internet and hence more than one host name.

1.4 Gateways and Routing

When a message is to be send from one host to another, some mechanism must provide the functionality of choosing the exact path of which the message is to be transmitted. When routing a message, two distinct situations can occur :

Direct Routing The transmitting and receiving host are connected to the same physical network

Indirect Routing The transmitting and receiving host are separated by one or many networks

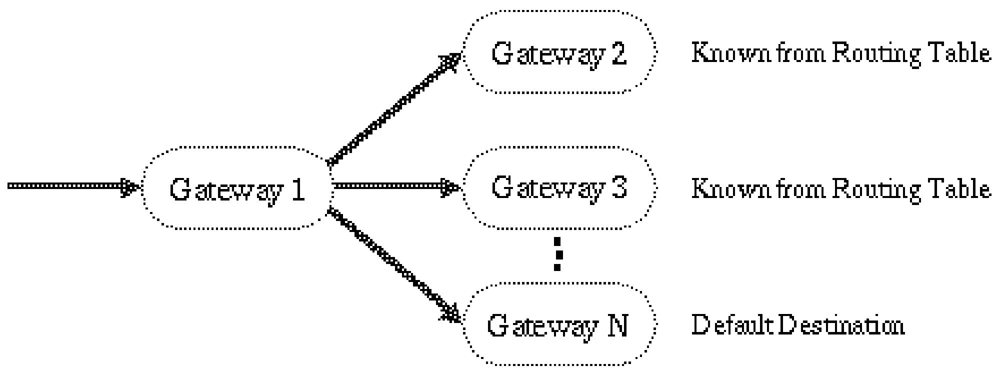
In the first case, routing is a question of resolving the IP-address into a physical address as described in Physical Addresses. Then the sender encapsulates the IP-datagram into physical frames and sends it directly to the destination. This section will give an overview of how the latter case is handled using *gateways.*

Routing does not lead to changes in the original message. The source and destination address remain the same. The source always specifies the address of the original host and the destination address is that of the destination host. The original message is instead encapsulated in another message in order to specify the *next hop* address.

The standard routing algorithm used on the Internet is based on *routing tables* situated in every gateway. An advantage of the Internet Address Scheme is that it is sufficient for a gateway to look at the netid part of the IP-address in order to find the destination network. Only the gateway directly attached to the destination network needs to look at the hostid in order to resolve the IP-address into a physical address.

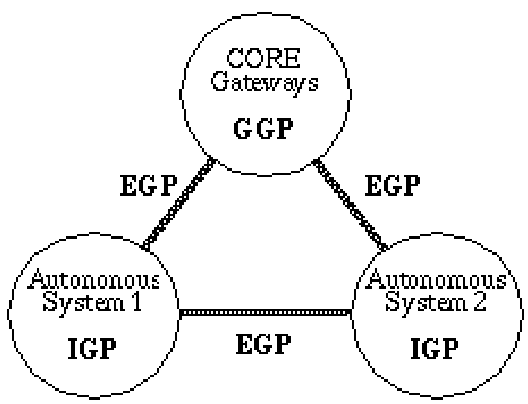
However, even if the routing tables only contains netids, it would be impossible to have routing information on every node on the Internet. The solution to this problem is to use partial routing information. The idea is to first look at the routing table to see if the netid is there. If it is not then the gateway sends the IP-datagram to a default destination as illustrated in the following figure.

As the default gateway again might send it to its default destination a mechanism must assure that the routing converge towards the final destination. This guarantee is provided by a set of core gateways that contain full routing tables. All partial routing finally ends



up in an core gateway and the message can then be directed to the right subnet of the Internet.

Until no assumptions have been made on how the gateways actually get the routing tables and how updated information gets spread throughout the Internet. There are several protocols to do this but before mentioning them, it is necessary to look into the organization of gateways on the internet.



The CORE Gateways are a small group of gateways such as the NFSNET backbone net which guarantees that the partial routing algorithm will converge towards a final destination. An autonomous systems is a set of networks organized under the same administrative authority. All routing information within the system are passed to other autonomous system via a few exterior gateways close to the outer edge of the system. The protocols indicated in the figure are shortly summarized below:

GGP Gateway-to-Gateway Protocol This is set of protocols used internally between core gateways in order to exchange updated routing tables. They are often based on the Shortest Path First (SPF) algorithm where a data base contains the complete network topology and connectivity in every gateway. Then the core gateway can compute the best shortest routing path and guarantee that it will converge.

EXP Exterior Gateway Protocol The Exterior Gateway Protocol is used to exchange routing tables between a few dedicated gateways across autonomous systems.

IGP Interior Gateway Protocol Actually this is a set of different protocols that all have the same purpose of having consistent routing tables internally in an autonomous system. The best known is Routing Information Protocol (RIP). This protocol is based on a vector distance algorithm defined as the minimum number of hops between gateways. International Standards Organization (ISO) has defined the Intermediate System-to-Intermediate System (IS-IS) as another IGP protocol. This means that OSI Networks and TCP/IP networks can share routing information.

1.5 Problems in the Internet Model

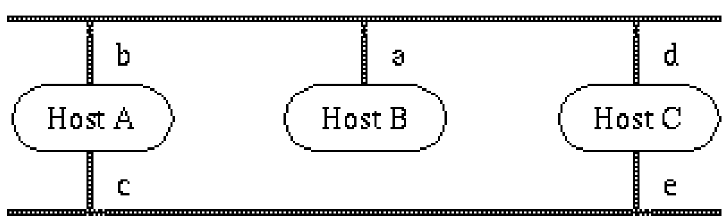
Now when the basic properties of the Internet model has been introduced some problems or weaknesses in the current model have become clear. This section will shortly summarize the most important limitations in the current Internet architecture.

1.5.1 Address Scheme

The basic disadvantage of having the netid as a part of the IP-address is that if a host is moved from one place to another it must have a new IP-address. As more and more portable computers are connected to the Internet this has turned out to be a real problem in the address scheme.

1.5.2 Routing

When routing is based on the netid of the IP-address multi-homed hosts might have a significant difference in access time dependent of the IP-address used.



If **Host B** in the figure wants to communicate with **Host C** in the figure and chooses the IP-address of **node e** then the message have to go through **Host A**. Unless the system administrators explicitly have told the local part of the Domain Name Service to return the IP-address of **node d** there is no way for **Host B** to know the optimal route.

1.5.3 Security

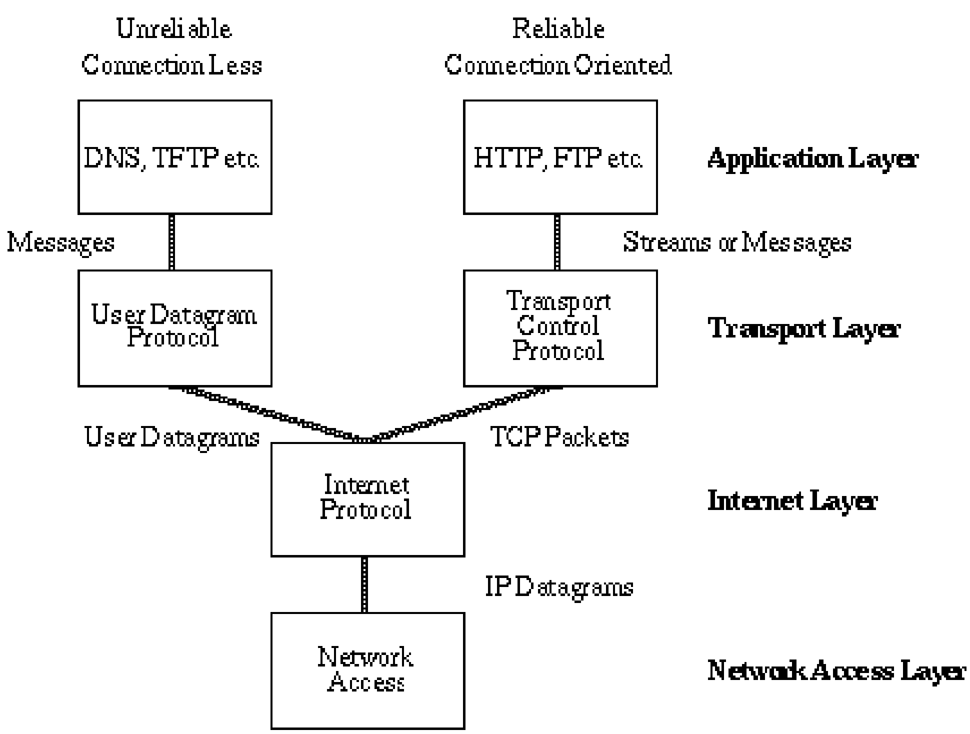
Another important aspect not described here is *security considerations* when using the Internet. What means do people have to gain access to classified information when communicating to Internet sites. Today security precautions on the Internet is often based on the assumption that the transport service provided by the Internet can be considered as a trusted carrier. This is equivalent to the generally accepted assumption that letters send via the public postal system is actually delivered to the addressee without being read by anyone during transportation.

This is, however, not true on the Internet and many problems have arisen simply from people listening to the net traffic. Especially protocols like FTP and the Telnet protocol (the control connection in the FTP protocol is actually a telnet connection) have proven to be very insecure as passwords are transmitted unencoded across the Internet.

Chapter 2

TheInternet Protocol Stack

As mentioned in the Internet Section the Internet is an abstraction from the underlying network technologies and physical address resolution. This section introduces the basic components of the Internet protocol stack and relates the stack to the ISO OSI reference protocol stack model. The model of the Internet protocol stack is illustrated in the figure below.



This documents describes the various parts presented in this diagram. The upper layer protocols, e.g., FTP, Telnet, TFTP etc. are described in the Presentation Layer Protocol section. This leaves the following topics as sections in this document:

1. Internet Protocol (IP)

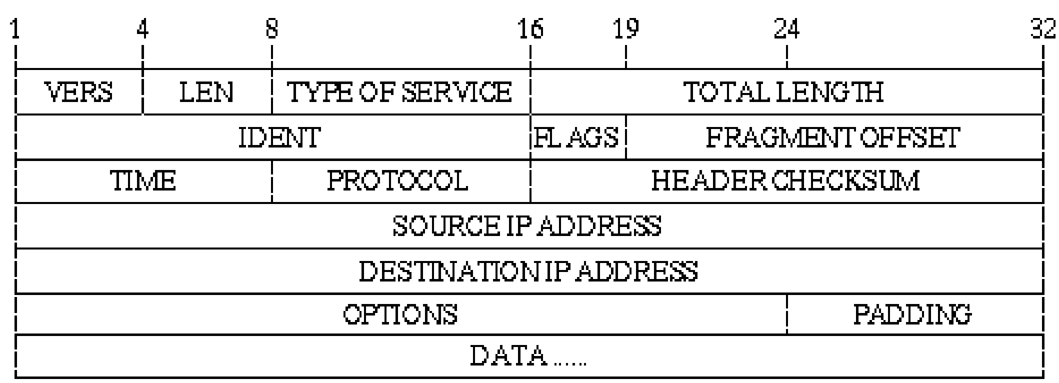
2. User Datagram Protocol (UDP)
3. Transmission Control Protocol (TCP)
4. Transactional Transmission Control Protocol (T/TCP)
5. TCP/IP and OSI/RM

2.1 Internet Protocol (IP)

As seen in the figure above, the Internet protocol stack provides a connection oriented reliable branch (TCP) and an connectionless unreliable branch (UDP) both build on top of the Internet Protocol.

The Internet Protocol layer in the TCP/IP protocol stack is the first layer that introduces the virtual network abstraction that is the basic principle of the Internet model. All physical implementation details (ideally even though this is not quite true) are hidden below the IP layer. The IP layer provides an unreliable, connectionless delivery system. The reason why it is *unreliable* stem from the fact the protocol does not provide any functionality for error recovering for datagrams that are either duplicated, lost or arrive to the remote host in another order than they are send. If no such errors occur in the physical layer, the IP protocol guarantees that the transmission is terminated successfully.

The basic unit of data exchange in the IP layer is the Internet Datagram. The format of an IP datagram and a short description of the most important fields are included below:



LEN The number of 32 bit-segments in the IP header. Without any OPTIONS, this value is 5

TYPE OF SERVICE Each IP datagram can be given a precedence value ranging from 0-7 showing the importance of the datagram. This is to allow *out-of-band* data to be routed faster than normal data. This is very important as Internet Control Message Protocol (ICMP) messages travels as the data part of an IP datagram. Even though an ICMP message is encapsulated in a IP datagram, the ICMP protocol is normally thought of as a integral part of the IP layer and not the UDP or TCP layer. Furthermore, the TYPE OF SERVICE field allows a classification of the datagram in order to specify is the service desired requires short delay time, high reliability or high throughput. However, in order for this to have any effect, the gateways must know more than one route to the remote host and as described in the Introduction, this is not the case.

IDENT, FLAGS, and FRAGMENT OFFSET These fields are used to describe fragmentation of a datagram. The actual length of an IP datagram is in principle independent of the length of the physical frames being transferred on the network, referred to as the network's *Maximum Transfer Unit (MTU)*. If a datagram is longer than the MTU then it is divided in to a set of fragments having almost the same header as the original datagram but only the amount of data that fits into a physical frame. The IDENT flag is used to identify segments belonging to the same datagram, and the FRAGMENT OFFSET is the relative position of the fragment within the original datagram. Once a datagram is fragmented it stays like that until it receives the final destination. If one or more segments are lost or erroneous the whole datagram is discarded.

However, the underlying network technology is not completely hidden below the IP layer in spite of the fragmentation functionality. The reason is that the MTU can vary from 128 or less to several thousands of bytes dependent of the physical network (Ethernet has a MTU of 1500 bytes). It is hence question of efficiency when choosing the right datagram size so that fragmentation is minimized. It is recommended that gateways are capable of handling datagrams of at least 576 bytes without having to use fragmentation.

TIME This is the remaining *Time To Live (TTL)* for a datagram when it travels on the Internet. The Routing Information Protocol (RIP) specifies that at most 15 hops are allowed.

SOURCE IP-ADDRESS and DESTINATION IP-ADDRESS Both the source and destination address is indicated in the datagram header so that the recipient can send an answer back to the transmitting host. However, note that only the host address is specified - not the port number. This is because the IP protocol is an IMP-to-IMP protocol - it is *not* an end-to-end protocol. A layer more is needed to actually specify which two processes on the transmitting host and the final destination that should receive the datagrams.

Note that the IP-datagram only leaves space for the original source IP-address and the original destination IP-addrss. As mentioned in the section Gateways and Routing the *next hop* address is specified by encapsulation. The **Internet Layer** passes the IP-addres of the *next hop* address to the **Network Layer**. This IP-address is bound to a physical address and a new frame is formed with this address. The rest of the original frame is then encapsulated in the new frame before it is send over the communication channel.

2.2 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a very thin protocol build on top of the Internet Protocol. The basic unit of data is a *User datagram* and the UDP protocol provides the same unreliable, connectionless service transferring user datagrams as the IP protocol does transferring its datagrams. The main difference is that the UDP protocol is an *end-to-end protocol*. That is, it contains enough information to transfer a user datagram from one process on the transmitting host to another process on the receiving host. The format of a user datagram is illustrated below:

1	16	32
SOURCE PORT	DESTINATION PORT	
LENGTH	UDP CHECKSUM	

The LENGTH field is the length of the user datagram including the header, that is the minimum value of LENGTH is 8 bytes. The SOURCE PORT and DESTINATION PORT are the connection between a IP-address and a process running on a host. A network port is normally identified by an integer. However, the user datagram does not contain any IP-address so how does the UDP protocol know when the final destination is reached?

When calculating the CHECKSUM header, the UDP protocol appends a 12-byte pseudo header consisting of the SOURCE IP-ADDRESS, the DESTINATION IP-ADDRESS and some additional fields. When a host receives a UDP datagram it takes the UDP header and creates a new pseudo header using its own IP-address as the DESTINATION IP-ADDRESS and the SOURCE IP-ADDRESS extracted from the IP datagram. Then it calculates a checksum and if it equals the UDP checksum, then the datagram has received the final destination.

As indicated in the Internet Protocol Stack Figure the UDP protocol is often used as the basic protocol in client-server application protocols such as TFTP, DNS etc. where the overhead of making a reliable, connection oriented transmission is considerable. This problem will be considered further in the next two sections.

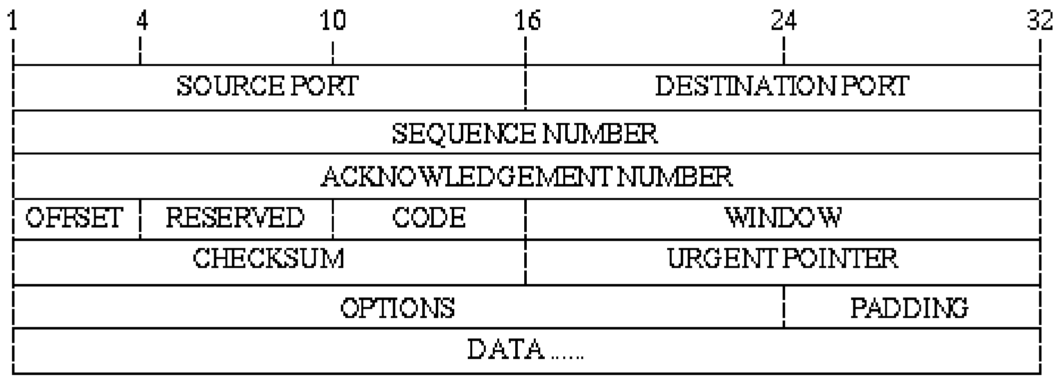
2.3 Transmission Control Protocol (TCP)

The Transmission Control Protocol provides a full duplex, reliable, connection oriented service to the application layer as indicated in the Internet Protocol Stack Figure. This section described the basic principle of the TCP protocol and how it provides a reliable service to the application layer protocols.

The TCP protocol is a stream oriented protocol. It is designed to provide the application layer software with a service to transfer large amount of data in a reliable way. It establishes a full duplex virtual circuit between the two transmitting hosts so that both host simultaneously can put data out on the Internet without specifying the destination host once the connection is established. In the Transactional Transmission Control Protocol (T/TCP) section an client-server based extension to the TCP protocol is presented as an alternative to the stream architecture.

2.3.1 TCP Segment Format

A segment is the basic data unit in the TCP protocol. As much of the following sections are based on this data unit, the format is presented here:



SOURCE PORT, DESTINATION PORT The TCP protocol uses the same trick of using a pseudo header instead of transmitting the source IP-address and the destination IP-address as is already included in the IP-datagram. Therefore only the port numbers are required to uniquely define the communicating hosts.

CODE This field is used to indicate the content of the segment and if a specific action has to be taken such as if the sender has reached EOF in the stream.

OPTIONS The TCP protocol uses the OPTIONS field to exchange information like *maximum segment size accepted* between the TCP layers on the two hosts. The flags currently defined are

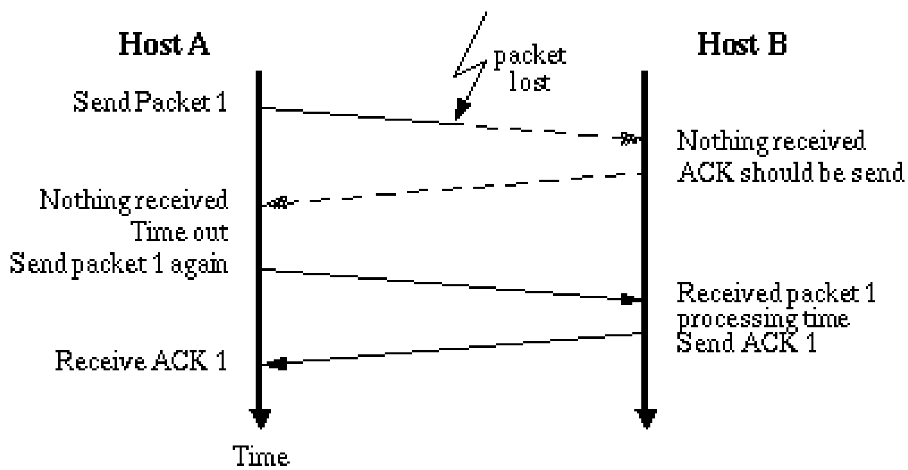
- **URG** Urgent pointer field is valid
- **ACK** Acknowledgement field is valid
- **PSH** This segment requests a push
- **RST** Reset the connection
- **SYN** Synchronize sequence numbers
- **FIN** Sender has reached end of its byte stream

OFFSET This integer indicates the offset of the user data within the segment. This field is only required as the number of bits used in the OPTIONS field can vary

URGENT POINTER This field can be initialized to point to a place in the user data where urgent information such as escape codes etc. are placed. Then the receiving host can process this part immediately when it receives the segment.

2.3.2 Reliable Transmission

At the IP-protocol layer packets can get discarded due to network congestion, noise gateway failure etc. In order to provide a reliable service, the TCP must recover from data that is damaged, lost , duplicated, or delivered out of order by the Internet communication system. This is achieved by assigning a SEQUENCE NUMBER to each byte transmitted, and requiring a *positive acknowledgment (ACK)* from the receiving host. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments. The principle is illustrated in the figure below:

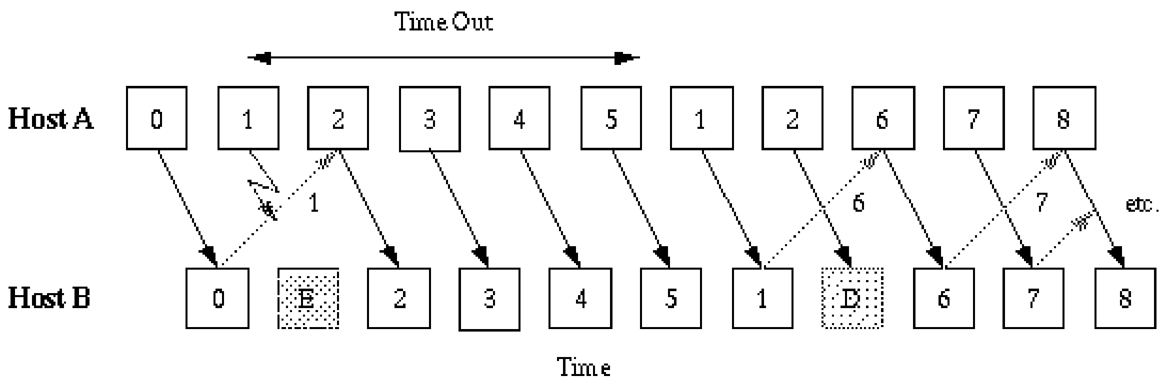


Host A is transmitting a packet of data to **Host B**, but the packet gets lost before it reaches its destination. However, **Host A** has set up a timer when to expect the ACK from **Host B** so when this timer runs out, the packet is retransmitted. The difficult part of the method is to find a value of the time out period as a TCP segment can travel through different speed networks with different loads. This means that the *Round trip Time (RTT)* can vary from segment to segment. A simple way of calculating the RTT is by using a recursive mean value with an exponential window to decrease the importance of old values.

As mentioned in the introduction to the TCP section, the protocol is a stream oriented protocol. It uses unstructured streams with no method of indexing the user data, e.g. as records etc. Furthermore, the length of a TCP segment can vary as is the case for the IP-datagram and the UDP user datagram. Therefore the acknowledgement can not be based on the segment number but must be based on bytes successfully transferred.

However, the PAR principle is very inefficient as the sending host must await the acknowledgement before it can send the next segment. This means that the the minimum time between two segments is 1 RTT plus the time required to serve the segments at both ends. The TCP protocol solves this by using sliding windows at both ends.

This method permits the transmitting host to send as many bytes as can be stored in the sending window and then wait for acknowledgements as the remote host receives the segments and sends data in the other direction. The acknowledgement send back is cumulative so that it at all times shows the *next byte* that the receiving host expects to see. An example with a large window size and selective retransmission is shown in the figure:



Byte number 1 is lost so **Host B** never sends back a positive acknowledgement. When **Host A** times out on byte 1 it retransmit it. However, as the rest of the bytes from 2-5 are transmitted successfully the next acknowledgement can immediately jump to 6 which is the next expected byte. Byte 2 is also retransmitted as **Host A** does not know exactly how many bytes are erroneous. **Host B** just discards byte 2 as it already is downloaded.

The window technique can also be used to provide a congestion control mechanism. As indicated in the TCP Segment Format Figure every segment has a WINDOW field that specifies how much data a host is willing to receive. If the host is heavyly loaded, it can decrease the WINDOW parameter and hence the transmission speed drops.

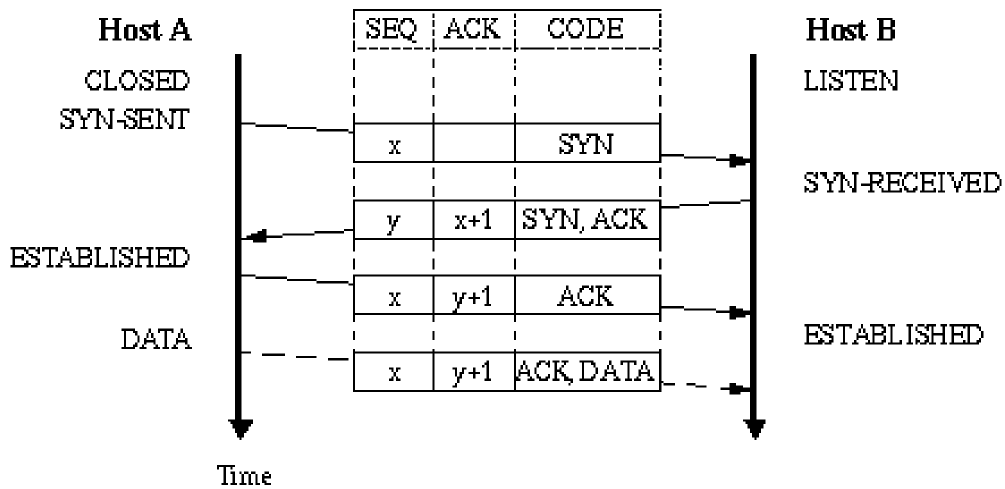
However, as the TCP protocol is an end-to-end protocol it can not see if a congestion problem has occurred in an intermediate *Interface Message Processor (IMP)* (often called a *packet switched node*) and hence, it has no means to control it by adjusting the window size. TCP solves this problem by using the Internet Control Message Protocol (ICMP) source quench messages.

2.3.3 Connection Establishment

When a TCP connection is to be opened a 3-way handshake (3WHS) is used in order to establish the virtual circuit that exists until the connection is closed at the end of the data transfer. The 3WHS is described in the following as it is an important part of the TCP protocol but also shows some inefficiencies in the protocol. The principle of a 3WHS is illustrated in the figure below:

The blocks in the middle symbolizes the relevant part of the TCP segment, that is the SEQUENCE NUMBER, the ACKNOWLEDGEMENT NUMBER and the CODE. The active **Host A** sends a segment indicating that it starts its SEQUENCE NUMBER from x. **Host B** replies with an ACK and indicates that it starts with SEQUENCE NUMBER y. On the third segment both hosts agree on the sequence numbers and that they are ready to transmit data.

In the figure only **Host A** does an active open. Actually the two hosts can do a simultaneously open in which case both hosts perform a SYN-RECEIVED and then synchronize accordingly. The principle reason for the 3WHS is to prevent old duplicate connection initiations from causing confusion.



Note that the SEQUENCE NUMBER of segment 3 and 4 is the same because the ACK does not occupy sequence number space (if it did, the protocol would wind up ACKing ACK's!).

However, the TCP connection establishment is somewhat long cumbersome in many applications, especially in the client-server applications such as the World-Wide Web. In the next section an alternative having a lighter connection establishment is presented.

2.4 Transactional Transmission Control Protocol (T/TCP)

The TCP protocol is a highly symmetric protocol in that both hosts can transmit and receive data simultaneously. However, not all applications are symmetrical by nature. A typical example is a client-server protocol such as the Domain Name Service. The Transactional Transmission Control Protocol (T/TCP) that is a very new protocol (July 1994) offers an alternative to TCP when high performance is required in client-server applications. Some of the requirements of an high performance transaction oriented protocol are listed below:

- The interaction between the client and the server is based on a request followed by a response, that is a stateless approach.
- The protocol must guarantee that a transaction is carried out at most one time and any duplicate packets received by the server should be discarded.
- No explicit open or close procedure of the connection. This is opposite to TCP and the 3WHS as described above.
- The minimum transaction latency for a client should be *Round Trip Time (RTT)* + *Server Processing Time (SPT)*. That is basically the same requirement as no explicit open or close procedure.
- The protocol should be able to handle a reliable minimum transaction of exactly 1 segment in both directions.

This section describes how the TTCP protocol deals with these requirements and also that might affect the World-Wide Web model with respect to performance.

2.4.1 Implicit Connection Establishment

The T/TCP protocol is as indicated by the name based on the TCP protocol and T/TCP is backwards compatible with TCP. However, one of the features of the T/TCP protocol is that it can bypass the 3WHS described in the previous section but in case of failure can resolve to the 3WHS procedure.

The 3HWS has been introduced in order to prevent old duplicate connection initiations from causing confusion. However, T/TCP provides an alternative to this by introducing three new parameters in the OPTION field in the TCP Segment:

CONNECTION COUNT (CC) This is a 32-bit incarnation number where a distinct value is assigned to all segments send from **Host A** to **Host B** and another distinct number the other way. The kernel on both hosts keeps a cache of all the CC numbers currently used by connections to remote hosts. On every new connection the client CC number is monotonically incremented by 1 so that a segment belonging to a new connection can be separated from old duplicates from previous connections.

CONNECTION COUNT NEW (CC.NEW) In some situations, the principle of a monotonically increasing value of CC can be violated, either due to a host crash or that the maximum number, that is 4G, is reached and the counter returns to 0. This is possible in practice because the same CC number is global to all connections. In this situation a CC.NEW is send and the the remote host resets its cache and returns to a normal 3WHS TCP connection establishment. This signal will always be send *from* the client and *to* the server.

CONNECTION COUNT ECHO (CC.ECHO) In the server response the CC.ECHO field contains the CC value send by the client so that the client can validate the response as belonging to a specific transaction.

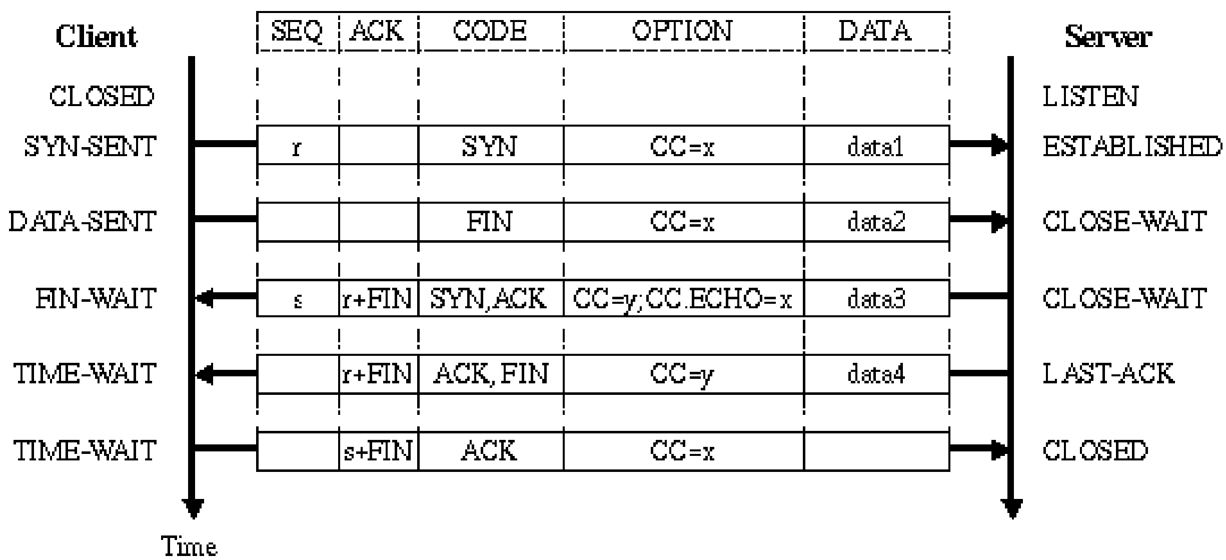
The bypass of the 3WHS is illustrated in the following figure:

In the example, two segments are send in both directions. The connection is established when the first segment reaches the server. The client is left in a TIME-WAIT state which is explained in the next section.

2.4.2 Connection Shotdown

Every TCP or UDP connection between two hosts are uniquely identified by the following 5-tuple:

- Protocol (UDP, TCP)
- IP-address of Host A



- Port number of Host A
- IP-address of Host B
- Port number of Host B

Whenever a TCP connection has been closed, the association described by the 5-tuple enters a wait state to assure that both hosts have received the final acknowledgement from the closing procedure. The time of the wait state is called TIME-WAIT and is by default 2*MSL (120 seconds) where MSL is the Maximum Segment LifeTime. That is, two hosts can not perform an new transaction using the same 5-tuple until at least 120 seconds after the previous connection has been terminated. One way to circumvent this problem is to select another 5-tuple but as mentioned in Extending TCP for Transactions – Concepts this does not scale due to the excessive amount of kernel space occupied by terminated TCP connections hanging around.

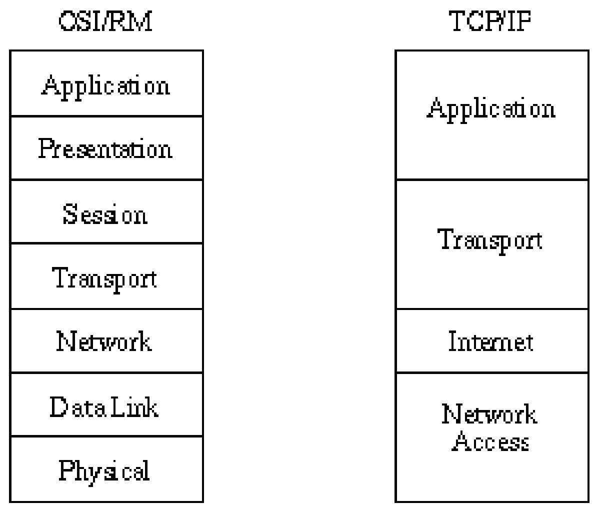
However, the T/RCP CC numbers gives a unique identification of each transaction so the T/TCP protocol is capable of truncating the WAIT-STATE by comparing the CC numbers. This principle can be looked at as expanding the state machine of one transaction to also include information on previous and future transactions using the same 5-tuple.

2.4.3 TTCP and the World-Wide Web

As will be shown in the description of World-Wide Web of this thesis, the principle of the World-Wide Web is a transaction oriented exchange of data object. This is the reason why the T/TCP protocol is very interesting in this prospective.

2.5 TCP/IP and OSI/RM

International Standards Organization (ISO) has designed the second dominating protocol layering scheme, called ISO *Open System Interconnection Reference Model (OSI/RM)*. This section presents the OSI reference model and compares it to the TCP/IP protocol stack as illustrated in the figure.



Physical Layer Specifies the physical connection between the host computers and IMPs and how bits are transferred over a communication channel.

Data Link Layer This layer specifies how data travels between IMPs using *frames*. Its maintask is to change the service from the physical layer into a packet oriented error-free transmission.

Network Layer The frames from the Data Link Layer are organized into *packets* and directed through the network. The communication is still between IMPs.

Transport Layer The first layer that provides an end-to-end transport service. It ensures that the transferred data arrive correctly at the other end.

Session Layer This layer specifies how two hosts can establish sessions where data can be transferred in both directions on a virtual connection between the two hosts.

Presentation Layer The Presentation layer introduces a set of syntax and semantics of the information transmitted through the lower protocol layers.

Application Layer This layer defines a platform independent virtual network terminal so that application programs can exchange data regardless of the internal data representation used.

Even though OSI/RM and TCP/IP can be compared like this, there still exist several significant differences between OSI/RM and the TCP/IP protocol stack but on the the most fundamental is that OSI/RM is a *standardized model* for how the functionality of a protocol stack can be organized. It doesn't specify the exact services and protocols to be used in each layer whereas the TCP/IP is an result of experimental research. In spite of this, the OSI /RM model has been the basis of several protocol stack implementation such as X.25, discussed in A Critique of X.25

Another difference is where the intelligence is placed in the layering. OSR /RM introduces a reliable service on the Data Link Layer whereas the TCP/IP only has intelligence in the Transport Layer. Both solutions have advantages and disadvantages. When a reliable data transfer service is placed in the lower layers the clients using the network for communication can be kept very simple as they do not have to handle complicated error situations. The disadvantage is that performance decreases due to an excessive amount of control information transferred and processed in every host.

Chapter 3

Presentation Layer Protocols

This section introduces some of the Presentation Layer protocols on the Internet that are related to the World-Wide Web project. The main WWW protocol, Hypertext Transfer Protocol is described in the The HTTP Protocol. The protocols presented are:

1. Multipurpose Internet Mail Extensions
2. File Transfer Protocol
3. Network News Transfer Protocol

3.1 Multipurpose Internet Mail Extensions

The Multipurpose Internet Mail Extensions (MIME) protocol is an extension to the Standard for the Format of ARPA Internet Text Messages. This protocol has defined the standard format of textual mail messages on the Internet since it came out in 1982. It describes the format of message *headers* but it tells little about the content of the *body* of the message which is limited to 7-bit ASCII characters. The MIME protocol provides the necessary extension to the MAIL protocol in order to transfer possibly multi-part textual and non-textual data object in the body of a MAIL message.

The protocol basically specifies a set of header lines that together with a set of *name-value* pairs. The reason for describing the MIME protocol in this document is that it is an important part of the Hypertext Transfer Protocol (HTTP) described later. In the following, the most important header lines are introduced:

3.1.1 Content Type

The content type header field is a set of types and subtypes that specifies the content of the body part of a MAIL message. The protocol specifies 7 content types and a large set of subtypes, specified in the header as

```
Content-Type: <type> / <subtype> *(";" parameter)
```

The idea behind this format is to let MIME compliant software know the maintype of a data object even though it might not be able to handle the specific subtype. The main types are:

Text Used to represent textual data objects. The character set used can be specified as a parameter and can in be any ASCII or ISO character set.

Multipart This type does not describe the specific content of the body but allows the body to consist of several body parts into the same message. Each body can then have its own content-type that again might be a multi-part message. The multi-part content type provides the possibility of a hierarchical data object structure in MIME conforming messages.

Application This content type describes binary data in some form. In practice it is often used to describe data objects that a MIME frontend does not know how to handle otherwise. The default action taken is to dump the content to, e.g., a local file to allow further processing by other processes.

Message It is often desirable to encapsulate a MAIL message into another, e.g. when forwarding a message to a new recipient. The Message content type has been defined for this purpose.

Image This content type describes still images and pictures

Audio Used for audio or voice data objects

Video Used for transmitting video frames or moving image data. The current subtype is *mpeg*

The last three content-types all have a vast amount of subtypes as the number of graphical and audio data formats is very big. However the content type allows every MIME conforming frontend to handle it in an intelligent manner.

3.1.2 Content Transfer Encoding

The content transfer encoding describes the encoding or merely the format of the data being transferred in the message. Three special transfer encodings that all mean *No encoding* are defined as:

7bit The body consists of short lines separated by the <CRLF> (carriage return; Line Feed) sequence. All characters can be represented in a 7-bit character set. The Simple Mail Transfer Protocol defines a short line as maximum 1000 characters including the <CRLF>.

8bit The body consists of short lines separated by the <CRLF> (carriage return; Line Feed) sequence. All characters can be represented in a 8-bit character set.

binary The body is a byte stream without any specific line separation. All characters can be represented in a 8-bit character set.

The content type can also specify a special encoding of, e.g., a 8-bit message into a 7-bit so that it can transferred over a 7-bit communication channel.

3.2 File Transfer Protocol

The File Transfer Protocol is one of the best known protocols on the Internet. It has a highly state oriented structure using a control channel using the Telnet Protocol and a data channel for each data transfer. A FTP-session is characterized by having an excessive amount of control information exchange. This makes it very expensive to exchange small data objects as the relative part of control information is high.

Even though the FTP protocol is session oriented, the FTP address space is completely flat. This means that no guarantee is given for the user to be able to return to the same point in the address space without closing the session and start all over again.

FTP clients and servers tend to be implemented as minimum solutions which only provides the absolute minimum service to the user.

3.3 Network News Transfer Protocol

The Network News Transfer Protocol (NNTP) is used for the destribution, inquery, retrieval, and posting of news articles. The protocol is based on the TCP protocol and is a point-to-multi point protocol where information is spread like an avalanche. This is the oposite of the World-Wide Web princip where information is spread using a point-to-point method.

The NNTP protocol is client-server based in that news articles are stored on NNTP servers repeatedly getting updated when communicating to other NNTP servers. A client gets desired information spread via NNTP by requesting a local NNTP server.

Chapter 4

TheWorld-Wide Web

The official description of the World-Wide Web (WWW, W3) is a "wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents". It is a way of viewing all the on-line information available on the Internet as a seamless, browsable continuum. This section introduces the general concept of the World-Wide Web and the basic elements in the model. The content of the section is as follows:

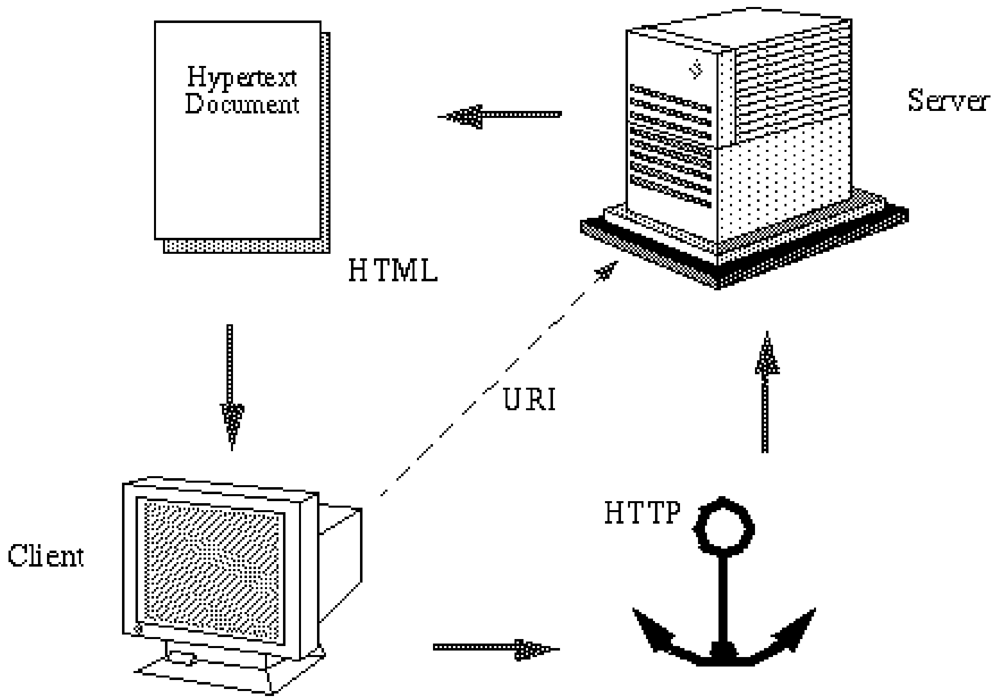
1. Basic World-Wide Web Model
2. Universal Resource Identifies
3. Hypertext Transfer Protocol
4. Hypertext Markup Language
5. Interactive World-Wide Web Model

4.1 Basic World-Wide Web Model

The basic idea behind the World-Wide Web is based on a client server application and hypertext documents as illustrated in the figure below. The model is simplified in that it only contains elements that exists within the World-Wide Web concept. Later the model will be expanded to a generic resource accessing model.

The Client The client is the user's interface to the Internet. Whatever type of service requested this interface stays the same, so users do not need to understand the differences between the many different access schemes in common use on the Internet. This principle is the same as is seen from other popular applications such as Microsoft Windows , Machintosh etc. where the user is always presented to the same GUI interface.

Uniform ResourceIdentifier URI The user initiates a request by specifying a Uniform Resource Identifier or a "hyperlink". This link can specify any accessible information or resource on the Internet as long as it can be uniquely identified as an



object. The word "Web" refers to the combination of accessible objects and the links pointing to them throughout the Internet.

TheServer The server is responsible for handling the request sent from the client. This can either be a local accessible resource or the server can request the resource from another server in which case the first server temporarily turns into a client.

HypertextTransport Protocol HTTP The client sends of the user request to a WWW server using the Hypertext Transfer Protocol (HTTP). This is a typical client-server application based on a stateless connection between the client requesting the URI and the server handling the request.

Hypertext Markup Language HTML On a successful request, a data object is returned from the server to the client. The object is written in the Hypertext Markup Language (HTML) which is a hypertext language with the possibility of containing hyperlinks that the user can follow.

The model basically reflects the first version of the World-Wide Web as it is described in the HTTP Protocol version 0.9 and HTML version 1.0. However, the WWW specifications have been rapidly changing during the last 3-4 years, even though the current model is still based on a client-server approach. From being a HTML and HTTP based model, the World-Wide Web is now capable of handling virtually any existing data format on the Internet using a large set of access methods apart from HTTP such as FTP, Gopher, WAIS, Telnet etc. In other words, the World-Wide Web represents a generic information exchange tool capable of accessing information throughout the Internet. Though, before the more advanced model is presented, it is necessary to get an overview of the basic elements in the WWW model mentioned above.

4.2 Universal Resource Identifies

In order to address a data object or more general, a resource, in the model above it is necessary to define a name space that not only contains information about hosts but also about resources available on each host. The World-Wide Web model defines Uniform Resource Identifiers or URIs that specifies a syntax for encoding the names and addresses of data objects on the Internet and how they can be accessed. The set of URIs covers

Universal Resource Identifier (URI) A generic set of all addresses in the address space of all resources on the Internet. They describe a hierarchical naming scheme that together with the HTTP protocol makes a significant difference between the World-Wide Web model and other Internet access schemes such as FTP that has a flat address space.

Uniform Resource Locator (URL) The term "URI" has been introduced by the IETF and is a a general description of all URL that are not persistent. In practice the URLs consist of the current set of Internet protocols supported by the WWW, i.e, HTTP, FTP, Gopher, WAIS, etc., followed by a directory path, a file name, and possibly a search directive.

Uniform Resource Name (URN) However, the ultimate goal for URIs is to be a persistent naming scheme independent of the mean of access, i.e., the protocol used and of the physical structure of resources on the specific host. The only way to obtain this is to have a naming scheme like the Internet Domain Name Service. URNs are currently under consideration in IETF but little is known about the status of the research.

Uniform Resource Citation (URC) This is meta information about a URI. They consist of pairs of attribute/value which can contain information on the author, publisher etc. The URC are currently not used.

4.3 Hypertext Transfer Protocol

The Hypertext Transport Protocol (HTTP) is a generic stateless presentation layer protocol with elements from other Internet presentation layer protocols. The HTTP protocol is built on a client-server model where the client initiates a request and the server replies with a response.

The basic format of the HTTP protocol is based on the MIME Protocol with a set of HTTP Headers possibly followed by a message body containing a data object in any 7-bit or 8-bit accepted by the client. The client specifies what data format it can handle by having a list of accept headers in the request.

The basic WWW-model indicates that the client initiates a request and the server responds by sending a data object to the client. However, often the client wants to post a data object to the server, e.g. to post a mail message to an email address, to a news group, or to create a new file on the remote server. The HTTP protocol provides two methods for the client to transfer a data object to the server. Though, the client is not guaranteed that the request can be fulfilled - even on a successful return code. The action can at all times be cancelled by the responsible person of the remote server.

One of the characteristics of the HTTP protocol is that it is a superset of the other Presentation Layer supported by the WWW-model. This means that messages generated by other protocols can be handled by the HTTP protocol by wrapping a set of HTTP/MIME headers around the message. This is an essential feature for the concept of Proxy servers.

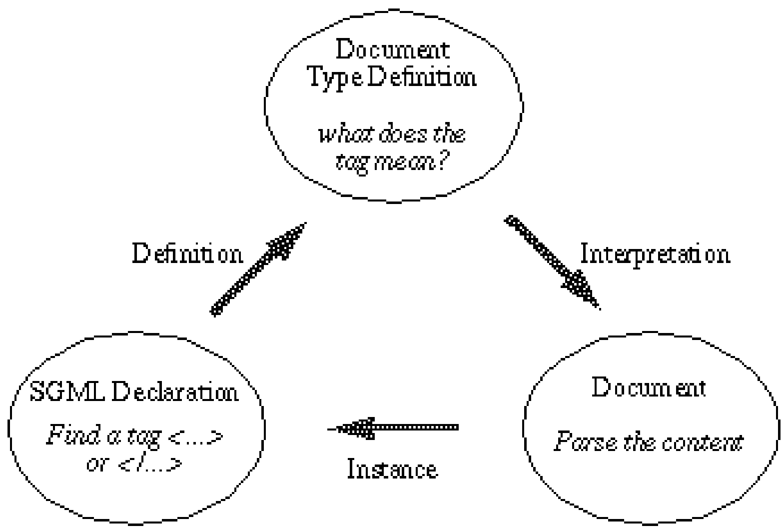
The current version 1.0 of the HTTP protocol is built on top of the TCP Protocol that is a connection oriented protocol with a 3 way handshake connection establishment. This causes an substantial overhead in a client-server oriented environment like the HTTP protocol. It would therefore be an excessive optimization if the HTTP protocol was moved to a lighter Transport Layer protocol such as the Transactional TCP Protocol which still provides a reliable stream transport service.

4.4 Hypertext Markup Language

The Hypertext Markup language (HTML) is the users interface to create information on the World-Wide Web. The description of the World-Wide Web has until now focused on the technology that due to specifications and conventions provide the functionality necessary to request and serve information across the Internet. HTML is defined to be the hypertext language of communication which actually flows over the network. There is no requirement that files are stored in HTML. Servers may store files in any other formats and then generate a HTML on the fly upon a client request. This gives the possibility of having virtual documents instead of static documents on rapidly changing information like weather reports etc. HTML can be used to represent:

- Hypertext news, mail, online documentation, and collaborative hypermedia
- Menus and options
- Database query results
- Simple structured documents with inlined multi media elements like images, audio and movie
- URI-Links to other resources on the Internet.

HTML is built on top of the International Standard ISO 8879 Standard Generalized Markup Language (SGML). SGML is a system for defining structured document types and markup languages to represent instances of the document types. That is, HTML is a Document Type Definition (DTD) used on top of a SGML parser. Every SGML based document contains three elements as illustrated in the figure:

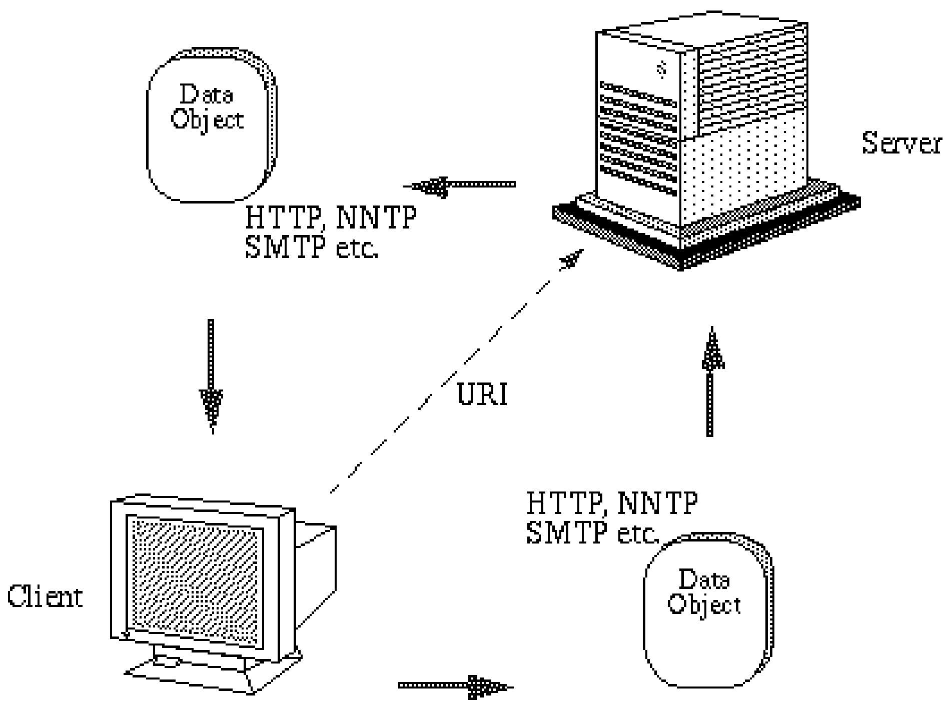


HTML is now superseeded by HTML+ that is an enriched DTD with possibilities of handling tables, math, images etc. Currently many browsers support a subset of the HTML+ specifications in addition to the basic HTML features.

4.5 Interactive World-Wide Web Model

The description of the Universal Resource Identifiers, the Hypertext Transfer Protocol, and the Hypertext Markup Language now calls for an update of the Basic WWW-model as illustrated in the figure.

This model is a generic resource exchange model based on a client-server concept. Instead of the limited model with data flowing only from the server to the client, the client is capable of posting data to the server if the server allows this kind of service. Furthermore, the format of the data transferred in the message body can have any format from 7-bit ASCII text to 8-bit binary data. The transfer carrier can be any protocol supported by the World-Wide Web but the main protocol is HTTP as it can be used to encapsulate the other protocols supported, even a FTP message that is a highly state dependent protocol.


Data
Object
Server
HTTP, NNTP
SMTP etc.
URI
HTTP, NNTP
SMTP etc.
Client
Data
Object

Chapter 5

World-Wide Web Software at CERN

This document is an overview of the World-Wide Web software developed at CERN. The development of the World-Wide Web code was started by Tim Berners-Lee in 1990. Ever since the code has been subject for changes due to modifications in the architectural model, additions of new features etc. The code is freely available as public domain software with a very mild Copyright statement.

During the last two years more and more World-Wide Web applications have become available from a large amount of software providers on almost every platform connected to the Internet. Many of them are based on the same architectural model as the CERN software but with additional functionality and increased performance. Most of the software is characterized by being freely available as public domain for educational institutions and other non-profit organizations whereas commercial companies must pay a fee for using them.

The CERN World-Wide Web software is written in plain C and is especially designed to be used on a large set of different platforms. It has often been discussed if the CERN code or especially the Common Code Library should be rewritten in ANSI C using the IEEE Std 1003.1-1990 standard (commonly referred to as POSIX.1), but eventually this will limit the portability on many platforms currently supported. A newly started collaboration between WWW software providers will expand this portability to also include MS-DOS and MacIntosh so that the most popular platforms are covered from large computers down to PCs. The document describes the following software product maintained at CERN:

1. The Library of Common Code
2. The Line Mode Browser
3. The HTTP Server
4. The Proxy Server

5.1 The Library of Common Code

The CERN World-Wide Web Library of Common Code is a general code base that can be used to build World-Wide Web clients and servers. It contains code for accessing HTTP, FTP, Gopher, News, WAIS, Telnet servers, and the local file system. Furthermore it provides modules for parsing, managing and presenting hypertext objects to the user and a wide spectra of generic programming utilities. The Library is the basis of many World-Wide Web applications and all the CERN WWW software is built on top of it. Even though it is written in plain C, many of the data structures used are highly object oriented - an implementation form often referred to as "a poor man's C++". The following figure is an overview of the current architecture of the library. The view is especially for the client side of the library. The CERN Proxy Server and the CERN HTTP server have slightly different views of the architecture.

The flow of the library shows that all network communication and data object parsing is handled internally. Only the presentation to the user is left to the client as this is a very platform dependent task. The main elements in the figure are explained in the following sections. A more specific description of the implementation of the library is given in Internals and Programmer's guide

5.1.1 Graphic Object

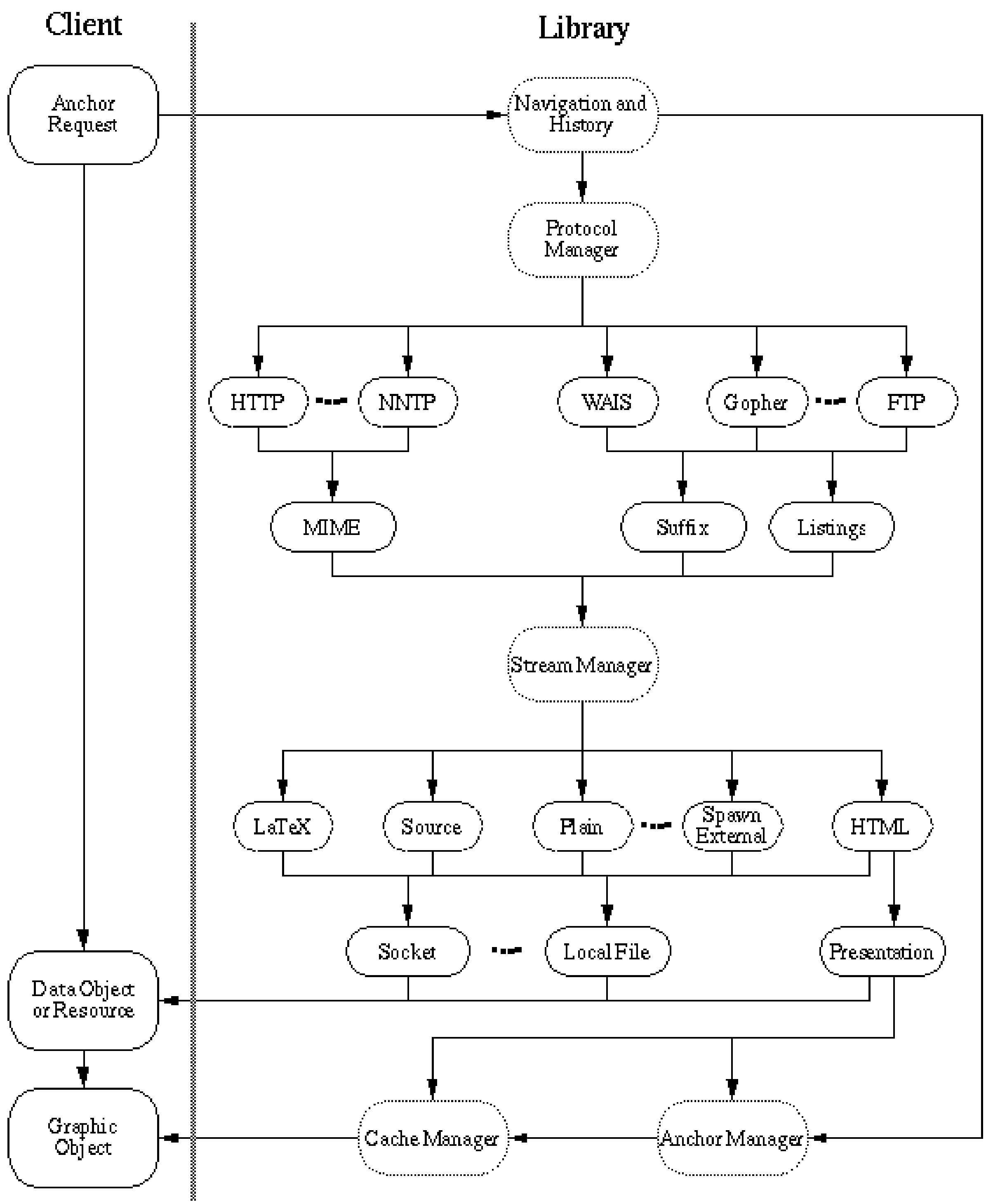
A graphic object is a displayable entity handled and maintained by the client. It is built from the data contained in a server response upon a successful request initiated by the client. The object can either be build directly from the data, e.g, if the data object returned is a HTML document, or it can be generated from a format converter within the library. The latter could be the generation of a HTML object from a FTP directory listing (7-bit ASCII).

Graphic objects are in general necessarily coded differently on different window systems. The graphic object is responsible for displaying itself, catching mouse clicks, and calling the navigation object in order to follow links. Often the more common term "document" is used to describe the logical entity which a graphics object represents and displays.

For the moment, a graphic object is created and maintained in the client side of the library and the client itself. However, it would be possible to extend the definition of a graphic object to also describe a data object being transferred from the server to the client using the HTTP protocol. The client can then use meta information given in the graphic object to display the raw data in the representation desired or available in the client.

5.1.2 Anchor Manager

Anchors represent parts of graphic objects which may be the sources or destinations of links. There are basically two types of anchors: *Parent anchors* and *child anchors*.


Client
Library
Anchor Request
Navigation and History
Protocol Manager
HTTP
NNTP
WAIS
Gopher
FTP
MIME
Suffix
Listings
Stream Manager
LaTeX
Source
Plain
Spawn External
HTML
Socket
Local File
Presentation
Data Object or Resource
Graphic Object
Cache Manager
Anchor Manager

Parent Anchors

These represent whole graphic objects (documents). Every graphic object has an associated parent anchor. Associated with a parent anchor is data including:

- The title of the associated document, if known. This allows the document's title to be displayed in lists of previous nodes visited , etc., even when the document itself has been freed.
- A flag as to whether the document is an index
- The address of the document. When a new anchor is created, the code ensures that if an anchor with that address already exists, then this anchor is returned instead, so no duplicates can exist. A problem concerning parent anchors is how to know when two URLs are equivalent, i.e., they point to the same resource on the Web. Host aliases, soft links and other constraints makes a directly comparison impossible even if the URL's are canonicalized.
- A list of children

Child Anchors

These represent parts of documents. The graphic object stores the correlation between the identity of the anchor and the actual space shape which is referred to. Child anchors contain

- A pointer to the parent
- Address of this anchor relative to the parent anchor

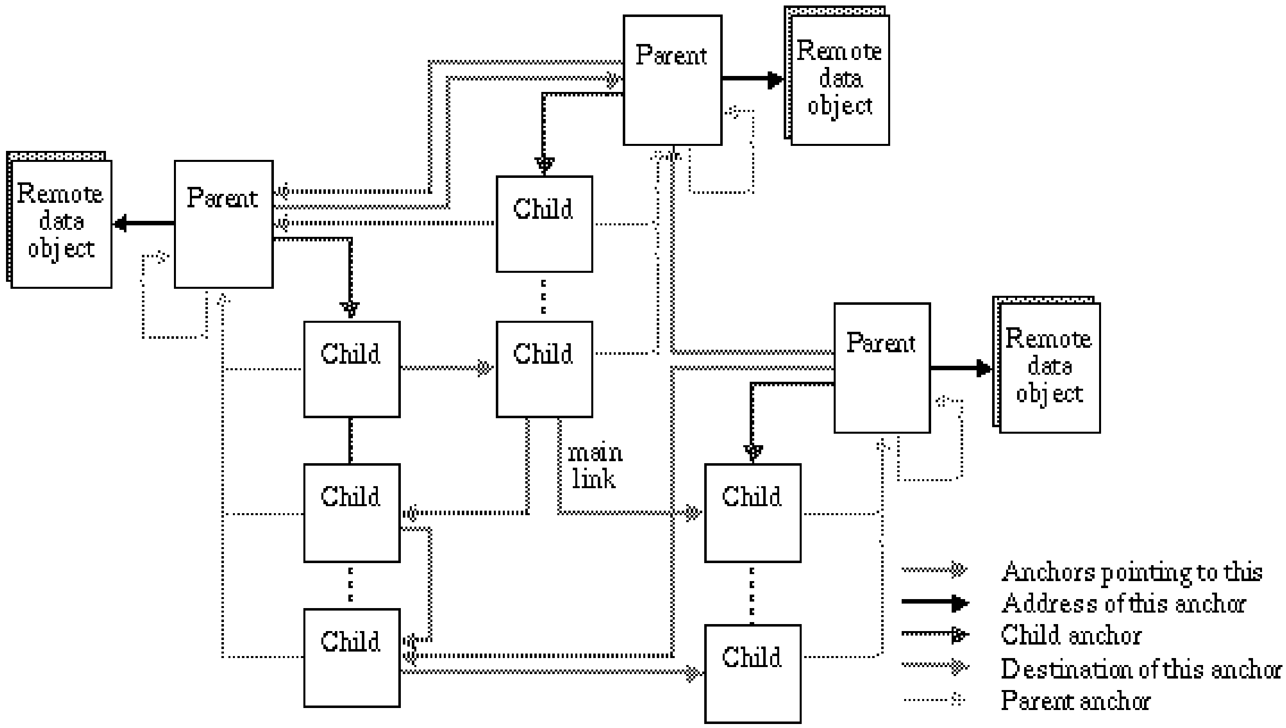
The relationship between parent anchors and child anchors can be illustrated as follows:

An anchor can be the source of zero, one, or many links . It has one "main" link for the (common) case in which it is the source for one link. When posting a data object to, e.g, a NNTP News Group or using the POST method in the HTTP Protocol it is common to have more than one recipient for the data object to be posted. The list of recipients are all in the "link list " of the anchor, this is explained in a later section on Put and Post

An anchor may be the destination of zero, one, or many links. The anchor module stores all links known by the program, and so in fact manages a copy of a small part of the Web.

5.1.3 Cache Manager

This is a local cache module specifically for WWW Clients. It is used to save data objects once they have been down loaded from the Internet. The CERN Proxy server has its own cache manager to handle a large scale cache that can serve hundreds of clients with documents once they have been received form the remote host. The client cache is made for clients not using a proxy cache or having a very slow link but a large local temporary storage.



5.1.4 Navigation and History

This module keeps track of the part of the Web that the user has visited during the World-Wide Web session. When a request is passed from the client to the library, this module searches the list of previous requested resources managed by the Anchor Manager. If it has already been accessed it first checks if the graphic object is still in memory on the client machine or if not it asks the cache manager if the object is stored in a temporary storage (local file system) on client side. The difference between a data object in the cache and in memory is that the memory version is a graphic object whereas the cache version is a resource that has to be loaded into memory and passed in order to be transformed into a graphic object.

5.1.5 Protocol Manager

The Protocol Manager is invoked by the client in order to access a document. Each protocol module is responsible for extracting information from a local file or remote server using a particular protocol. Depending on the protocol, the protocol module either builds a graphic object (e.g. hypertext) itself, or it passes a socket descriptor to the format manager for parsing by one of the parser modules. As mentioned in the Graphic Object section it can also perform a conversion of the raw data returned from the remote server into, e.g. a HTML object.

5.1.6 Stream Manager

Streams are unidirectional objects which accept characters, strings, and blocks of data to be written to them. The Stream Manager handles a generic representation of a stream class so that the interface is always the same for all types of different input and output streams to the manager.

Streams can be thought of as like files open for write. The stream-based architecture allows the software to be event-driven in the sense that when input arrives, it is put into a stream, and any necessary actions then cascade off that.

Stream might be cascaded so that one stream writes into into another stream after having performed some processing on the data. An output stream is often referred to as the "target" or "sink" stream.

Structured streams

A structured stream is a subclass of a stream, but instead of just accepting data, it also accepts SGML events such as begin and end elements. A structured stream therefore represents a structured document. A structured stream can be thought of as the output from an SGML parser. It is more efficient for modules which generate hypertext objects to output a structured stream than to output SGML which is then parsed.

The elements and entities in the stream are referred to by numbers, rather than strings. The DTD contains the mapping between element names and numbers , so each structured stream when created is associated with the DTD which it using. Any instance of a structured stream has a related DTD which gives the rules and element and entity names for events on the structured stream. The only DTD which is currently in the library is an extended version of a HTML DTD version 1.0.

The SGML parser uses a DTD to output to a structured stream from a stream of SGML. A hypertext editor will output to a structured stream when writing out a document. Many protocol modules output to a structures stream when generating their data structures.

Format Conversion and Stream Stacks

Often it is desired to perform a format conversion between the entry point and the output point of the stream. As illustrated in the Figure of the library the stream manager is the node between the input format given by the protocol modules and the desired output format specified by the client. Though, often it is desirable to perform more than one data conversion on a data object. Therefore, the stream manager is designed as a stream stack where several streams can be cascaded, each one performing a part of the total data conversion.

5.2 The Line Mode Browser

The CERN Line Mode Browser is a character based World-Wide Web Browser. It is developed for use on dumb terminals and as a test tool for the CERN Common Code Library. It can be run in interactive mode, non-interactive mode , as a proxy client and a set of other run modes that all are explained in Command Line Options. Even though it is not often used as a World-Wide Web browser, the possibility of executing it in the background or from a batchjob makes it a useful tool. Furthermore it gives a variety of possibilities for data format conversion, filtering etc.

The easist way to get an idea of what the Line Mode Browser is all about is actually to try directly from The info server at CERN. No userid or password is needed.

5.3 The HTTP Server

CERN httpd is a generic hypertext server which can be used as a regular HTTP server. The allocated port for HTTP connections is TCP port 80 , but the server can be put up to listen on any other TCP port (above 1024 if not running as root). The CERN server includes features such as Access Authentication, Clickable Images etc.

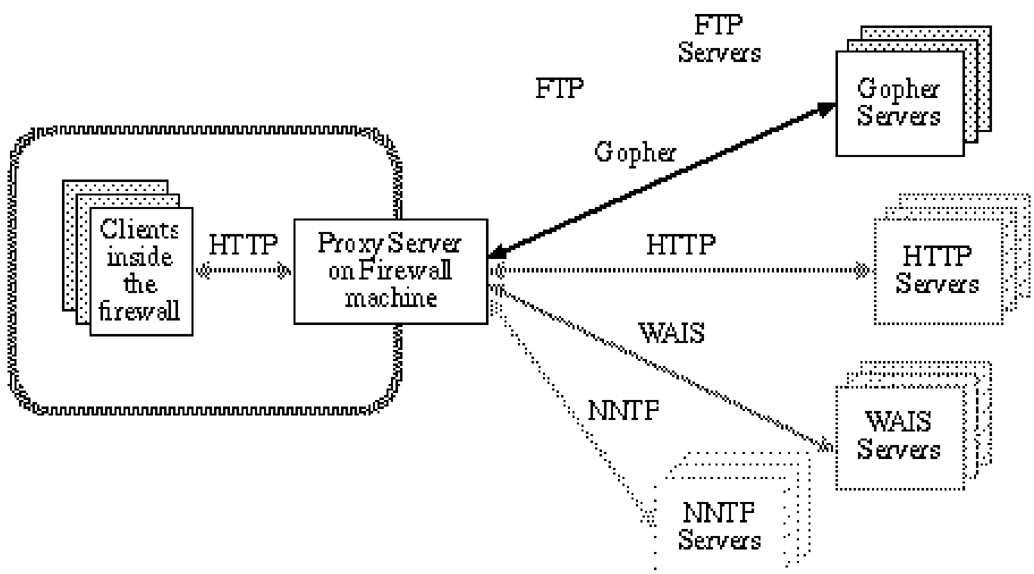
5.4 The Proxy Server

The CERN server also has the possibility of running as a proxy server. A proxy is a special HTTP server that typically runs on a firewall machine. The proxy waits for a request from inside the firewall, forwards the request to the remote server outside the firewall, reads the response and then sends it back to the client. Kevin Altis, Ari Luotonen and Lou Montulli have been the principle designers behind the current proxy standard as is illustrated in the following figure:

As seen from the figure, all communication between the client inside the firewall and the Proxy server is done using HTTP. This makes the client application much more effective as it can concentrate on the user interface and not on the Internet interface including presentation protocol clients etc.

In the usual case, the same proxy is used by all the clients within a given subnet. This gives another advantage of using a proxy server as it is possible for the proxy to do efficient caching of documents that are requested by a number of clients. The ability to cache documents also makes proxies attractive to groups of clients not inside a firewall as it cuts down the network traffic costs to remote hosts.

The CERN server had gateway features for a long time provided by Tim Berners-Lee, but this has recently been extended to support all the methods in the HTTP protocol used by WWW clients. Clients don't lose any functionality by going through a proxy, except special processing they may have done for non-native Web protocols such as Gopher and FTP.


FTP
Servers
FTP
Gopher
Servers
Gopher
Clients
inside
the
firewall
HTTP
Proxy Server
on Firewall
machine
HTTP
HTTP
Servers
WAIS
NNTP
WAIS
Servers
NNTP
Servers

Chapter 6

Implementation of the HTTP Client

This document describes the current implementation of the HTTP Protocol as from version 3.0 (unreleased, August 1994) of the Library of Common Code (see current version number). The HTTP client is based on the HTTP 1.0 specification but is backwards compatible with the 0.9 version. The major difference between the implementation before version 3.0 is that this version is a state machine based on the state diagram illustrated below. The advantage of this approach will become obvious in the section on Multi Threaded Clients even though the HTTP protocol is stateless by nature.

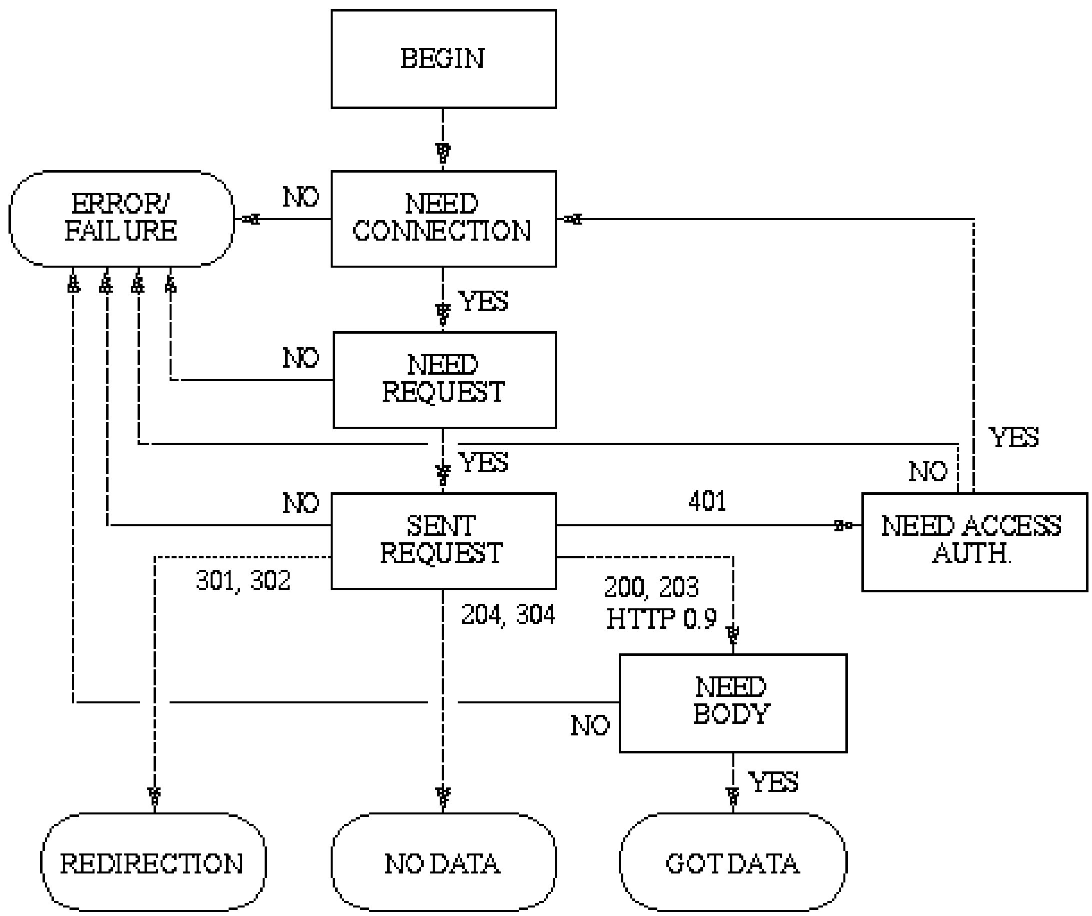
The individual states and the transitions between them are explained in the following sections. Note the difference in notation between the *client* (client side application built un top of the library) and *HTTP Client* that is the protocol module in the library.

BEGIN State This state is the idle state or initial state where the HTTP client awaits a new request passed from the client. If the user at this point has typed in a userid and a passwd for access authorization the HTTP client also prepares the Authorization header.

NEED_CONNECTION State The HTTP client is now ready for setting up a connection to the remote host. The connection is always initiated by a *connect* system call. In order to minimize the access to the Domain Name Server, all host names to previous visited hosts no matter of the access schemes used) are stored in a local host cache. The cache handles multi-homed hosts in a special way in that it measures the time it takes to actually make a connection to one of the IP-addresses. This time is stored together with the specific IP-address and the hostname in the cache and on the next connection to the same host the IP-address with the fastest connect time is chosen.

NEED_REQUEST State The HTTP Request is what the client sends to the remote HTTP server just after the establishment of the connection. The request consists of a HTTP header line, a set of MIME Headers, and possibly a data object to be posted to the server. The header line has the following format:

```
<METHOD> <URI> <HTTP-VERSION> CRLF
```


BEGIN
NEED CONNECTION
ERROR/ FAILURE
NO
YES
NEED REQUEST
NO
YES
SENT REQUEST
NO
401
NEED ACCESS AUTH.
YES
NO
301, 302
204, 304
200, 203
HTTP 0.9
NEED BODY
NO
YES
REDIRECTION
NO DATA
GOT DATA

Current methods supported in the clients are

GET This is for requesting a URI or for specifying a text search. Text searches are initiated by placing a "?" in the URI.

HEAD The HEAD method is equivalent to GET in the sense that it is requesting a URI at the remote server. The difference is that the server only returns the HTTP headers but no data object. This is used for updating cache information, getting information on the size of the data object (or body) etc.

In the section on Put and Post the implementation of the client interface to the library is described. However, the actual implementation of PUT and POST in the HTTP protocol is yet to be specified. The reason is that the current specification is limited and does not allow the HTTP protocol to be a superset of existent Internet Presentation Protocols.

The HTTP headers are as mentioned a set of MIME headers even though they are not all officially accepted by the MIME specifications. The HTTP client supports the following headers:

Accept: The current implementation uses one accept line for each MIME-type supported by the client. For advanced clients this means that the "Accept: " sequence is repeated 20-30 times which gives a overhead of 200-300 bytes per request (including the CRLF telnet EOL-sequence). This should be changed so that either a comma separated list is transmitted instead or only the MIME content types without any subtypes.

Referer: If any parent anchor is known to the requested URI this is send in the referer field. This is to let the server know what link has led to the current request. Nothing is sent if the parent anchor is unknown or does not exist.

From: The full email address is sent along the request. It is meant as an informative service to the recipient and can be changed to any value the user wishes to sign the request with. As it is possible to manipulate the email address, this field can not be used for any security verification or precaution.

User-Agent: The user agent is by many clients currently generated in a somewhat verbose format. The goal is to make this field machine readable so it can be used on the server side to perform individual actions as a function of the client version. As a side effect it can also be used for statistics etc.

Authorization: The authorization header as introduced in the NEED_CONNECTION state.

SENT_REQUEST State When the request is sent the client waits until a response is given from the server or the connection is timed out in case or an error situation. As the client does not know whereas the remote server is a HTTP 0.9 server or a HTTP 1.0 it must look at the first part of the response to figure out what version of HTTP is returned. The reason is that the HTTP protocol 0.9 does not contain a HTTP header line in the response. It simply starts to send the requested data object as soon as the GET request is handled. Future versions of the HTTP protocol will all contain a header line with the protocol version like the MIME protocol.

NEED_ACCESS_AUTHORIZATION State If a 401 Unauthorized status code is returned the client asks the user for a user id and a password, see also the Access Authorization Scheme. The connection is closed before the user is asked for the userid and password so any new request initiated upon a 401 status code causes a new connection to be established.

REDIRECTION State The remote server returns a redirection status code if the URI has either been moved temporaryly or permanent to another location, possibly on another HTTP server or any other server supported by the WWW-model. The HTTP Client supports both a temporaryly and a permanent redirection code returned from the server:

301 Moved The load procedure is recursively called on a 301 redirection code. The new URI is parsed back to the user as information via the Error and Information module, and a new request generated. The new request can be of any access scheme accepted in a URI. An upper limit of redirections has been defined (default to 10) in order to avoid infinite loops.

302Found The functionality is the same as for a 301 Moved return status. A clever client can use the returned URI to change the document in which the URI originates so that the URI points to the new location.

NO_DATA State When a return code indicates that no data object or resource follows the HTTP headers the HTTP client can terminate the request and pass control back to the client.

NEED_BODY State If a body is included in the response from the server, the client must prepare to read the data from the network and direct it to the destination set up by the client. This is done by setting up a stream stack with the required conversions.

GOT_DATA State When the data object has been parsed through the stream stack, the HTTP client terminates the request and handles control back to the client.

ERROR or FAILURE State If at any point in the request handling a fatal error occurs the request is aborted and the connection closed. All information about the error is parsed back to the client via the Error and Information Module. As the HTTP protocol is stateless, all errors are fatal between the server and the client. If the erroneous request is to be repeated, the request starts in the initial state.

Chapter 7

Multi Threaded Clients

This section describes the design and implementation of the multi-threaded, interruptable I/O HTTP Client as it is implemented in the World-Wide Web Library of Common Code. The section is an extension to the description of the HTTP Client and is divided into the following sections:

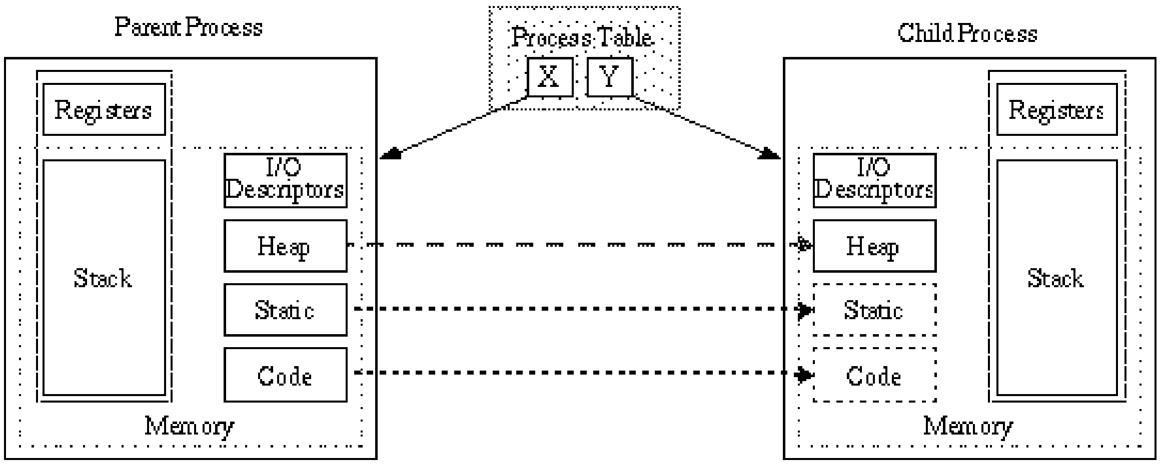
1. Introduction
2. Platform Independent Implementation
3. Modes of Operation
4. Control Flow

7.1 Introduction

In a single-process, single-threaded environment all requests to, e.g., the I/O interface blocks any further progress in the process. Any combination of a multi-process or multi-threaded implementation of the library makes provision for the user of the client application to request several independent documents at the same time without getting blocked by slow I/O operations. As a World-Wide Web client is expected to use much of the execution time doing I/O operation such as "connect" and "read", a high degree of optimization can be obtained if multiple threads can run at the same time.

A multi-process environment requires an extensive support from the underlying operating system. Unix is the classic platform that provides this functionality. The Unix system call "fork" generates a new child process that is an exact copy of the parent but in another address space as illustrated in the figure.

The static parts like the code segment and the static memory can be shared between the two processes and does not require a copy of the memory. Often a technique called *copy-on-write* is used when creating the heap and stack of the child. The kernel marks the regions in memory as read-only and only when either of the processes accesses the memory with a write request, the particular page is copied. Finally all open file descriptors and



socket descriptors are copied so that the child can continue any operation including I/O from the same state as the parent.

The process of forking a child process is not unique for Unix, but the exact behavior is often quite platform dependent. Under VMS, "fork" is an extremely resource expensive procedure that in practice is unusable for fast program execution. Due to extensive security regulations in VMS, every process has a large set of environment variables that has to be initialized at the creation of the process. Furthermore, a process is created in an initial state independent of the parent process, so synchronization of the state of the parent and child process has to be established before the child is ready to execute the request.

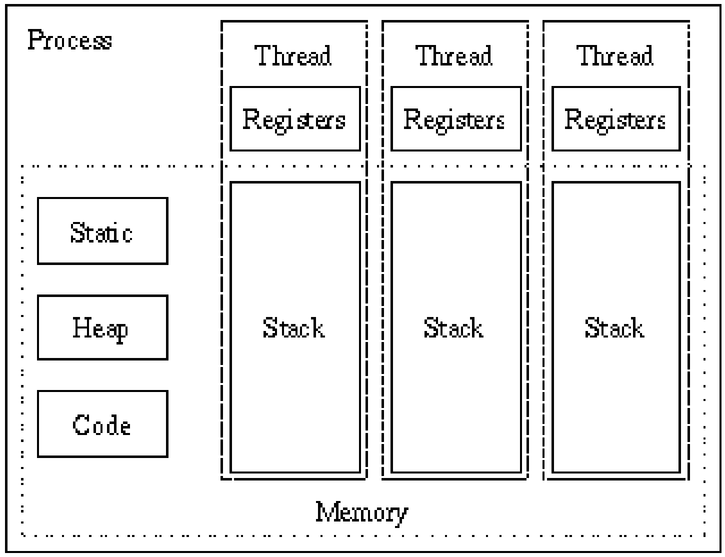
Threads provide another technique for obtaining an environment with a multiple set of execution points. A thread is a smaller unit compared to a process in that it is a single, sequential flow of control within a process. As mentioned above, when creating a new process much of the environment does never change and can therefore be reused. Threads takes the full consequence of this and creates an environment with multiple execution points within the same process. Hence threads provide a more lightweight solution than process forking and this is a part of the reason for their implementation in the Library of Common Code.

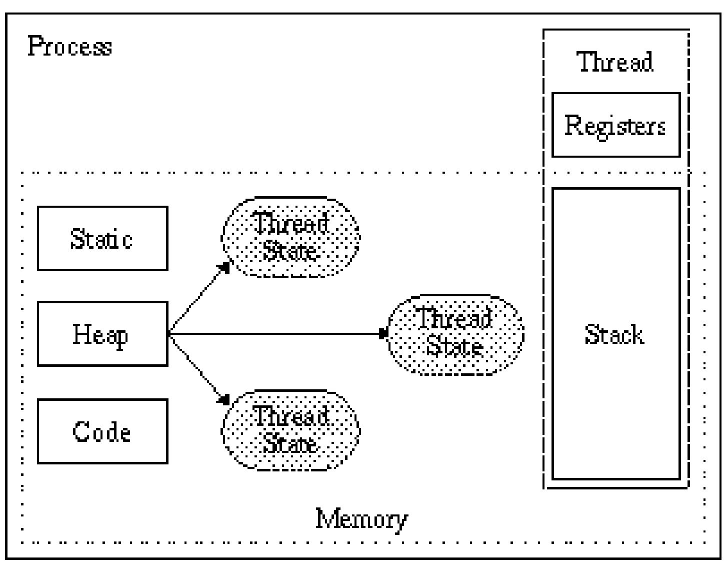
7.2 Platform Independent Implementation

The major concern in the design has been to make an implementation that is as platform independent as possible. This means that it has not been possible to use traditional thread packages like DECthreads which contain a code library with a complete set of thread handling routines and a consistent user interface. IEEE has publicized the POSIX standard 1003.4 for multi-threaded programming but even this will eventually limit the portability of the code so that it will not be usable on small platforms like PCs.

Instead the multi-threaded functionality of the HTTP client has been designed to be used in a single-processor, single-threaded, environment as illustrated in the figure.

The difference between this technique and "traditional" threads as illustrated above is


Process
Thread
Thread
Thread
Registers
Registers
Registers
Static
Heap
Code
Stack
Stack
Stack
Memory


Process
Thread
Registers
Static
Heap
Code
Thread State
Thread State
Thread State
Stack
Memory

that all information about a thread is stored in a data object which lives throughout the lifetime of the thread. This implies that the following rules must be kept regarding memory management:

- Global variables can be used only if they at all time are independent of the current state of the active thread.
- Automatic variables can be used only if they are initialized on every entry to the function and stay state independent of the current thread throughout their lifetime.
- All information necessary for completing a thread must be kept in an autonomous data object that is passed round the control flow via the stack.

These rules makes it possible to animate a multi-threaded environment using only one stack without any portability problems as the implementation is done in plain C on top of the Internet TCP API.

7.3 Modes of Operation

In order to keep the functionality of the HTTP Client as general as possible, three different modes of operation are implemented:

Base Mode This mode is strictly single-threaded and is what the library is today, that is version 2.16pre2 (and 2.17 (unreleased, August 94). The difference between this mode and the other two is basically that all sockets are made blocking instead of non-blocking. The HTTP client itself is the same as for the other modes and is basically a state machine as described in the section on the Implementation of the HTTP Client. The mode is preserve compatibility with clients that use a single-threaded approach. This is also the mode used for the CERN Proxy server using forking. Currently this mode does not provide interruptable I/O as this is a integral part of the event loop.

Active Mode In this mode the event loop (select-function) is placed in the library. This mode is for dumb terminal clients that can interrupt the execution only through standard input using the keyboard. The client can , however, still be multi-threaded in the sense that it can activate pre-fetch of documents not yet requested by the user. If a key is hit, the library has a call back function to the client so that the client decides whether the current operation should be interrupted or not. If so, the library stops all I/O activity and handles the execution back to the client. The active mode should only cause minor changes to the client in order to obtain a simple form of multiple threads and interruptable I/O.

Passive mode This is the mode that requires the most advanced client , e.g., a GUI client. On every HTTP request from the client, the library initiates the connection and as soon as it is ready for reading or writing, it returns an updated list of active socket descriptors used in the library to the client. When the client sees that a socket is ready for action or it has been interrupted, it calls a library socket-handler passing the socket number and what has to be done. Then the socket handler finds the corresponding request and executes the read, write or interrupt. As soon as the thread has to access the network again, the socket handler stops and returns the execution to the client.

7.4 Data Structures

The basic data structure for all requests to the library regardless of the access scheme used is the HTRequest structure. This structure was introduced in the 2.15 release of the library, but was a completely flat data model in the first version. In version 2.16 and later, the request structure has turned into a hierarchical data model in order to establish more clean interfaces between the data structures in the library.

As no automatic or global variables are available in this implementation model every thread has to be state dependent and must contain all necessary information in a separate data object. In order to make a homogeneous interface to the HTRequest structure the new protocol specific data structure HTNetInfo has been defined.

The definition of this data object is highly object oriented as every protocol module in practice can define a sub class of the HTNetInfo structure in order to add information necessary for completing a thread. Again this is all done in plain C in order to maintain a high degree of portability.

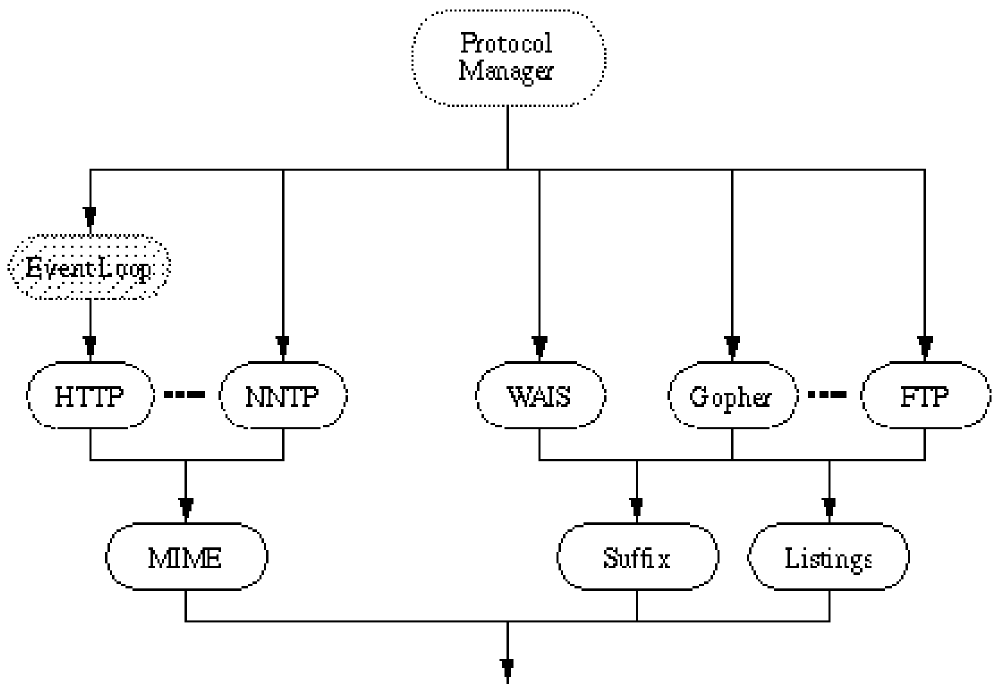
7.5 Control Flow

A consequence of having multiple threads in the library is that the control flow changes to be an event driven flow where any action is initiated by an event either caused by the user or the network interface. However, as the current implementation of multiple threads is valid for HTTP access only, the data flow of the library has basically been preserved, see the general control flow diagram.

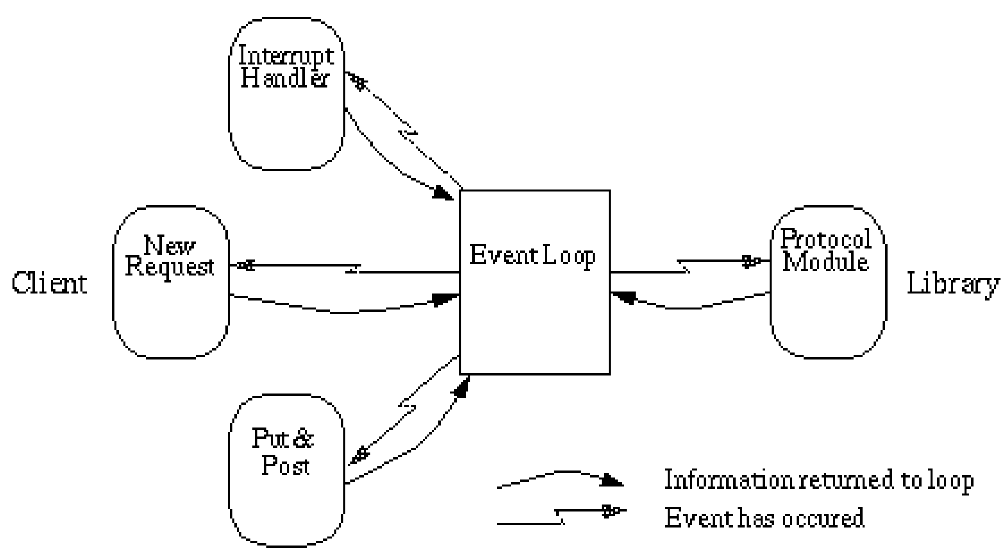
All other access schemes but HTTP protocol still use blocking I/O and the user will not notice any difference from the current implementation. The result of this is that full multi-threaded functionality is enabled only if the client uses consecutive HTTP requests even though the FTP and Gopher clients now also are implemented as state machines and in principle can use the same approach.

When a request is initiated having another access scheme than HTTP, e.g. FTP, the multi-threaded functionality partly stops as the new request gets served using blocking I/O. It is currently for the client to decide whether a new non-HTTP request can be activated when one or more HTTP request are already active. It is strongly recommended for the active mode that the client awaits the return from the HTTP event-loop, i.e., that no more HTTP requests are active or pending.

For the HTTP access, however, a socket event loop has been introduced. This might as indicated in the Introduction either be implemented by the client or the library. When other protocol modules than the HTTP client are fully implemented as multi-threaded clients they can be moved down under the event loop just like the HTTP client.



The event loop is designed using event driven call back functions. When used in active mode, the only events recognized are from a given set of file descriptors including standard input (often specified as file descriptor 0). As indicated in the figure, the event loop handles two kinds of call back functions: the ones that are internal library functions such as the loading function in the protocol modules, and the ones that require an action taken by the client application.



7.6 Interrupting a HTTP Request

The current interrupt handler meant for active mode is quite lazy as it only looks for interrupts when about to execute a blocking I/O operation and program execution returns to the event loop. The reason for this is that the user is not blocked even though the interrupt does not get caught right away so it is not as critical as in a single-threaded environment. If using the passive mode then the client has complete control over when to catch interrupts from the user and also how and when to handle them.

Chapter 8

Client Interface for Posting

This document describes the posting interface between a World-Wide Web client and the Library of Common Code. The design has been made so that the interface is flexible enough to support not only the POST method in the HTTP Protocol, but at the same time provide functionality for posting to email addresses using SMTP and NNTP news groups. Posting to other Internet servers such as FTP servers is not considered in this document, but the concept is designed with the possibility of including additional Internet protocols.

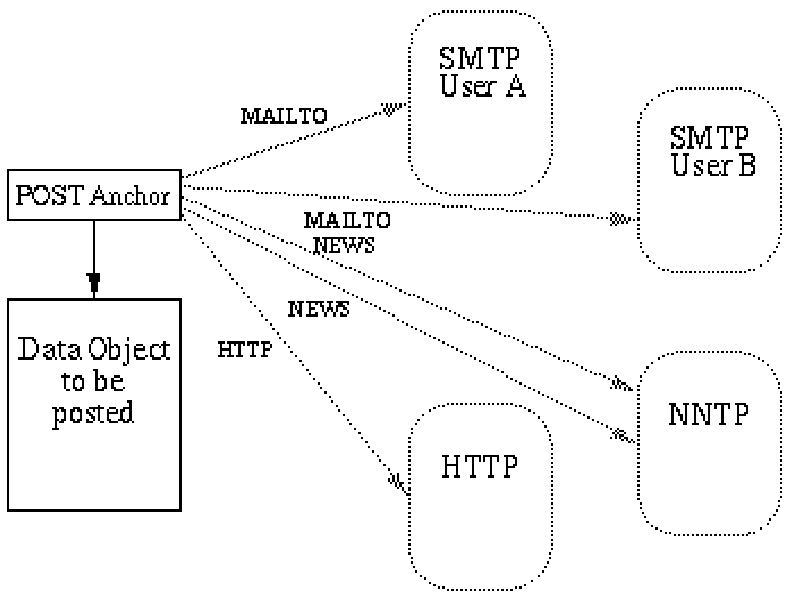
The document only describes the Client interface to the library but not how the actual posting should take place using the HTTP protocol. A draft specification is under development but has not been completed at this point (August 1994).

8.1 Building a POST Web

The basic idea of the post interface to the library is that when using POST , the user is often interested in posting the data object to more than one recipient, e.g. the user can send the same data object to two mailing lists, a news group and at the same time store the data object as a file on a remote HTTP server. The term "data object" is used instead of "document" because the object to be posted can be in any MIME format and furthermore, it can be compressed or encoded before transmission.

As indicated in the figure, the user builds a "POST Web" around an anchor element before the POST is committed and passed to the library. The rounded boxes are remote logical data objects that are still to be created and the lines are logical links between the post anchor and the remote data objects. The two links to the NNTP box represents that the data object is to be posted to two different news groups. In practice only one physical link would actually exist to the NNTP as all posting to a NNTP Server is done in one single transaction.

The actual user interface is for the client to implement, but typically a GUI-client could use drag-and-drop icons for building the Web. The POST Web could be visualized using a user implemented menu of icons of the most used recipients and then let the user drag lines between the data object to be posted and the recipients. The CERN library helps



the user building the Web by providing the functionality of linking the anchors together.

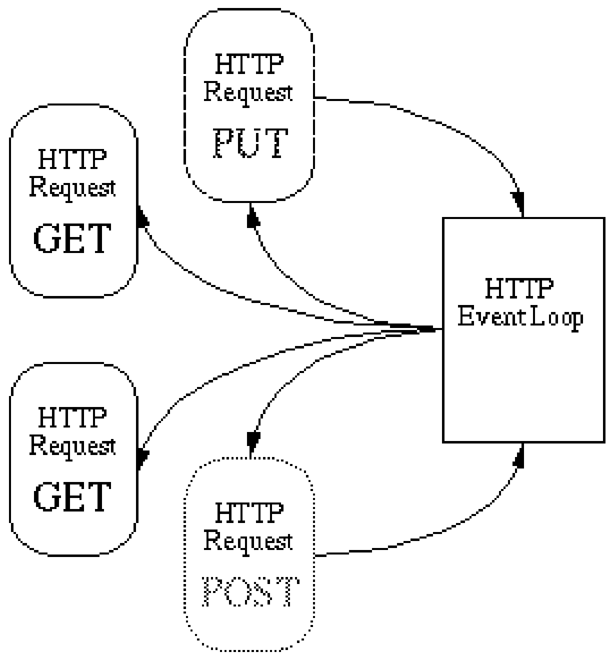
8.2 Posting a Document

When the user has decided to commit the POST, the post anchor is passed from the client to the library together with a request structure. This is very similar to the current implementation of a HTTP GET Method. An important consideration in the design has been to include the multi-threaded functionality in the posting module so that both PUT and POST in principle behaves exactly like a GET or any other method using non-blocking I/O, see also the multi-threaded control flow.

However, PUT and POST methods are inherently different from the GET method as they require a data flow in both directions. This is especially the case using PUT as a data object often will be returned from the remote server.

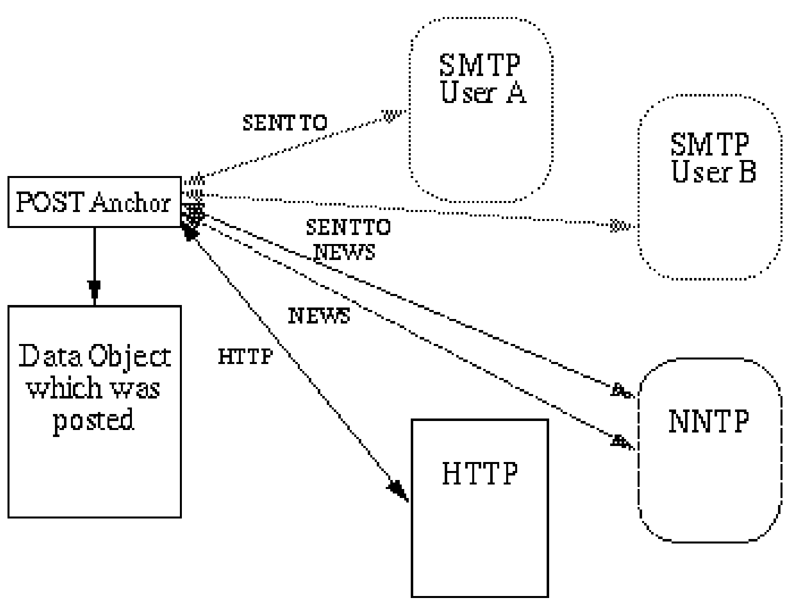
The routine for handling the POST in the library parses the POST Web and groups the individual requests into protocol categories. In the case of a NNTP request, all POST requests are handled at the same time and send to the actual NNTP server. When the parsing of the Web is done, the post module recursively calls the protocol modules to execute the POST.

The posting mechanism is designed to be compatible with the multi-threaded structure of the library. This means that the client is asked for data based on an event driven action taken by the event loop. The client then gets a stream so that the document to be posted can be pumped down the stream and out on the net.



8.3 Handling the POST Result

The result of the posting varies as a function of the protocol used. It is a general rule through out the design of the Library of Common Code that other protocols than HTTP should be supported but not extended beyond their individual limitations. This means that the library has to be flexible enough to handle more than one result from a posting transaction dependent on the protocol used.



An immediate result from a post transaction is available using NNTP or HTTP but when using SMTP, the result might be delayed several days. In practice there is no way that the client can await a response for that amount of time. Therefore it is proposed to introduce "sentto" as a new URI scheme specifically to indicate the state of the posting to a mail address. This code means that the data object has been posted but no guarantee is given whether the recipient has actually received the data object or if the data object has been modified under the posting transaction. However, note that even on a successful return code from the server, the user is not guaranteed that the posting action has actually been executed or that the posted data object will not be changed or deleted. The returned URLs from the posting will therefore have the following syntax:

http://remote.server/local/location/foo.bar The syntax of the generated HTTP URL as a result of a succeeded posting is identical to any other HTTP URL

news:messageid@news.host The message ID is a unique string generated by the library NNTP Module.

sentto:recipient@host.name As the success of a mail posting can not be guaranteed, the access scheme has changed from "mailto" to "sentto". Then it is for the user or client to decide what to do with the URL. One solution could be to keep a copy of the posted document in a local cache for a limited time period. Though it is not foreseen that the World-Wide Web client can actually handle delayed return codes from the SMTP protocol, as the client can in fact be a HTTP Proxy server.

In the figure, postings using SMTP are still rounded because it is not possible for the user to actually access the document using the URL returned from the library. As the NNTP posting failed in the figure, the document is still non-existent. Only the HTTP posting has turned into a physical document reference on the remote server.

When all postings are terminated, a new anchor is generated containing the allocated links on the remote servers for future references (except in the SMTP case). The amount of links returned might be a subset of the requested recipients as only postings which terminated successfully are registered. Using this set, the client must determine what to do with failed postings. One possibility would be to try again using the failed part of the POST Web, or simply to discard it.

8.4 Summary

This document describes the posting interface between the client application and the World-Wide Web Library of Common Code. The user first builds a "POST Web" where logical links are created between the document to be posted and the set of desired recipients. On commit of the posting the POST web and a request structure is parsed to the general posting routine within the library. The posting routine groups the postings and recursively calls the NNTP, SMTP and or the HTTP module of the library until the POST Web is traversed. The NNTP module is only done once due to the NNTP protocol implementation of posting.

The "sentto" access scheme has been proposed as a new URI access scheme to compensate for the fact that when posting to a SMTP server, no guarantee is given that the data object has actually reached the recipient.

As mentioned in the introduction this document doesn't describe how the HTTP protocol should handle POST as this is all handled internally in the Library. However, the interface between the Library and the client stays the same regardless of the HTTP POST Implementation.

Chapter 9

Summary

This report describes the World-Wide Web as a generic model for exchanging information on the Internet. The model supports several Presentation Layer protocols such as FTP, Gopher, etc. but the main protocol is the Hypertext Transfer Protocol. This protocol is a client-server based protocol based on the TCP protocol. The TCP protocol has an advantage over UDP in that it provides a reliable stream-oriented service to the HTTP protocol. On the other hand, TCP implies a substantial overhead in the 3 Way Handshake (3WHS) technique used for setting up a connection. A very interesting alternative to the TCP protocol is Transactional TCP (T/TCP) protocol that short circuits the 3WHS connection establishment and optimizes the reliable stream transport for client-server applications with many consecutive connections between the same hosts.

The following software modules and programs which are all developed and maintained by the CERN World-Wide Web team

World-WideWeb Library of Common Code A general purpose code basis for many World-Wide Web applications. It contains code for accessing and parsing information from the Internet and pass it to the client application. An updated flow diagram is presented.

LineMode Browser A test tool for the library and useful as a filter run from batch jobs

HTTPServer A generic HTTP server for handling client requests handling access authentication etc.

Proxy Server A special HTTP server that can be used on a firewall machine. The proxy waits for a request from inside the firewall, forwards the request to the remote server outside the firewall, reads the response and then sends it back to the client. Furthermore, the CERN Proxy server has a cache manageer included in the application. As the proxy is a central node for connections network performance can be improved significantly by this cache.

A new HTTP client has been designed and implemented as an alpha version (August 1994). It is based on a state machine concept in order to provide it with a multi-threaded, interruptable I/O interface to the client application. The multi-threaded design is based on a single-stack, single-process environment in order to make the library portable to inherently single-threaded platforms such as PCs. An event loop has been introduced into the control flow diagram of the library. The event loop, implemented inside the library, is based on a set of call back functions to the client application initiated by events from either the network interface or standard input.

A design for how World-Wide Web Clients interface to the library when posting a data object has been designed and implemented in an alpha version (August 1994). It gives the user the opportunity of posting data object to more than one recipient at the same time by creating a POST Web with links to the different destinations, for example a NNTP news group, a SMTP email address, and a HTTP server.

The posting interface is to be followed up by an enhancement of the HTTP protocol to handle multiple POST requests at the same time. A draft proposal is under development but is not finished at this moment. The plan is to combine this feature with a multiple GET feature so that the HTTP protocol is expanded to handle compound requests in both directions all handled within the same transaction. However, the basic stateless client-server model is preserved as the client still sends exactly one request and the server sends exactly one response. The current specifications only allow one single data object to be transferred per transaction.

Chapter 10

References

The following references are strictly references to information that I havn't been able to find on the Internet.

- *Advanced Programming in the UNIX Environment*, **W. Richard Stevens**, Addison-Wesley 1992, ISBN 0-201-56317-7
- *Computer Networks*, Second Edition, **Andrew S. Tanenbaum**, Prentice-Hall International Editions 1989, ISBN 0-13-166836-6
- *DNS and BIND*, **Paul Albitz & Cricet Liu**, O'Reilly & Associates, Inc. 1992, ISBN 1-56592-010-4
- *Guide to DECThreads*, Digital Equipment Corporation, November 1991, Order Number AA-PJNEA-TK
- *Internetworking with TCP/IP, Principles, Protocols, and Architecture*, **Douglas Comer**, Prentice-Hall 1988, ISBN 0-13-470154-2
- *UNIX Internetworking*, **Uday O. Pabrai**, Artech House 1993, ISBN 0-89006-686-X
- *Modern Operation Systems*, **Andrew S. Tanenbaum**, Prentice-Hall International Editions, 1992, ISBN 0-13-595752-4
- *UNIX Network Programming*, **W. Richard Stevens**, Prentice-Hall 1990, ISBN 0-13-949876-1

Henrik Frystyk, frystyk@info.cern.ch, July 1994