

UNIT-6

TRANSPORT LAYER & APPLICATION LAYER

APPLICATION LAYER:

DOMAIN NAME SYSTEM: The essence of DNS is the invention of a hierarchy domain based naming scheme and a distributed database system for implementing this naming scheme.

It is primarily used for mapping host names and email destinations to IP Addresses. But can also be used other purposes. DNS is defined in RFCs 1034 and 1035.

To map a name onto an IP Address, an application program calls a library procedure called “**Resolver**” passing it the name as parameter.

The resolver sends a UDP packet to a local DNS server, which then looks up the name and returns the IP address to the resolver, which then returns it to the caller.

Armed with the IP address, the program can then establish a TCP connection with the destination, or send it UDP packets.

Domain Name Space (DNS-Name Space): The Internet is divided into several hundred top level domains, where each domain covers many hosts. Each domain is partitioned into sub domains, and these are further partitioned as so on. All these domains can be represented by a tree, in which the leaves represent domains that have no sub domains. A leaf domain may contain a single host, or it may represent a company and contains thousands of hosts. Each domain is named by the path upward from it to the root. The components are separated by periods (pronounced “dot”)

Eg: Sun Microsystems Engg. Department = eng.sun.com.

The top domain comes in 2 flavours:-

- ✓ **Generic:** com (commercial), edu (educational institutions), mil(the U.S armed forces, government), int (certain international organizations), net(network providers), org (non profit organizations).
- ✓ **Country:** include 1 entry for every country. Domain names can be either absolute (ends with a period e.g. eng.sum.com) or relative (doesn't end with a period). Domain names are case sensitive and the component names can be up to 63 characters long and full path names must not exceed 255 characters.

Insertions of a domain into the tree can be done in 2 ways:-

- Under a generic domain (Eg: cs.yale.edu)
- Under the domain of their country (E.g.: cs.yale.ct.us)

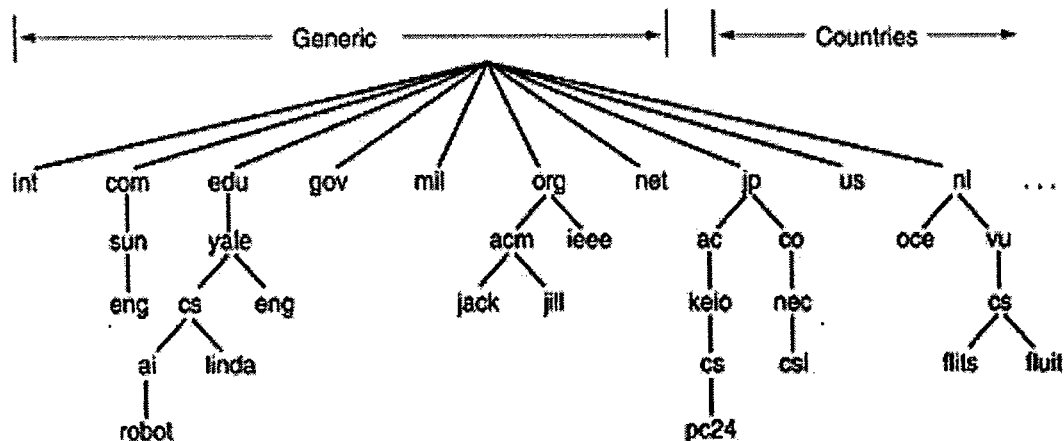


Figure 5-1. A portion of the Internet domain name space.

Resource records: Every domain can have a set of resource records associated with it. For a single host, the most common resource record is just its IP address. When a resolver gives a domain name to DNS, it gets both the resource records associated with that name i.e., the real function of DNS is to map domain names into resource records. A resource record is a 5-tuple and its format is as follows:

Domain name	Time to live	Class	Type	Value
-------------	--------------	-------	------	-------

Domain_name: Tells the domain to which this record applies.

Time-to-live: Gives an identification of how stable the record is (High Stable = 86400 i.e. no. of seconds /day) (High Volatile = 1 min)

Type: Tells what kind of record this is.

Class: It is IN for the internet information and codes for non internet information.

Value: This field can be a number a domain name or an ASCII string.

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

Name servers:

It contains the entire database and responds to all queries about it. DNS name space is divided up into non-overlapping zones, in which each zone contains some part of the tree and also contains name servers holding the authoritative information about that zone.

When a resolver has a query about a domain name, it passes the query to one of the local name servers:

1. If the domain being sought falls under the jurisdiction of name server, it returns the authoritative resource records (that comes from the authority that manages the record, and is always correct).
2. If the domain is remote and no information about the requested domain is available locally the name server sends a query message to the top level name server for the domain requested.

E.g.: A resolver of flits.cs.vle.nl wants to know the IP address of the host Linda.cs.yale.edu

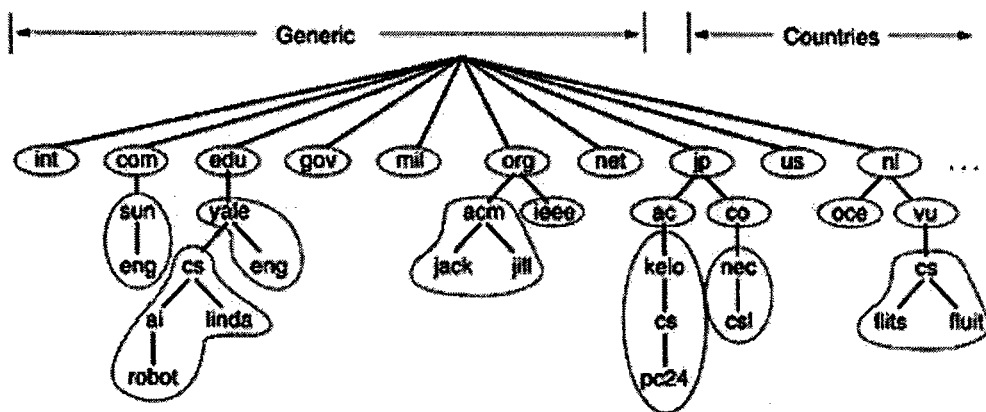


Figure 5-2. Part of the DNS name space showing the division into zones.

Step 1: Resolver sends a query containing domain name sought the type and the class to local name server, cs.vu.nl.

Step 2: Suppose local name server knows nothing about it, it asks few others nearby name servers. If none of them know, it sends a UDP packet to the server for edu-server.net.

Step 3: This server knows nothing about Linda.cs.yale.edu or cs.yale.edu and so it forwards the request to the name server for yale.edu.

Step 4: This one forwards the request to cs.yale.edu which must have authoritative resource records.

Step 5 to 8: The resource record requested works its way back in steps 5-8 This query method is known as **Recursive Query**

3. When a query cannot be satisfied locally, the query fails but the name of the next server along the line to try is returned.

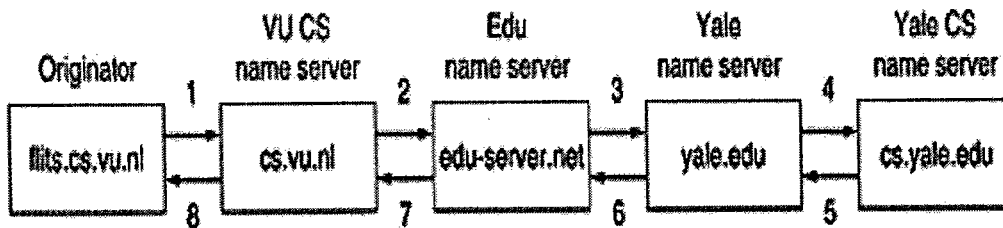


Figure 5-3. How a resolver looks up a remote name in eight steps.

Electronic Mail: Electronic mail, or email, is a very popular application in computer networks such as the Internet. Email appeared in the early 1970s and allows users to exchange text based messages. Initially, it was mainly used to exchange short messages, but over the years its usage has grown. It is now not only used to exchange small, but also long messages that can be composed of several parts as we will see later.

Advantages

E-mail has proved to be powerful and reliable medium of communication. Here are the benefits of **E-mail**:

- ✓ Reliable
- ✓ Convenience
- ✓ Speed
- ✓ Inexpensive
- ✓ Printable
- ✓ Global
- ✓ Generality

Disadvantages

Apart from several benefits of E-mail, there also exists some disadvantages as discussed below:

- ✓ Forgery
- ✓ Overload
- ✓ Misdirection
- ✓ Junk
- ✓ No response.

Architecture and services:

E-mail systems consist of two subsystems. They are:-

- (1). **User Agents**, which allow people to read and send e-mail
- (2). **Message Transfer Agents**, which move messages from source to destination

E-mail systems support 5 basic functions:-

- a. Composition
- b. Transfer
- c. Reporting
- d. Displaying
- e. Disposition

(a). Composition: It refers to the process of creating messages and answers. Any text editor is used for body of the message. While the system itself can provide assistance with addressing and numerous header fields attached to each message.

(b). Reporting: It has to do with telling the originator what happened to the message that is, whether it was delivered, rejected (or) lost.

(c). Transfer: It refers to moving messages from originator to the recipient.

(d). Displaying: Incoming messages are to be displayed so that people can read their email.

(e). Disposition: It concerns what the recipient does with the message after receiving it. Possibilities include throwing it away before reading (or) after reading, saving it and so on.

Most systems allow users to create **mailboxes** to store incoming e-mail. Commands are needed to create and destroy mailboxes, inspect the contents of mailboxes, insert and delete messages from mailboxes, and so on.

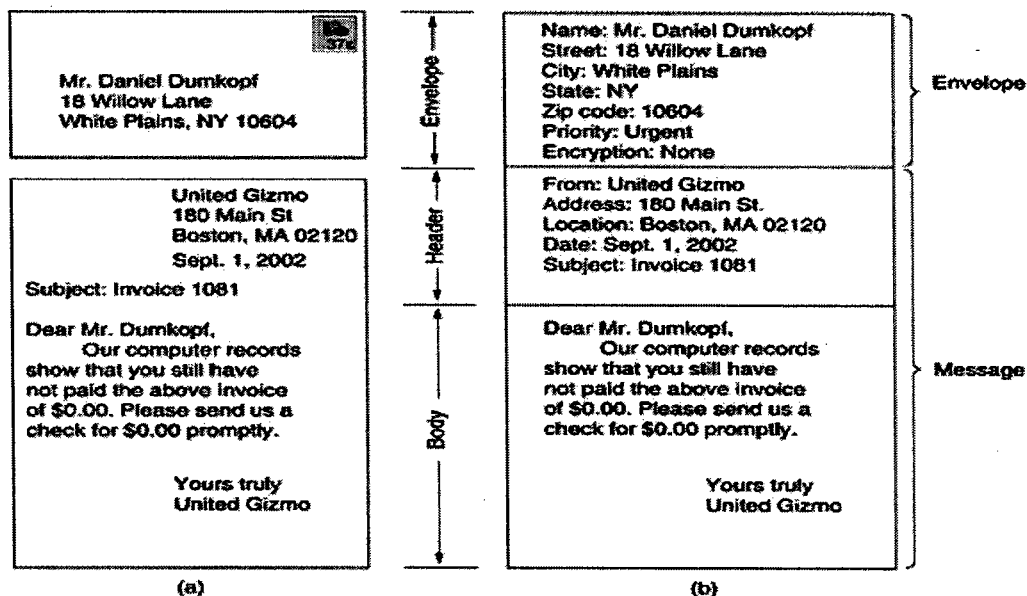


Figure 5-4: Envelopes and messages. (a) Paper mail. (b) Electronic mail.

User Agent : A user agent is normally a program (sometimes called a mail reader) that accepts a variety of commands for composing, receiving, and replying to messages, as well as for manipulating mailboxes.

Sending E-Mail : To send an e-mail message, a user must provide the message, the destination address, and possibly some other parameters. The message can be produced with a free-standing text editor, a word processing program, or possibly with a specialized text editor built into the user agent. The destination address must be in a format that the user agent can deal with. Many user agents expect addresses of the form **user@dns-address**.

However, it is worth noting that other forms of addressing exist. In particular, X.400 addresses look radically different from DNS addresses. They are composed of attribute=value pairs separated by slashes, for example :

/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/

This address specifies a country, state, locality, personal address and a common name (Ken Smith). Many other attributes are possible, so you can send e-mail to someone whose exact e-mail address you do not know, provided you know enough other attributes.

Reading E-Mail : When a user agent is started up, it looks at the user's mailbox for incoming e-mail before displaying anything on the screen. Then it may announce the number of messages in the mailbox or display a one-line summary of each one and wait for a command.

#	Flags	Bytes	Sender	Subject
1	K	1030	asw	Changes to MINIX
2	KA	6348	trudy	Not all Trudys are nasty
3	K F	4519	Amy N. Wong	Request for information
4		1236	bal	Bioinformatics
5		104110	kaashoek	Material on peer-to-peer
6		1223	Frank	Re: Will you review a grant proposal
7		3110	guido	Our paper has been accepted
8		1204	dmr	Re: My student's visit

An example display of the contents of a mailbox.

In the basic example, the first field is the message number. The second field, Flags, can contain a K, meaning that the message is not new, but was read previously and kept in the mailbox; an A, meaning that the message has already answered; and /or an F, meaning that the message has been forwarded to someone else. Other flags are also possible.

The third field tells how long the message is, and the fourth one tells who sent the message. Finally, a Subject field gives a brief summary of what the message is about. The older systems were text based and typically used one-character commands for performing these tasks, such as T(type message), A(answer message), D(delete message), F(forward message).

Message Formats : Messages consist of a primitive envelope (described in RFC 821), some number of header fields, a blank line, and then the message body. Each header field (logically) consists of a single line of ASCII text containing the field name, a colon, and, for most fields, a value.

Header	Meaning
To:	E-mail address(es) of primary recipient(s)
Cc:	E-mail address(es) of secondary recipient(s)
Bcc:	E-mail address(es) for blind carbon copies
From:	Person or people who created the message
Sender:	E-mail address of the actual sender
Received:	Line added by each transfer agent along the route
Return-Path:	Can be used to identify a path back to the sender

Figure 5-5: RFC 822 header fields related to message transport

The Principal header fields related to message transport are listed in the above figure. The To: field gives the DNS address of the primary recipient. Having multiple recipients is also allowed. The Cc: field gives the addresses of any secondary recipients. The Bcc: (Blind Carbon copy) field is like the Cc: field, except that this line is deleted from all the copies sent to primary and secondary recipients. This feature allows people to send copies to third parties without the primary and secondary recipients knowing this.

The next two fields, From: and Sender:, tell who wrote and sent the message, respectively. These need not be same. The From: field is required, but the Sender: field may be omitted if it is the same as the From: field. These fields are needed in case the message is undeliverable and must be returned to the sender.

A line containing Received: is added by each message transfer agent along the way. The line contains the agent's identity, the date and time the message was received and other information that can be used for finding bugs in the routing system. The Return-path: field is added by the final message transfer agent and was intended to tell how to get back to sender.

The Date: field is the date and time the message was sent. Reply-to: field is sometimes used when neither the person composing the message nor the person sending the message wants to see the reply.

Header	Meaning
To:	E-mail address(es) of primary recipient(s)
Cc:	E-mail address(es) of secondary recipient(s)
Bcc:	E-mail address(es) for blind carbon copies
From:	Person or people who created the message
Sender:	E-mail address of the actual sender
Received:	Line added by each transfer agent along the route
Return-Path:	Can be used to identify a path back to the sender

MIME-The Multipurpose Internet Mail Extensions : RFC 822 specified the headers but left the content entirely up to the users. Nowadays, on the worldwide Internet, this approach is no longer adequate. The problems include sending and receiving

1. Messages in languages with accents (e.g., French and German).
2. Messages in non-Latin alphabets (e.g., Hebrew and Russian).
3. Messages in languages without alphabets (e.g., Chinese and Japanese).
4. Messages not containing text at all (e.g., audio or images).

A solution was proposed in RFC 1341 called **MIME (Multipurpose Internet Mail Extensions)**

The basic idea of MIME is to continue to use the RFC 822 format, but to add structure to the message body and define encoding rules for non-ASCII messages. By not deviating from RFC 822, MIME messages can be sent using the existing mail programs and protocols. All that has to be changed are the sending and receiving programs, which users can do for themselves.

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

MIME defines five new message headers as shown in the above figure. First field describes about which version of MIME it uses. Next the Content-Description: is an ASCII string telling what is in the message. The Content-Id: header identifies the content. It uses the same format as the standard Message-Id header.

The Content-Transfer-Encoding: tells how the body is wrapped for transmission through a network that may object to most characters other than letters, numbers, and punctuation marks. The last header which specifies the nature of the message body.

Type	Subtype	Description
Text	Plain	Unformatted text
	Enriched	Text including simple formatting commands
Image	Gif	Still picture in GIF format
	Jpeg	Still picture in JPEG format
Audio	Basic	Audible sound
Video	Mpeg	Movie in MPEG format
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in PostScript
Message	Rfc822	A MIME RFC 822 message
	Partial	Message has been split for transmission
	External-body	Message itself must be fetched over the net
Multipart	Mixed	Independent parts in the specified order
	Alternative	Same message in different formats
	Parallel	Parts must be viewed simultaneously
	Digest	Each part is a complete RFC 822 message

- ✓ Text type is for straight ASCII Text. Text/plain is for unformatted text. Text/Enriched subtype allows including simple formatting commands.
- ✓ Image-GIF and JPEG are used to transmit still pictures.
- ✓ The audio and video types are for sound and moving pictures respectively.
- ✓ An Octet-stream in application is just a sequence of uninterpreted bytes.
- ✓ Post-script subtype in application used for describing printed pages.
- ✓ The message type allows one message to be fully encapsulated inside other if we use RFC822 subtype also.
- ✓ The partial subtype makes it possible to break an encapsulated message into pieces and send them separately.
- ✓ The external-body subtype can be used for very long messages.
- ✓ The final type is multipart, which allows a message to contain more than one part, with the beginning and end of each part being clearly delimited.
- ✓ The mixed subtype allows each part to be different, with no additional structure imposed.
- ✓ The alternative subtype allows each message to be included multiple times but expressed in two or more different media.
- ✓ The parallel subtype is used when all parts must be viewed simultaneously.
- ✓ The digest subtype is used when many messages are packed together into a composite message.

Message Transfer: The message transfer system is concerned with relaying messages from the originator to the recipient. The simplest way to do this is to establish a transport connection from the source machine to the destination machine and then just transfer the message.

SMTP—(THE SIMPLE MAIL TRANSFER PROTOCOL) : SMTP is a simple ASCII protocol.

After establishing the TCP connection to port 25, the sending machine, operating as the client, waits for the receiving machine, operating as the server, to talk first. The server starts by sending a line of text giving its identity and telling whether it is prepared to receive mail. If it is not, the client releases the connection and tries again later.

If the server is willing to accept e-mail, the client announces whom the e-mail is coming from and whom it is going to. If such recipient exists at the destination, the server gives the client the go-ahead to send the message. Then the client sends the message and the server acknowledges it.

Even though the SMTP protocol is completely well defined, a **few problems** can still arise.

One problem relates to message length. Some older implementations cannot handle messages exceeding 64 KB.

Another problem relates to timeouts. If the client and server have different timeouts, one of them may give up while the other is still busy, unexpectedly terminating the connection.

Finally, in rare situations, infinite mailstorms can be triggered.

For example, if host 1 holds mailing list *A* and host 2 holds mailing list *B* and each list contains an entry for the other one, then a message sent to either list could generate a never-ending amount of e-mail traffic unless somebody checks for it.

Final Delivery: As we saw, e-mail is delivered by having the sender establish a TCP connection to the receiver and then ship the e-mail over it.

This model worked fine for decades when all ARPANET hosts were, in-fact, on-line all the time to accept TCP connections.

With the advent of people who access the Internet by calling their ISP over a modem, it breaks down.

One solution is to have a message transfer agent on an ISP machine accept e-mail for its customers and store it in their mailboxes on an ISP machine. Since this agent can be on-line all the time, e-mail can be sent to it within 24 hours.

POP3: POP3 begins when the user starts the mail reader. The mail reader calls up the ISP (unless there is already a connection) and establishes a TCP connection with the message transfer agent at port 110. Once the connection has been established, the POP3 protocol goes through three states in sequence:

1. Authorization.
2. Transactions.
3. Update.

The authorization state deals with having the user log in.

The transaction state deals with the user collecting the e-mails and marking them for deletion from the mailbox.

The update state actually causes the e-mails to be deleted.

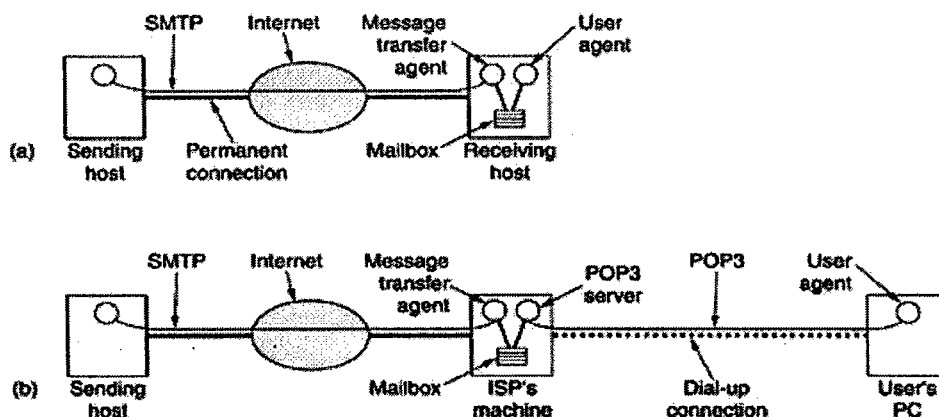


Figure:5-7

(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent.

(b) Reading e-mail when the receiver has a dial-up connection to an ISP

IMAP: POP3 normally downloads all stored messages at each contact, the result is that the user's e-mail quickly gets spread over multiple machines, more or less at random; some of them not even the user's.

Internet Message Access Protocol (IMAP) is a standard protocol for accessing email on a remote server from a local client. IMAP is an application layer Internet Protocol using the underlying transport layer protocols to establish host-to-host communication services for applications. This allows the use of a remote mail server. The well-known port address for IMAP is 143.

The IMAP architecture enables users to send and receive emails through a remote server, without support from a particular device. This type of email access is ideal for travelers receiving or answering emails from their home desktop or office computer.

This term is also known as interactive mail access protocol, Internet mail access protocol, and interim mail access protocol

This disadvantage gave rise to an alternative final delivery protocol, **IMAP (Internet Message Access Protocol)**.

IMAP assumes that all the e-mail will remain on the server indefinitely in multiple mailboxes.

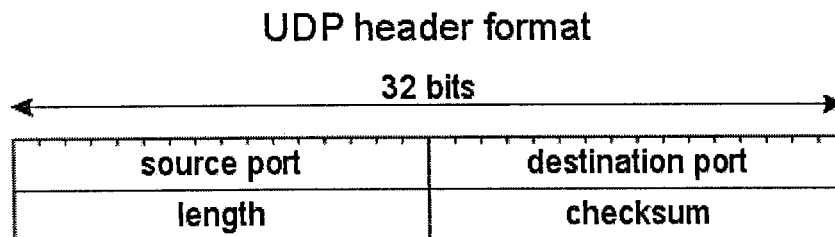
IMAP provides extensive mechanisms for reading messages or even parts of messages, a feature useful when using a slow modem to read the text part of a multipart message with large audio and video attachments.

Transport Layer

The Internet Transport Protocols: UDP: UDP is a standard protocol with STD number 6. UDP is described by RFC 768 – User Datagram Protocol. Its status is recommended, but in practice every TCP/IP implementation that is not used exclusively for routing will include UDP. UDP is basically an application interface to IP. It adds no reliability, flow-control, or error recovery to IP. It simply serves as a multiplexer/demultiplexer for sending and receiving datagrams, using ports to direct the datagrams.

UDP is connectionless and unreliable protocol. It doesn't require making a connection with the host to exchange data. Since UDP is unreliable protocol, there is no mechanism for ensuring that data sent is received.

UDP transmits the data in form of a datagram. The UDP datagram consists of five parts as shown in the following diagram:



Where:

- Source Port: Indicates the port of the sending process. It is the port to which replies should be addressed
- Destination Port: Specifies the port of the destination process on the destination host.
- Length: The length (in bytes) of this user datagram, including the header.
- Checksum: An optional 16-bit one's complement of the one's complement sum of a pseudo-IP header, the UDP header, and the UDP data.

UDP does not do flow control, error control, or retransmission upon receipt of a bad segment. One area where UDP is especially useful is in client-server situations. An application that uses UDP this way is DNS.

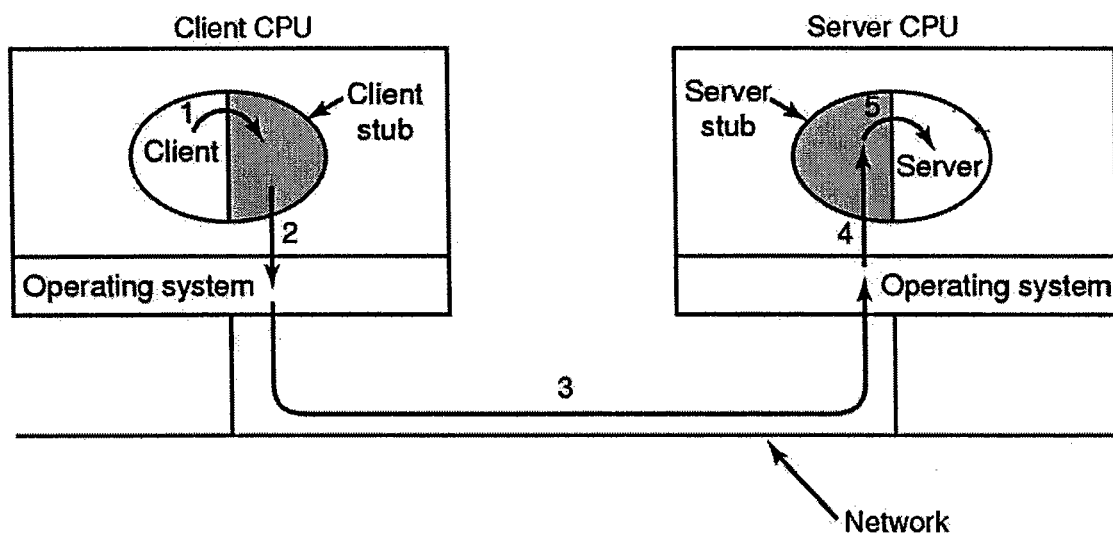
Remote Procedure Call: A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it

sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call is given as follows:

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.



Real-Time Transport protocol (RTP): Real-Time Transport Protocol (RTP) is an Internet Protocol standard that specifies the way programs manage the real-time transmission of multimedia data over unicast or multicast network services.

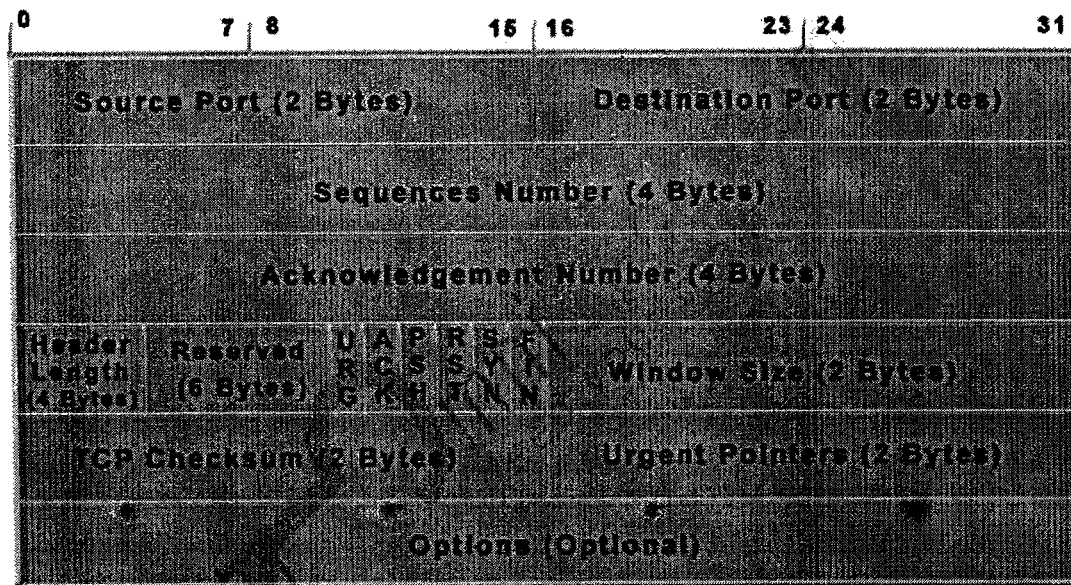
In comparison to TCP (Transmission Control Protocol) which favors data integrity rather than delivery speed, RTP favors rapid delivery and has mechanisms to compensate for any minor loss of data integrity.

RTP defines the standardized packet format for delivering audio and video over IP networks and used in conjunction with Real-Time Transport Control Protocol (RTCP) to ensure that multiple streams of media can be synchronized and Quality of Service (QoS) can be maintained.

Real-Time Transport Protocol is used extensively in entertainment and communication systems that involve streaming video such as video teleconference applications and Voice over Internet Protocol.

TCP (Transmission Control Protocol): TCP is a standard protocol with STD number 7. TCP is described by RFC 793. Transmission Control Protocol. Its status is recommended, but in practice, every TCP/IP implementation that is not used exclusively for routing will include TCP. TCP provides considerably more facilities for applications than UDP, notably error recovery, flow control, and reliability. TCP is a connection-oriented protocol, unlike UDP, which is connectionless. Most of the user application protocols, such as Telnet and FTP, use TCP.

TCP Segment Header: We will learn more about TCP Segment Header, different fields in TCP Header and the use of these fields.



Source port: 16 Bit number which identifies the Source Port number (Sending Computer's TCP Port).

Destination port: 16 Bit number which identifies the Destination Port number (Receiving Port).

Sequence number: 32 Bit number used for byte level numbering of TCP segments. If you are using TCP, each byte of data is assigned a sequence number.

Acknowledgment Number: 32 Bit number field which indicates the next sequence number that the sending device is expecting from the other device.

Header Length: 4 Bit field which shows the number of 32 Bit words in the header. also known as the Data Offset field. The minimum size header is 5 words (binary pattern is 0101).

Reserved: Always set to 0 (Size 6 bits).

Control Bit Flags: We have seen before that TCP is a Connection Oriented Protocol. The meaning of Connection Oriented Protocol is that, before any data can be transmitted, a reliable connection must be obtained and acknowledged. Control Bits govern the entire process of connection

establishment, data transmissions and connection termination. The control bits are listed as follows: They are:

URG: Urgent Pointer.

ACK: Acknowledgement.

PSH: This flag means Push function. Using this flag, TCP allows a sending application to specify that the data must be pushed immediately. When an application requests the TCP to push data, the TCP should send the data that has accumulated without waiting to fill the segment.

RST: Reset the connection. The RST bit is used to RESET the TCP connection due to unrecoverable errors. When an RST is received in a TCP segment, the receiver must respond by immediately terminating the connection. A RESET causes both sides immediately to release the connection and all its resources. As a result, transfer of data ceases in both directions, which can result in loss of data that is in transit. A TCP RST indicates an abnormal termination of the connection.

SYN: This flag means synchronize sequence numbers. Source is beginning a new counting sequence. In other words, the TCP segment contains the sequence number of the first sent byte (ISN).

FIN: No more data from the sender. Receiving a TCP segment with the FIN flag does not mean that transferring data in the opposite direction is not possible. Because TCP is a fully duplex connection, the FIN flag will cause the closing of connection only in one direction. To close a TCP connection gracefully, applications use the FIN flag.

Window: indicates the size of the receive window, which specifies the number of bytes beyond the sequence number in the acknowledgment field that the receiver is currently willing to receive.

Checksum: The 16-bit checksum field is used for error-checking of the header and data.

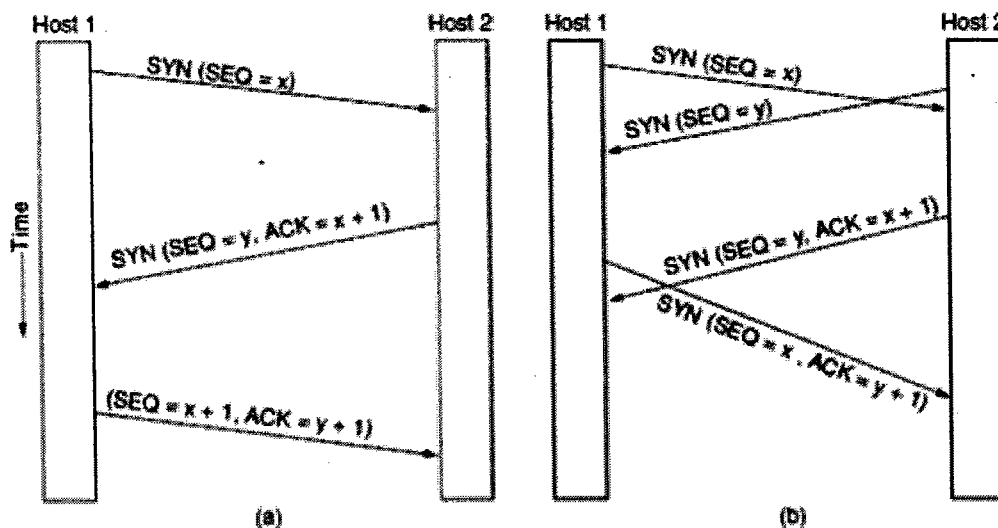
Urgent Pointer: Shows the end of the urgent data so that interrupted data streams can continue. When the URG bit is set, the data is given priority over other data streams (Size 16 bits).

TCP Connection Establishment: To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

- ✓ SYN: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
- ✓ SYN-ACK: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number ($A + 1$), and the sequence number that the server chooses for the packet is another random number, B.

- ✓ ACK: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. $A + 1$, and the acknowledgement number is set to one more than the received sequence number i.e. $B + 1$.

At this point, both the client and server have received an acknowledgment of the connection. The steps 1, 2 establish the connection parameter (sequence number) for one direction and it is acknowledged. The steps 2, 3 establish the connection parameter (sequence number) for the other direction and it is acknowledged. With these, a full-duplex communication is established.



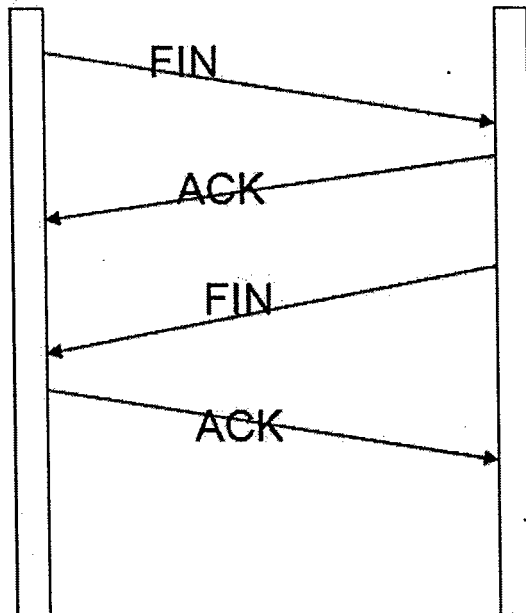
a) TCP connection establishment in the normal case. (b) Simultaneous connection establishment on both sides.

TCP Connection Release : The connection termination phase uses a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint. After both FIN/ACK exchanges are concluded, the side which sent the first FIN before receiving one waits for a timeout before finally closing the connection, during which time the local port is unavailable for new connections; this prevents confusion due to delayed packets being delivered during subsequent connections.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into the connection, but the other side can. The terminating side should continue reading the data until the other side terminates as well.

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK. This is perhaps the most common method.

It is possible for both hosts to send FINs simultaneously then both just have to ACK. This could possibly be considered a 2-way handshake since the FIN/ACK sequence is done in parallel for both directions.



TCP Connection Management Modeling: The steps required establishing and release connections can be represented in a finite state machine with the 11 states listed in Fig. 4.13. In each state, certain events are legal. When a legal event happens, some action may be taken. If some other event happens, an error is reported.

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

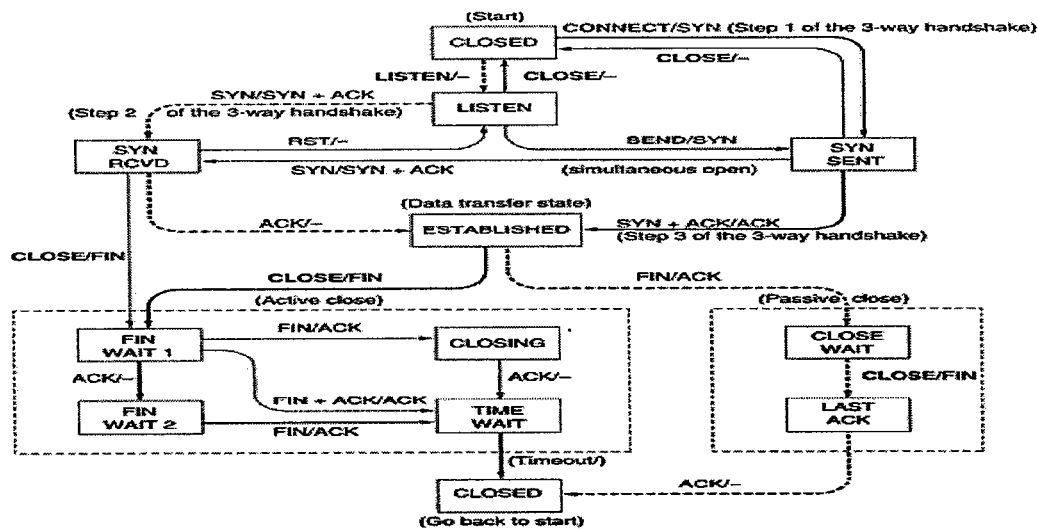


Figure 4.14 - TCP connection management finite state machine.

TCP Connection management from server's point of view:

1. The server does a **LISTEN** and settles down to see who turns up.
2. When a **SYN** comes in, the server acknowledges it and goes to the **SYNRCVD** state
3. When the servers **SYN** is itself acknowledged the 3-way handshake is complete and server goes to the **ESTABLISHED** state. Data transfer can now occur.
4. When the client has had enough, it does a close, which causes a **FIN** to arrive at the server [dashed box marked passive close].
5. The server is then signaled.
6. When it too, does a **CLOSE**, a **FIN** is sent to the client.
7. When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.

TCP Transmission Policy:

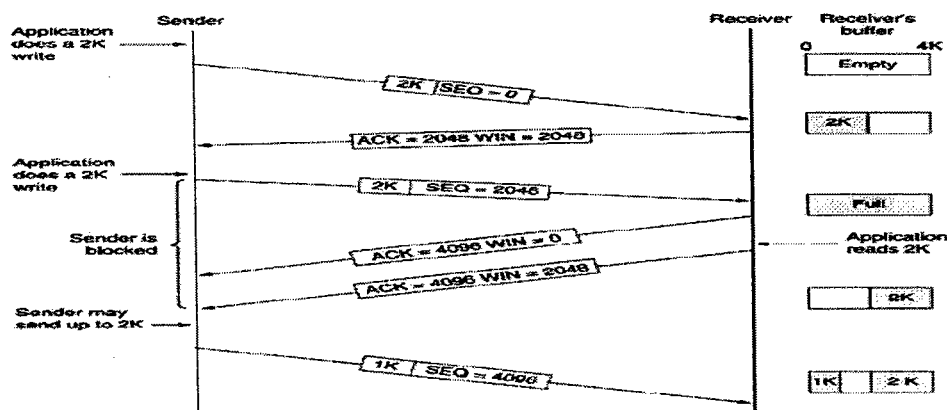


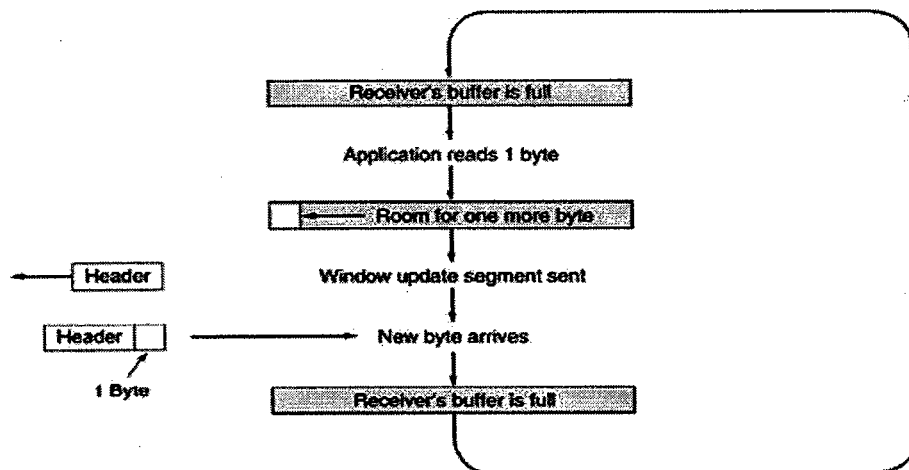
Figure 4.15 - Window management in TCP.

1. In the above example, the receiver has 4096-byte buffer.
2. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
3. Now the receiver will advertise a window of 2048 as it has only 2048 of buffer space, now.
4. Now the sender transmits another 2048 bytes which are acknowledged, but the advertised window is '0'.
5. The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a larger window.

SILLY WINDOW SYNDROME:

This is one of the problems that ruin the TCP performance, which occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads 1 byte at a time.

- ✓ Initially the TCP buffer on the receiving side is full and the sender knows this(win=0)
- ✓ Then the interactive application reads 1 character from tcp stream.
- ✓ Now, the receiving TCP sends a window update to the sender saying that it is all right to send 1 byte.
- ✓ The sender obligates and sends 1 byte.
- ✓ The buffer is now full, and so the receiver acknowledges the 1 byte segment but sets window to zero. This behavior can go on forever.



TCP CONGESTION CONTROL: TCP does to try to prevent the congestion from occurring in the first place in the following way:

When a connection is established, a suitable window size is chosen and the receiver specifies a window based on its buffer size. If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end. But they may still occur due to internal congestion within the network. Let's see this problem occurs.

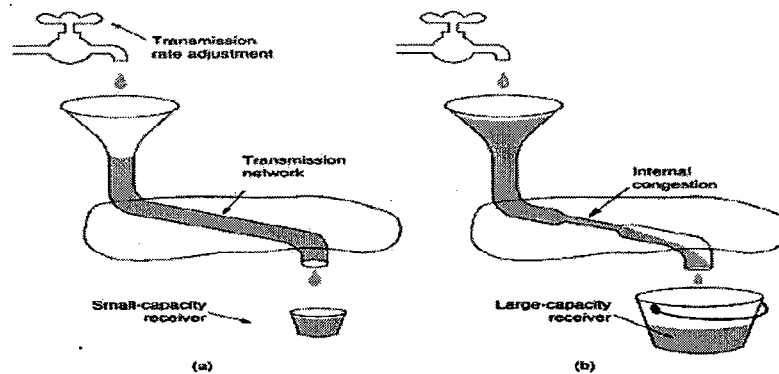


Figure 4.16. (a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver.

In fig (a): We see a thick pipe leading to a small- capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost.

In fig (b): The limiting factor is not the bucket capacity, but the internal carrying capacity of the n/w. if too much water comes in too fast, it will backup and some will be lost.

- ✓ When a connection is established, the sender initializes the congestion window to the size of the max segment in use our connection.
- ✓ It then sends one max segment .if this max segment is acknowledged before the timer goes off, it adds one segment s worth of bytes to the congestion window to make it two maximum size segments and sends 2 segments.
- ✓ As each of these segments is acknowledged, the congestion window is increased by one max segment size.
- ✓ When the congestion window is 'n' segments, if all 'n' are acknowledged on time, the congestion window is increased by the byte count corresponding to 'n' segments.
- ✓ The congestion window keeps growing exponentially until either a time out occurs or the receiver's window is reached.
- ✓ The internet congestion control algorithm uses a third parameter, the "threshold" in addition to receiver and congestion windows.

Different congestion control algorithms used by TCP are:

- ✓ RTT variance Estimation.
- ✓ Exponential RTO back-off
- ✓ Karn's Algorithm
- ✓ Slow Start
- ✓ Dynamic window sizing on congestion
- ✓ Fast Retransmit.
- ✓ Fast Recovery.

TCP TIMER MANAGEMENT: TCP uses 3 kinds of timers:

1. Retransmission timer
2. Persistence timer
3. Keep-Alive timer.

1. Retransmission timer: When a segment is sent, a timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer is started again.

2. Persistence timer:

It is designed to prevent the following deadlock:

- ✓ The receiver sends an acknowledgement with a window size of '0' telling the sender to wait later, the receiver updates the window, but the packet with the update is lost now both the sender and receiver are waiting for each other to do something
- ✓ when the persistence timer goes off, the sender transmits a probe to the receiver the response to the probe gives the window size
- ✓ if it is still zero, the persistence timer is set again and the cycle repeats
- ✓ if it is non zero, data can now be sent.

3. Keep-Alive timer: When a connection has been idle for a long time, this timer may go off to cause one side to check if other side is still there. If it fails to respond, the connection is terminated.

