UNIT IV

APPLICATION LAYER

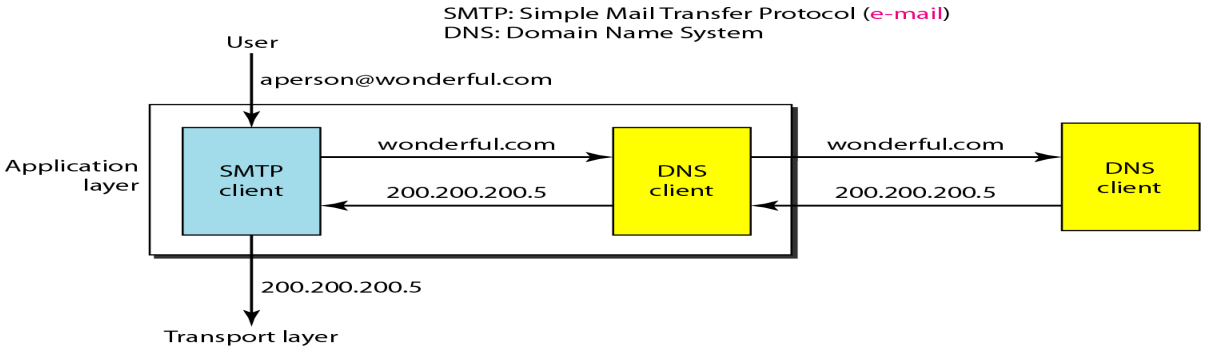
DOMAIN NAME SYSTEM

The client/server programs can be divided into two categories:

1. Programs that can be directly used by user.
2. Programs that support other application programs.

Domain Name System (DNS) is a supporting program that is used by other programs such as E-mail.

Basic concept of DNS



The above figure shows a DNS client/server program can support an E-mail program to find the IP address of an E-mail recipient.

- A user of an E-mail program knows the E-mail address of the recipient but the IP protocol needs the IP address.
- The DNS client program sends a request to a DNS server to map the E-mail address to the corresponding IP address.
- To identify an entity TCP/IP protocols uses the IP address, which uniquely identifies the connection of a host to the Internet.
- People prefer to use names instead of numeric addresses. Hence we need a system that can map a name to an address or an address to a name. DNS is designed for this purpose.

NAMESPACE

The names assigned to machines must be unique because the addresses are unique.

A name space that maps each address to a unique name can be organized in two ways:

- Flat Name Space
- Hierarchical Name Space

Flat Name Space

- In a flat name space, a name is assigned to an address.
- A name in this space is a sequence of characters without structure.

Hierarchical Name Space

- In Hierarchical name space, each name is made of several parts.

- The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization etc.

Example:

Assume three Education institutions named one of their computers **Challenger**. The three colleges have given names by the central authority such as iitm.ac.in, berkeley.edu and smart.edu.

When these organizations add the name **Challenger** the names will be :

- challenger.iitm.ac.in
- challenger.berkeley.edu
- challenger.smart.edu

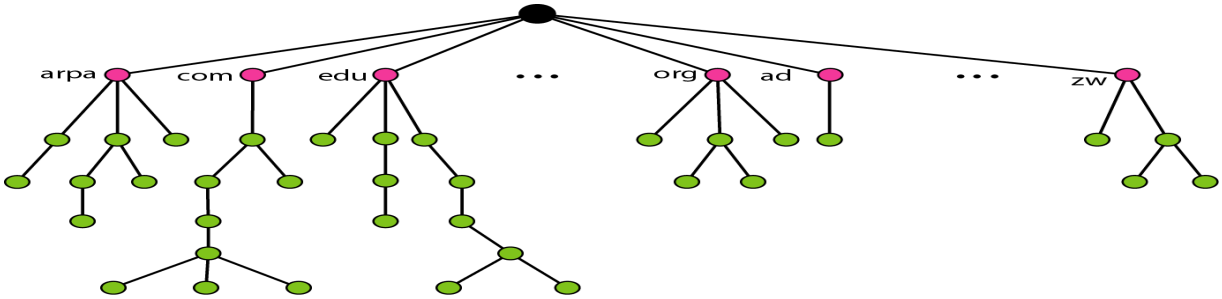
DOMAIN NAME SPACE

A domain name space was designed to have a Hierarchical Name Space.

In this design the names are defined in a tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127.

Label

- Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string).
- DNS requires that children of a node have different labels, which guarantees the uniqueness of the domain names.

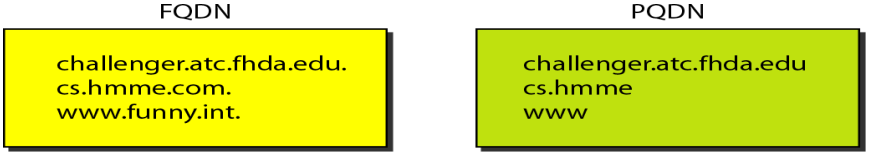


Domain Name

- Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots(.).
- Domain names are always read from the bottom to top.
- The last label is the label of the root (null). (i.e.) a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing.

Fully Qualified Domain Name (FQDN)

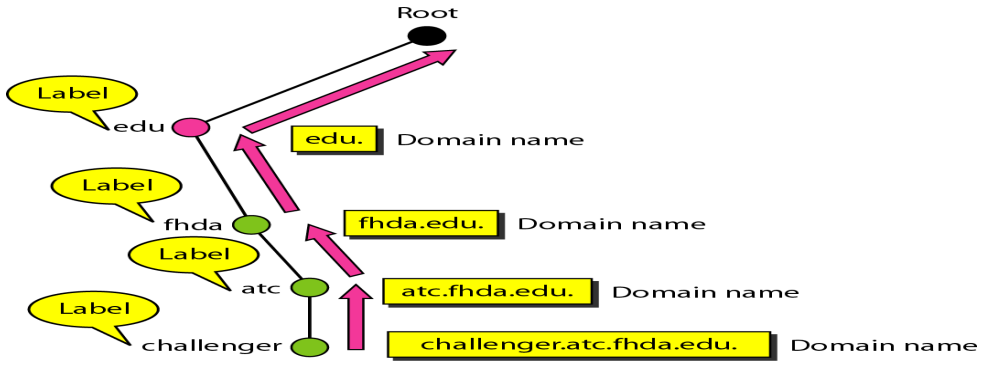
- If a label is terminated by a null string, it is called a fully qualified domain name(FQDN).
- Example:**challenger.atc.fhda.edu.**



Partially Qualified Domain Name (PQDN)

- If a label is not terminated by a null string, it is called a PQDN.
- Example:**challenger**

If a user at the **"fhda.edu."** site wants to get the IP address of the challenger computer, he or she can define the partial name "**challenger**".
The DNS client adds the suffix **"atc.fhda.edu."** before passing the address to the DNS server.



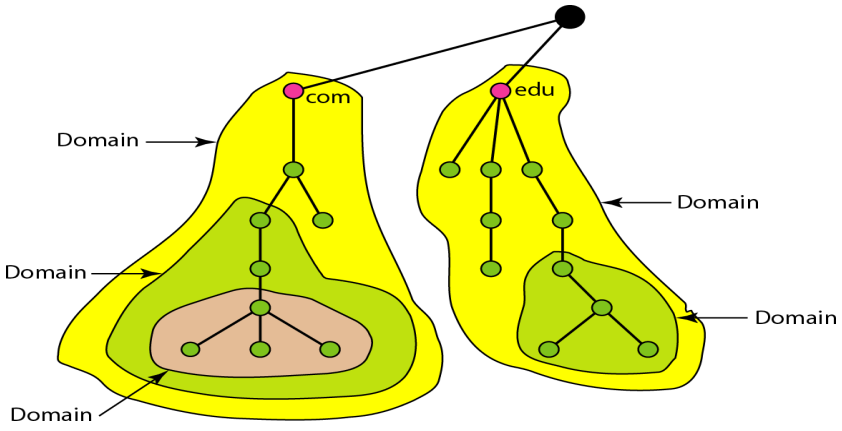
Note: The DNS client normally holds a list of suffixes such as:

- **atc.fhda.edu**
  - **fhda.edu**
  - **null**

These suffixes were added when the user defines an FQDN.

Domain

A **domain** is a sub-tree of the domain name space. The name of the domain is the domain name of the node at the top of the sub tree. A domain may itself be divided into sub-domains.



DNS IN THE INTERNET

In the Internet the domain name space tree is divided into three different sections:

- Generic Domains,
- Country Domains
- The Inverse Domain

Generic Domains

- The **generic domains** define Registered hosts according to their generic behavior.
- Each node in the tree defines a domain, which is an index to the domain name space database.

Generic domain labels are listed as:

Label	Description
com	Commercial organizations
org	Nonprofit organizations
net	Network support centers
edu	Educational institutions
gov	Government institutions

Country Domains

- The country domains section uses two-character country abbreviations. Ex: in for India, us for USA.
- Second labels can be organizational, or they can be more specific, national designations. Ex: .ac.in for nptel.ac.in, .gov.in for tspsc.gov.in etc.

Inverse Domain

The inverse domain is used to map an Address to a Name. It uses inverse query or pointer query.

RESOLUTION

Mapping a name to an address or an address to a name is called Name-Address Resolution.

Caching

- When a server asks for a mapping from another server and receives the response, it stores this information in its cache memory before sending it to the client.
- If the client asks for the same mapping, it can check its cache memory and returns the result.
- To inform the client that the response is coming from the cache memory and not from an authoritative source, the server marks the response as Un authoritative.

REGISTRARS

Registrar adds new domains to DNS. A registrar first verifies that the requested domain name is unique and then enters it into the DNS database. The registrars and their names, addresses found at: http://www.intenic.net

Example: Domain name: WS.wonderful.com (ws is a server name) IP address: 200.200.200.5 (new IP address).

REMOTE LOGGING

In the Internet, users may want to run application programs at a remote site and create results that can be transferred to their local site.

Example: Students may want to connect to their university computer lab from their home to access application programs for doing homework assignments or projects.

A General purpose client/server program that allows a user to log-on to a remote computer to access any application program on that remote computer.

After logging on, a user can use the available services on the remote computer and transfer the results back to the local computer.

TELNET (TErminaL NETwork)

TELNET is a client/server application program. It is the standard TCP/IP protocol for virtual terminal service as proposed by the International Organization for Standards (ISO).

TELNET enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

Timesharing Environment

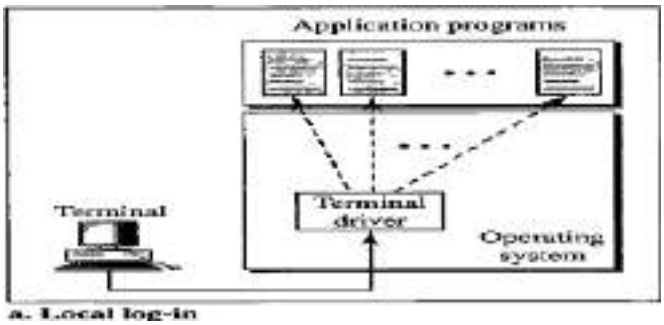
TELNET works in Time Sharing Environment. The interaction between a user and the computer occurs through a terminal.

Logging

- In a timesharing environment, users are part of the system with some right to access resources. Each authorized user has Identification (User ID) and a password.
- To access the system the user logs into the system with a user id or log-in name.
- The system also includes password checking to prevent an unauthorized user from accessing the resources.

Local log-in

- When a user logs into a local timesharing system it is called local log-in.
- As a user types at a terminal the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system.
- The operating system interprets the combination of characters and

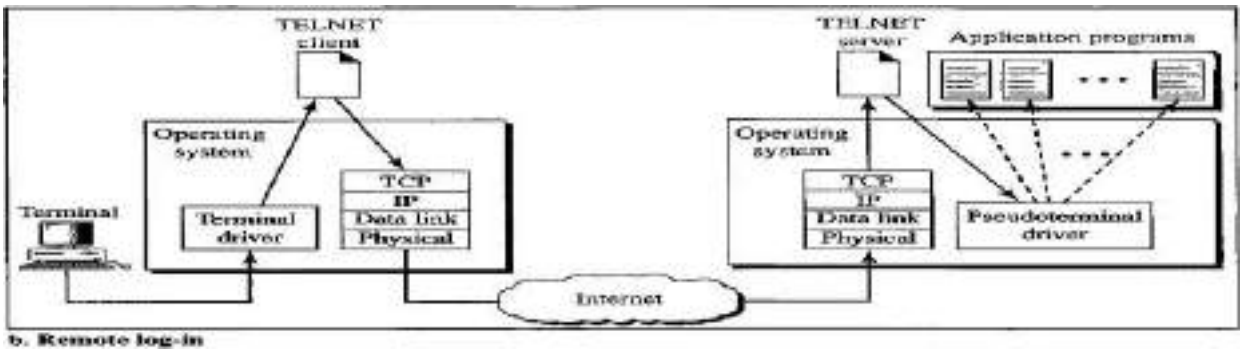


a. Local log-in

invokes the desired application program or utility.

Remote Log-in

- When a user wants to access an application program or utility located on a remote machine, the user performs remote log-in. TELNET uses client and server programs.
- The user sends the keystrokes to the terminal driver, where the local operating system accepts the characters but does not interpret them.
- The characters are sent to the TELNET client, which transforms the characters to a universal character set called Network Virtual Terminal (NVT) characters and delivers them to the local TCP/IP protocol stack.
- The commands or text in NVT form travel through the Internet and arrive at the TCP/IP stack at the remote machine.



b. Remote log-in

- The characters are delivered to the operating system and passed to the TELNET server, which changes the characters to the corresponding characters understandable by the remote computer.
- The characters cannot be passed directly to the operating system because the remote operating system is not designed to receive characters from a TELNET server. It is designed to receive characters from a terminal driver.
- Hence the characters can be passed to **Pseudo-terminal driver** which passes the characters to operating system.
- The operating system then passes the characters to the appropriate application program.

Network Virtual Terminal (NVT)

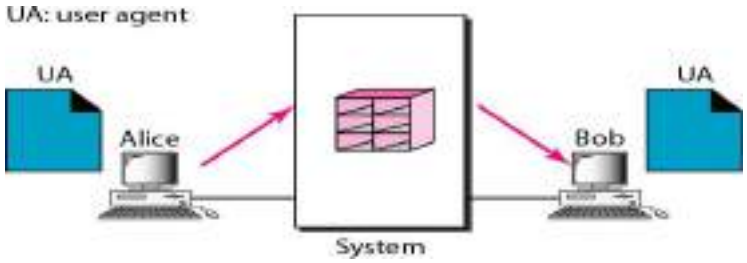
- TELNET defines a universal interface called the Network Virtual Terminal character set. Via this interface the client TELNET translates characters (data or commands) that come from the local terminal into NVT form and delivers them to the network.
- The server TELNET translates data and commands from NVT form into the form acceptable by the remote computer.

ELECTRONIC MAIL

Electronic mail (E-mail) is one of the most popular Internet services. E-mail allows a message to include text, audio, and video. There are Four Scenarios of E-mail:

First Scenario

- In the first scenario, the sender and the receiver of the E-mail are user application programs on the same system. They are directly connected to a shared system.
- The administrator has created one mailbox for each user where the received messages are stored.
- A mailbox is part of a local hard drive and it a special file with permission restrictions. Only the owner of the mailbox has access to it.

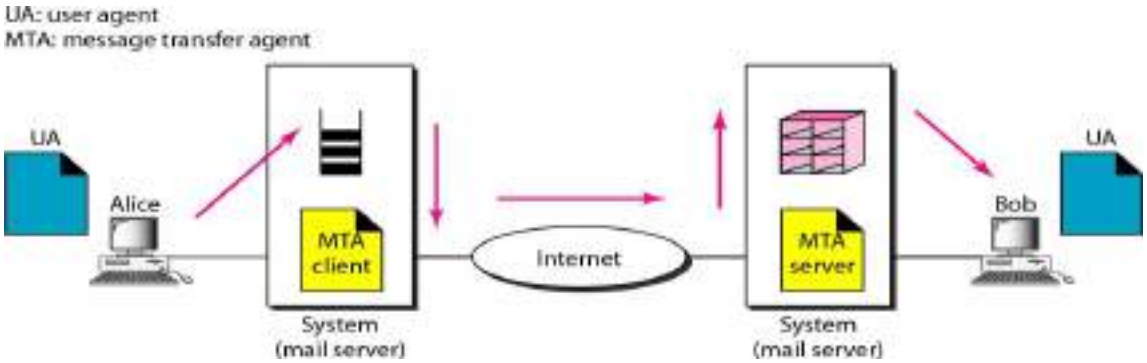


Example: Consider the above figure, Alice and Bob are two users of mail server.

- When a user Alice needs to send a message to Bob, Alice runs a User Agent (UA) program to prepare the message and store it in Bob's mailbox.
- The message has the sender and recipient mailbox addresses (names offiles).
- Bob can retrieve and read the contents of his mailbox using a User Agent.

Second Scenario

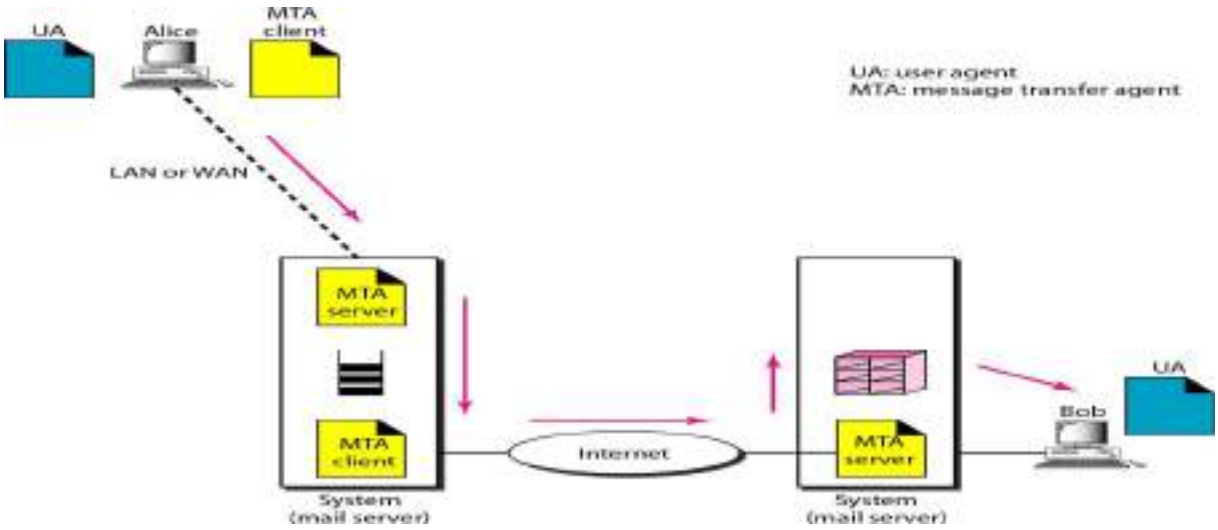
- In the second scenario, the sender and the receiver of the E-mail are user application programs on two different systems. The message needs to be sent over the Internet.
- We need User Agents (UAs) and Message Transfer Agents(MTA's).



- Alice needs to use a user agent program to send her message to the mail server at her own site.
- The mail server at Alice site uses a queue to store messages waiting to be sent.
- Bob also needs a user agent program to retrieve messages stored in the mailbox of the system at his site.

- The message needs to be sent through the Internet from Alice's site to Bob's site. Here two Message Transfer Agents are needed: one for client and one for server.
- Most client/server programs on the Internet, the server needs to run all the time because it does not know when a client will ask for a connection.
- The client can be alerted by the system when there is a message in the queue to be sent.

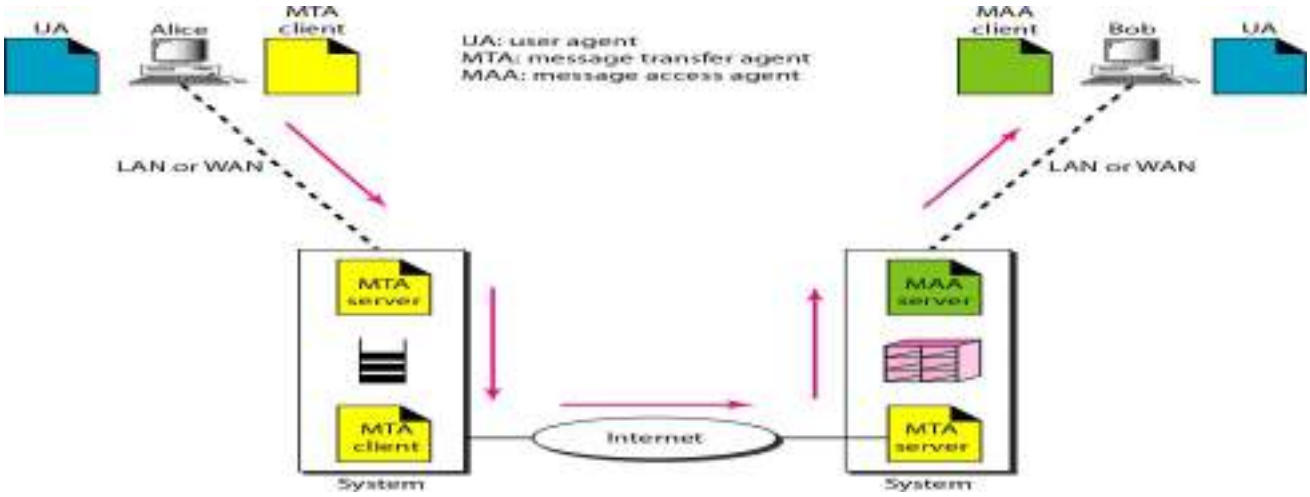
Third Scenario



- In the third scenario, Bob is directly connected to his system (i.e. Mail Server). Alice is separated from her system.
- Alice is connected to the mail server via WAN or LAN.
- In an organization that uses one mail server for handling E-mails, all users need to send their messages to this mail server.
- User agent of Alice prepares message and sends the message through the LAN or WAN.
- Whenever Alice has a message to send, Alice calls the user agent and user agent calls the MTA client.
- The MTA client establishes a connection with the MTA server on the system.
- The system at Alice's site queues all messages received. It then uses an MTA client to send the messages to the system at Bob's site. The system receives the message and stores it in Bob's mailbox.
- Bob uses his user agent to retrieve the message and reads it.
- It needs two MTA client and two MTA server programs.

Fourth Scenario

- It is the most common scenario, Alice and Bob both are connected to their mail server by a WAN or a LAN.
- After the message has arrived at Bob's mail server, Bob needs to retrieve it. Now Bob needs another set of client/server agents called Message Access Agents (MAA). Bob uses an MAA client to retrieve his messages.
- The client sends a request to the MAA server and requests the transfer of the messages.



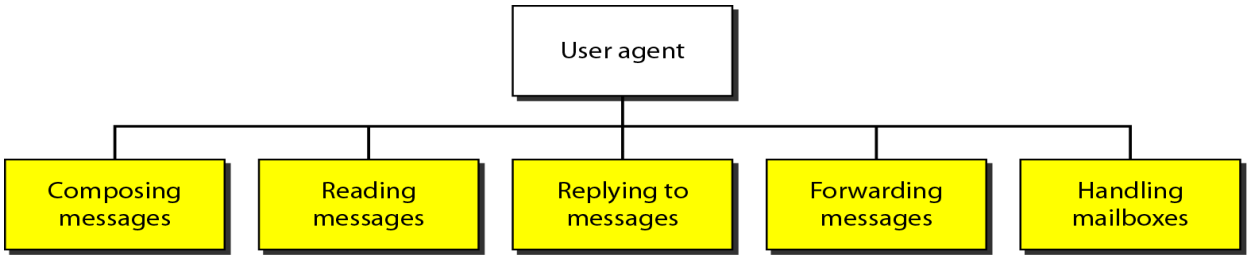
Architecture of E-mail

There are three major components in the architecture of E-mail:

1. User Agent
2. Message Transfer Agent
3. Message Access Agent

User Agent

User Agent provides services to the user to make the process of sending and receiving a message easier. Services provided by User agent are:



Composing Messages

A user agent helps the user to compose the E-mail message to be sent out. Most user agents provide a template on the screen to be filled in by the user.

Reading Messages

The user agent reads the incoming messages. When a user invokes a user agent, it first checks the mail in the incoming mailbox. Each E-mail contains the following fields:

1. A number field.
2. A flag field that shows the status of the mail such as new, already read but not replied to, or read and replied to.
3. The size of the message.
4. The sender.
5. The optional subject field.

Replying to Messages

After reading a message, a user can use the user agent to reply to a message. A user agent allows the user to reply to the original sender or to reply to all recipients of the message.

Forwarding Messages

It means sending a message to a third party. A user agent allows receiver to forward message.

Handling Mailboxes

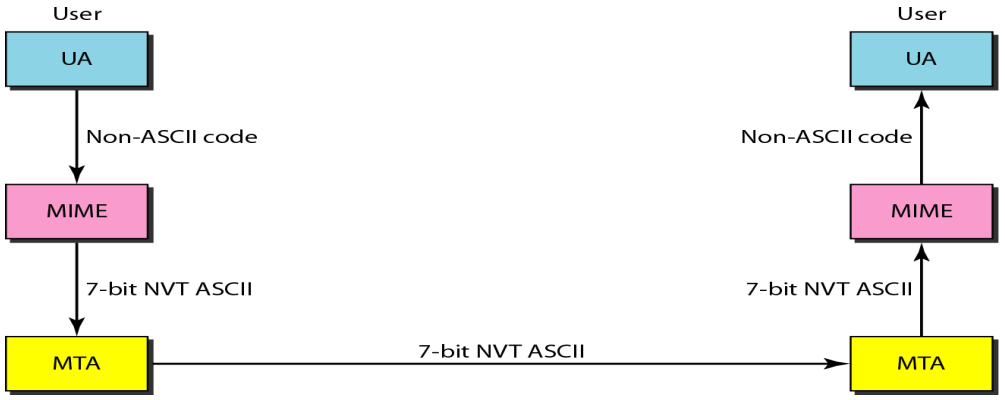
- A user agent normally creates two mailboxes: **Inbox** and **Outbox**.
- Inbox and Outbox is a file with a special format that can be handled by user agent.
- **Inbox** keeps all the received E-mails until they are deleted by the user.
- **Outbox** keeps all the sent E-mails until the user deletes them.

Mailing List

- Electronic mail allows one name (an alias) to represent several different E-mail addresses is called a mailing list.
- Every time a message is to be sent, the system checks the recipient's name against the alias database.

MIME (Multipurpose Internet Mail Extensions)

- MIME is a supplementary protocol that allows non-ASCII data to be sent through E-mail.
- French, German, Hebrew, Russian, Chinese, and Japanese are non-ASCII characters.
- MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers them to the client MTA to be sent through the Internet.
- The message at the receiving side is transformed back to the original data.



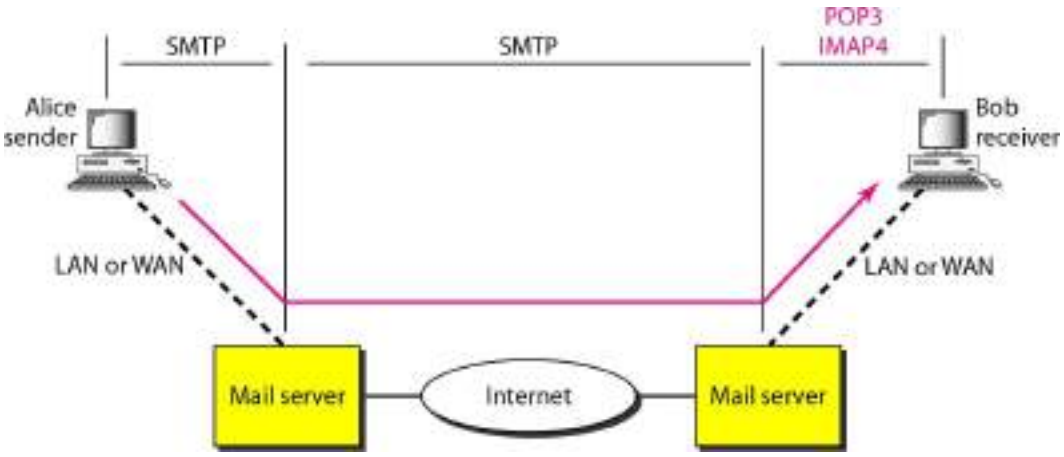
SIMPLE MAIL TRANSFER PROTOCOL (SMTP)

Message Transfer Agent: SMTP

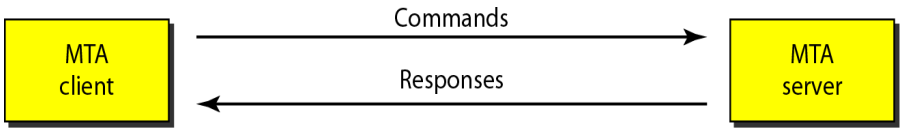
- The actual mail transfer is done through message transfer agents (MTA).
- Simple Mail Transfer Protocol defines the MTA Client and MTA server in the internet.
- MTA Client is used to send mail and MTA Server is used

to receive a mail. SMTP is used two times:

1. Between sender and sender mail server.
2. Between sender mail server and receiver mail server.



SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. Each command or reply is terminated by a two-character end-of-line token.



Commands

Commands are sent from the client to the server. It consists of a keyword followed by zero or more arguments.

There are five mandatory commands are used. Every implementation must support these five commands: HELO (Sender's Host name), MAIL FROM, RCPT TO, DATA, QUIT.

Responses

Responses are sent from the server to the client. A response is a three digit code that may be followed by additional textual information.

Responses are divided into four categories: The leftmost digit of the code 2, 3, 4, and 5 defines the category.

- 2-Positive Completion Reply
- 3-Positive Intermediate Reply
- 4-Transient Negative Completion Reply
- 5-Permanent Negative Completion Reply

Mail Transfer Phases

The process of transferring a mail message occurs in three phases:

- Connection Establishment
- Mail Transfer
- Connection Termination

FILE TRANSFER PROTOCOL (FTP)

File Transfer Protocol (FTP) is the standard mechanism provided by TCP/IP for copying a file from one host to another. FTP uses the services of TCP. FTP implemented to solve below problems.

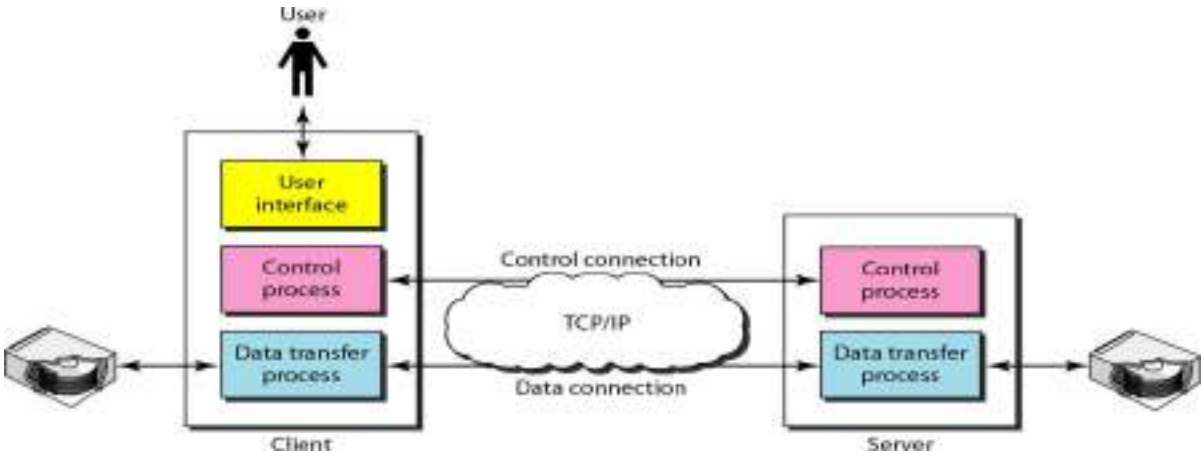
Problems with File transfer

- Two systems may use different file name conventions.
- Two systems may have different ways to represent text and data.
- Two systems may have different directory structures.

FTP differs from other client/server applications in that it establishes two connections between the hosts. FTP uses two well-known ports these connections.

1. Data transfer connection (Port 20 is used)
2. Control connection (Port 21 is used)

Basic Model of FTP



Consider the above figure that shows basic model of FTP that contains client and server components.

- Client has three components: User Interface, Client Control Process, and Client Data Transfer Process.
- Server has two components: Server Control Process and Server Data Transfer Process.
- The control connection is made between the control processes.
- The data connection is made between the data transfer processes.
- The control connection remains connected during the entire interactive FTP session.
- The data connection is opened and then closed for each file transferred.
- When a user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.

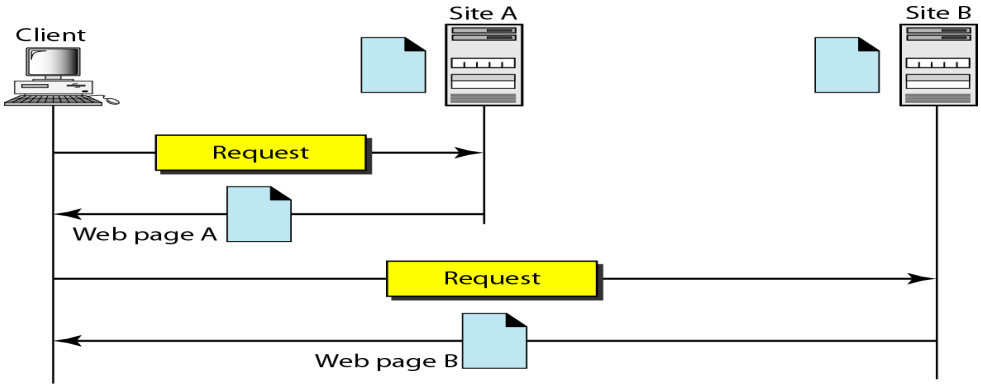
WORLD WIDE WEB (WWW)

World Wide Web (WWW) is a repository of information linked together from locations all over the world. WWW has a unique combination of flexibility, portability, and user-friendly features that distinguish it from other services provided by the Internet.

Architecture

The WWW is a distributed client/server service, in which a client using a browser can access a service using a server.

- The service provided is distributed over many locations called site.
- Each site holds one or more documents, referred to as Web pages.
- Each Web page can contain a link to other pages in the same site or at other sites is called Hyperlink.
- The pages can be retrieved and viewed by using browsers.



Architecture of WWW contains four parts: **1. Client 2. Server**
3.URL 4. Cookies Client(Browser)

A Client is a browser that interprets and displays a Web document.
Each browser consists of three parts: **Controller**, **Client protocol**, and **Interpreters.**

- The controller receives input from the keyboard or the mouse and uses the client programs to access the document.
- After the document has been accessed, the controller uses one of the interpreters to display the document on the screen.
- The interpreter can be HTML, Java, or JavaScript depending on the type of document.
- The client protocol can be FTP or HTTP.

Server

- The Web page is stored at the server. Each time a client request arrives, the corresponding document is sent to the client.
- To improve efficiency, servers normally store requested files in a cache in memory.
- A server uses multi-threading or multi-processing for answering more than one request at a time to increase the efficiency.

Uniform Resource Locator (URL)

The Uniform Resource Locator (URL) is a standard for specifying any kind of information on the Internet. A client that wants to access a Web page needs the address. To facilitate the access of documents distributed throughout the world, HTTP uses locators.

URL defines four things: Protocol, Host computer, Port, and Path.

- **Protocol:** It is the client/server program used to retrieve the document. Ex:FTP or HTTP.
- **Host:** The host is the computer on which the information is located. Web pages are usually stored in computers and computers are given **alias names** that usually begin with the characters "www". This is not mandatory.
- **Port:** The URL can optionally contain the port number of the server.
- **Path:** It is the pathname of the file where the information is located.

Note: The path can itself contain slashes that separate the directories from the subdirectories and files.

Cookies

Cookies are used to devise the following functionalities:

- Some websites need to allow access to registered clients only.
- Websites are being used as electronic stores (such as Flipkart or Amazon) that allow users to browse through the store, select wanted items, put them in an electronic cart, and pay at the end with a credit card.
- Some websites are used as portals: the user selects the Web pages he wants to see.
- Some websites are just advertising.

HYPERTEXT TRANSFER PROTOCOL (HTTP)

The Hypertext Transfer Protocol (HTTP) is a protocol used mainly to access data on the World Wide Web. HTTP uses the services of TCP on well-known port 80.

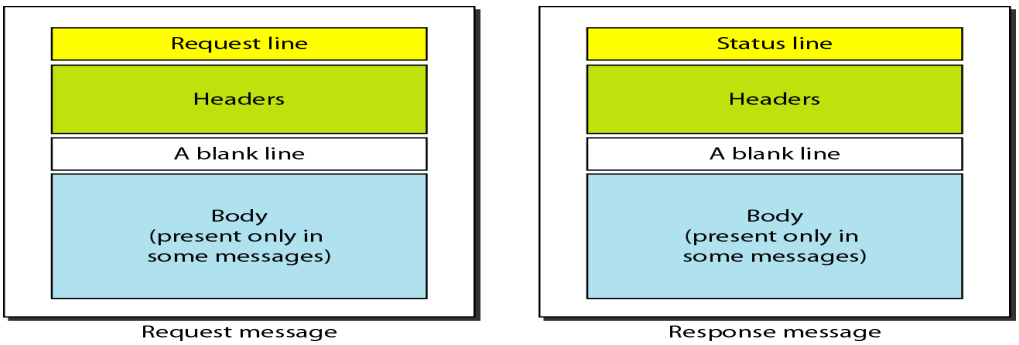
HTTP functions as a combination of FTP and SMTP.

HTTP Transaction

HTTP is a stateless protocol even though it uses TCP services. The client initializes the transaction by sending a request message. The server replies by sending a response.

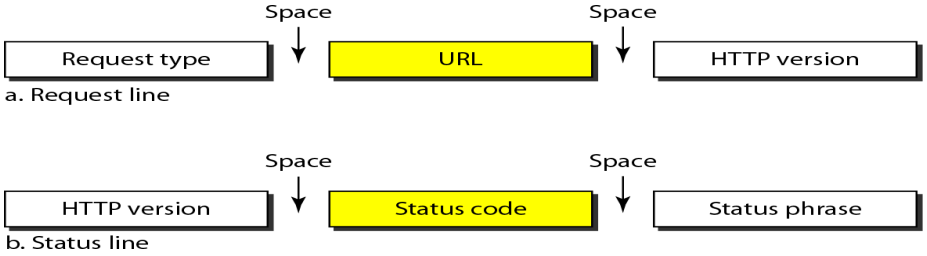
HTTP uses two types of messages: Request, Response

- A request message consists of a request line, a header, and optional body.
- A response message consists of a status line, a header, and optional body.



Request and Status Lines

- The first line in a request message is called a request line.
- The first line in the response message is called the status line.



Request type

This field is used in the request message. In version 1.1 of HTTP defines several request types. The request type is categorized into methods.

Methods	Action
GET	Requests a document from the server
POST	Sends some information from the client to the server
PUT	Sends a document from the server to the client

URL: By using URL, clients can access the webpage.

Version The most current version of HTTP is 1.1.

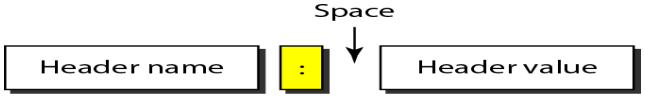
Status code is used in the response message. It consists of 3-digits. 100, 200, 300, 400, 500.

Status phrase: It explains the status code in text form, it is used in the response message.

Status Code	Status phrase	Description
100	Continue, switch	Informational
200	OK, CREATED, ACCEPTED	Successful request
300	Moved Permanently or temporarily, Not modified.	Redirect Client to another URL
400	400- Bad request, 404- Not found,	Error at client side
500	500- Internal server error 503-Service unavailable	Error at server side

Header

The header exchanges additional information between the client and the server. The header can consist of one or more header lines.



A Header line can be divided into 4 categories: 1. **General** 2.**Request** 3.**Response** 4.**Entity**.

1. A request message can contain Request header, General header, Entity header.
2. A response message can contain Response header, General header, Entity header.

General header

General header gives general information about the message such as Date, MIME version.

Request header

Request header specifies the client's configuration and the client's preferred documentformat. Example: Accept: Shows the medium format the client can accept

From: Shows the E-mail address of the user Host: Shows the host and port number of the server Referrer: Specifies the URL of the linked document User agent: Identifies the client program.

Response header

This header specifies the server's configuration and special information about the request. Example: Age: shows the age of the document, public: shows the supported list of methods, server: shows the server name and address.

Entity header

The entity header gives information about the body of the document. some request messages such as POST or PUT methods may contain a body also use this type of header.

Examples: E tag- Gives an entity tag, Content-type- Specifies the medium type, Last- modified- Gives the date and time of the last change etc.

Body

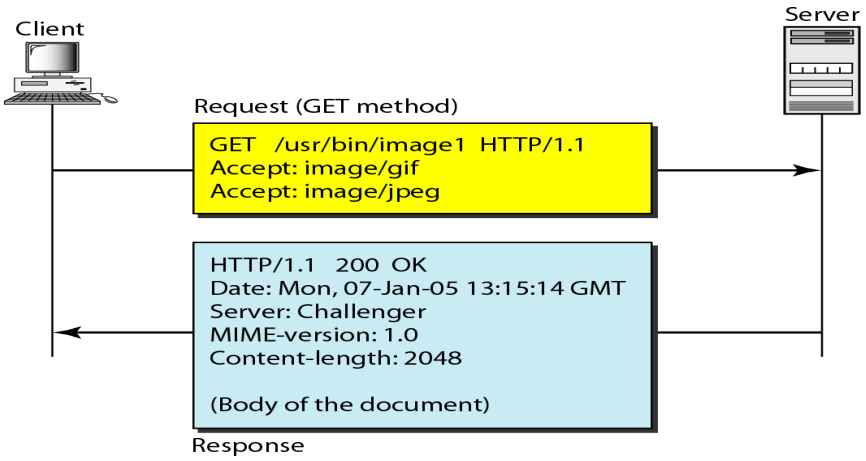
The body can be present in a request or response message. Body contains the document to be sent or received.

Example of HTTP transaction

Consider the below figure that shows how to retrieve a document.

- We use the GET method to retrieve an image with the path/usr/bin/imagel.
- The request line shows the method (GET), the URL, and the HTTP version(1.1).
- Header has two lines that show the client can accept images in the GIF or JPEG format.
- The request does not have a body.
- The response message contains the status line and four lines of header.
- The header lines define the date, server, MIME version, and length of the document.

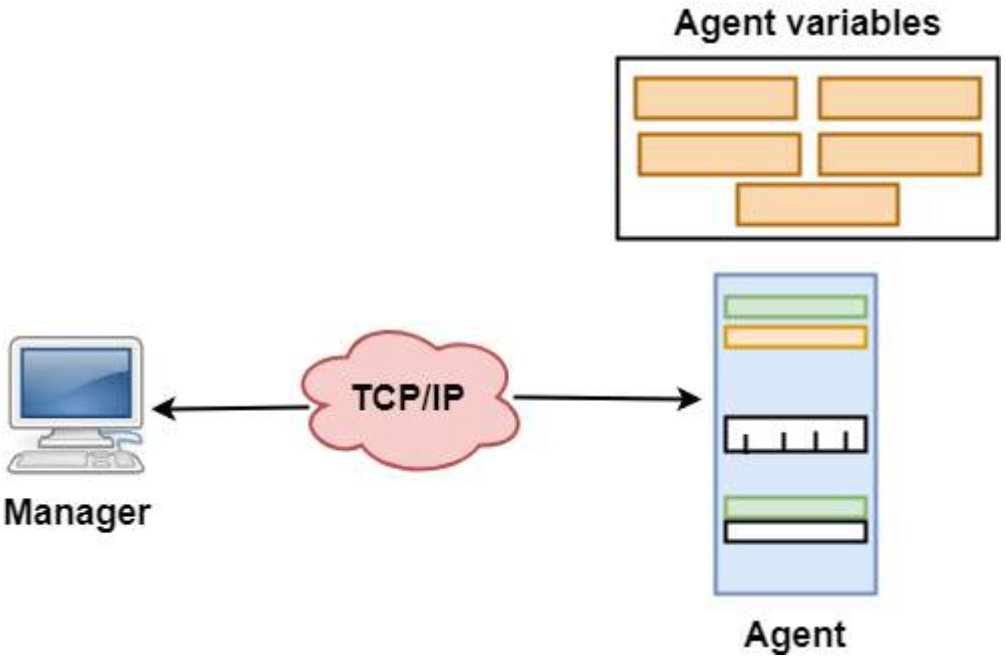
- The body of the document follows the header.



SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

- SNMP stands for **Simple Network Management Protocol**.
- SNMP is a framework used for managing devices on the internet.
- It provides a set of operations for monitoring and managing the internet.

SNMP Concept



- SNMP has two components Manager and agent.
- The manager is a host that controls and monitors a set of agents such as routers.
- It is an application layer protocol in which a few manager stations can handle a set of agents.
- The protocol designed at the application level can monitor the devices made by different manufacturers and installed on different physical networks.

- It is used in a heterogeneous network made of different LANs and WANs connected by routers or gateways.

Managers & Agents

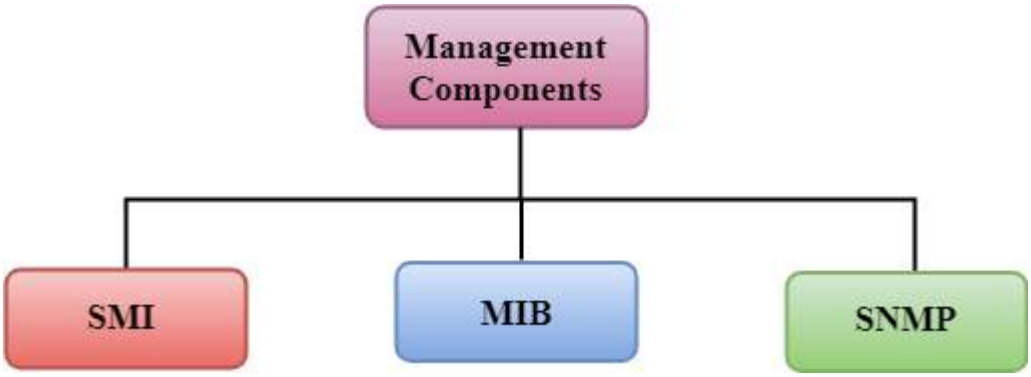
- A manager is a host that runs the SNMP client program while the agent is a router that runs the SNMP server program.
- Management of the internet is achieved through simple interaction between a manager and agent.
- The agent is used to keep the information in a database while the manager is used to access the values in the database. For example, a router can store the appropriate variables such as a number of packets received and forwarded while the manager can compare these variables to determine whether the router is congested or not.
- Agents can also contribute to the management process. A server program on the agent checks the environment, if something goes wrong, the agent sends a warning message to the manager.

Management with SNMP has three basic ideas:

- A manager checks the agent by requesting the information that reflects the behavior of the agent.
- A manager also forces the agent to perform a certain function by resetting values in the agent database.
- An agent also contributes to the management process by warning the manager regarding an unusual condition.

Management Components

- Management is not achieved only through the SNMP protocol but also the use of other protocols that can cooperate with the SNMP protocol. Management is achieved through the use of the other two protocols: SMI (Structure of management information) and MIB(management information base).
- Management is a combination of SMI, MIB, and SNMP. All these three protocols such as abstract syntax notation 1 (ASN.1) and basic encoding rules (BER).

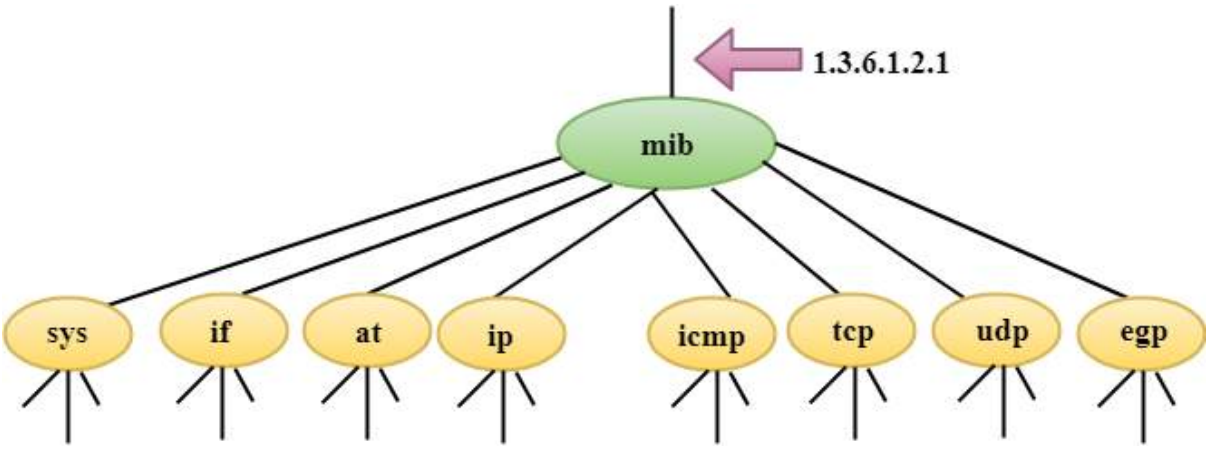


SMI

The SMI (Structure of management information) is a component used in network management. Its main function is to define the type of data that can be stored in an object and to show how to encode the data for the transmission over a network.

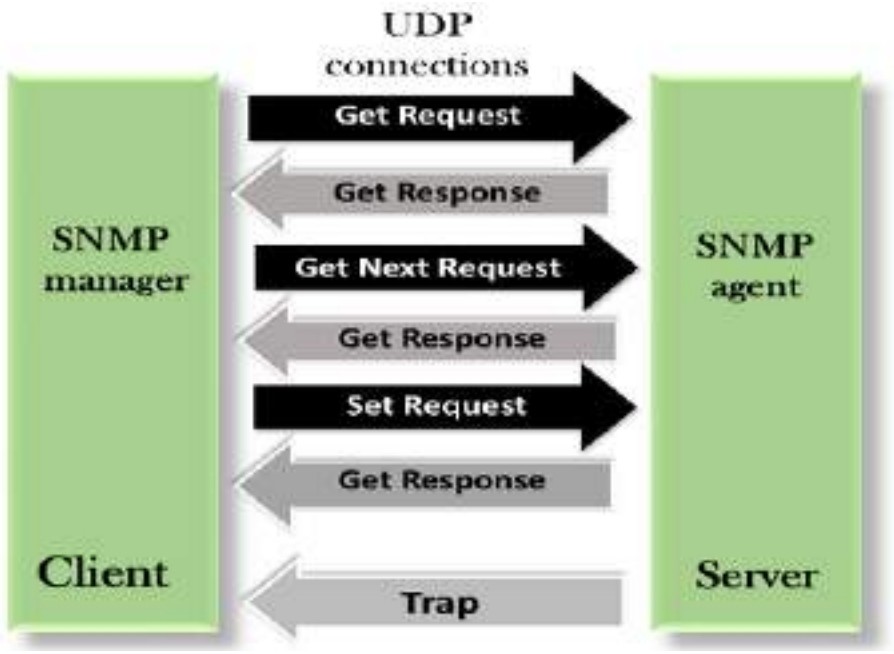
MIB

- The MIB (Management information base) is a second component for the network management.
- Each agent has its own MIB, which is a collection of all the objects that the manager can manage. MIB is categorized into eight groups: system, interface, address translation, ip, icmp, tcp, udp, and egp. These groups are under the mib object.



SNMP

SNMP defines five types of messages: GetRequest, GetNextRequest, SetRequest, GetResponse, and Trap.



GetRequest: The GetRequest message is sent from a manager (client) to the agent (server) to retrieve the value of a variable.

GetNextRequest: The GetNextRequest message is sent from the manager to agent to retrieve the value of a variable. This type of message is used to retrieve the values of the entries in a table. If the manager does not know the indexes of the entries, then it will not be able to retrieve the values. In such situations, GetNextRequest message is used to define an object.

GetResponse: The GetResponse message is sent from an agent to the manager in response to the GetRequest and GetNextRequest message. This message contains the value of a variable requested by the manager.

SetRequest: The SetRequest message is sent from a manager to the agent to set a value in a variable.

Trap: The Trap message is sent from an agent to the manager to report an event. For example, if the agent is rebooted, then it informs the manager as well as sends the time of rebooting.

SOCKET PROGRAMMING

Sockets:

- A **socket** is one endpoint of a **two-way** communication link between two programs running on the network.
- The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication takes place.
- **Sockets** in computer networks are used for allowing the transmission of information between two processes of the same machines or different machines in the network.
- The socket is the combination of IP address and software port number used for communication between multiple processes.

Types of Sockets**:** There are two types: the **datagram** socket and the **stream** socket.

Datagram Socket : This is a type of network that has a connectionless point for sending and receiving packets.

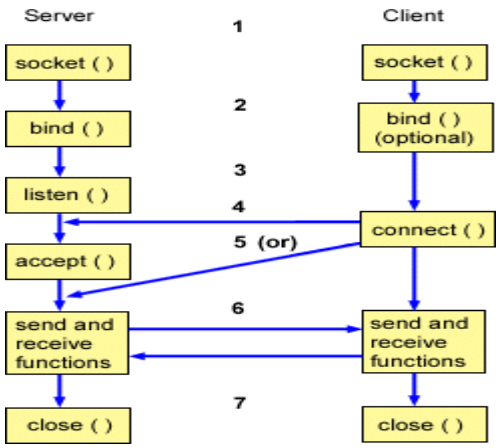
Stream Socket :A stream socket is a type of interprocess communications socket or network socket that provides a connection-oriented, sequenced, and unique flow of data without record boundaries with well-defined mechanisms for creating and destroying connections and for detecting errors.

The socket API is a collection of socket calls that enables you to perform the following primary communication functions between application programs:

- Set up and establish connections to other users on the network
- Send and receive data to and from other users
- Close down connections

How sockets work:

- A socket has a typical flow of events.
- The following figure shows the typical flow of events(and the sequence of issued APIs)for a connection-oriented socket session. An explanation of each event follows the figure.



Function Call	Description
Create()	To create a socket
Bind()	It's a socket identification like a telephone number to contact
Listen()	Ready to receive a connection
Connect()	Ready to act as a sender
Accept()	Confirmation, it is like accepting to receive a call from a sender
Write()	To send data
Read()	To receive data
Close()	To close a connection

This is a typical flow of events for a connection-oriented socket:

1. The socket() API creates an endpoint for communications and returns a socket descriptor that represents the endpoint.
2. When an application has a socket descriptor, it can bind a unique name to the socket. Servers must bind a name to be accessible from the network.
3. The listen() API indicates a willingness to accept client connection requests. When a listen() API is issued for a socket, that socket cannot actively initiate connection requests. The listen() API is issued after a socket is allocated with a socket() API and the bind() API binds a name to the socket. A listen() API must be issued before an accept() API is issued.
4. The client application uses a connect() API on a stream socket to establish a connection to the server.
5. The server application uses the accept() API to accept a client connection request. The server must issue the bind() and listen() APIs successfully before it can issue an accept() API.

6. When a connection is established between stream sockets (between client and server), you can use any of the socket API data transfer APIs. Clients and servers have many data transfer APIs from which to choose, such as send(), recv(), read(), write(), and others.
7. When a server or client wants to stop operations, it issues a close() API to release any system resources acquired by the socket.

Socket characteristics

- A socket is represented by an integer. That integer is called a *socket descriptor*.
- A socket exists as long as the process maintains an open link to the socket.
- You can name a socket and use it to communicate with other sockets in a communication domain.
- Sockets perform the communication when the server accepts connections from them, or when it exchanges messages with them.

Socket Programming in Connection Oriented and Connection Less Protocols:

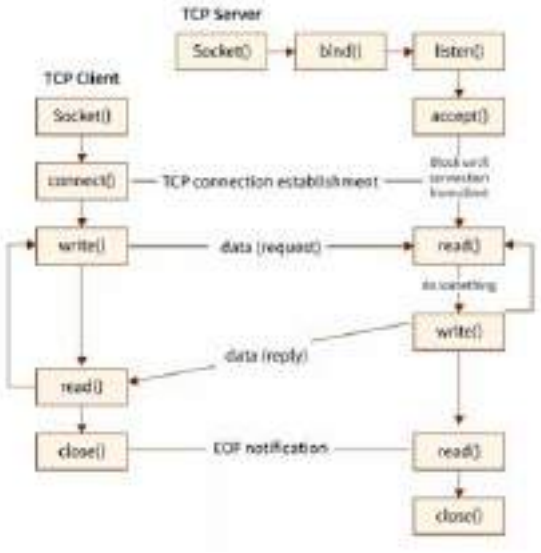
TCP stands for Transmission Control Protocol. TCP is a reliable connection-oriented protocol of the transport layer. TCP establishes the connection before data transmission. Steps for TCP socket programming for establishing TCP socket at the client-side:

- The first step is to create a socket and use the socket() function to create a socket.
- Use the connect() function for connecting the socket to the server address.
- Transmit data between two communicating parties using read() and write() functions.
- After data transmission completion close the connection using close() function..

Following are steps to be followed for establishing a TCP socket on the server-side:

- Use socket() for establishing a socket.
- Use the bind() function for binding the socket to an address.
- Then for listening client connections use listen() function.
- The accept() function is used for accepting the connection of the client.
- Transmit data with the help of the read() and write() function.

Below image to show TCP Socket connection



Socket Programming in UDP

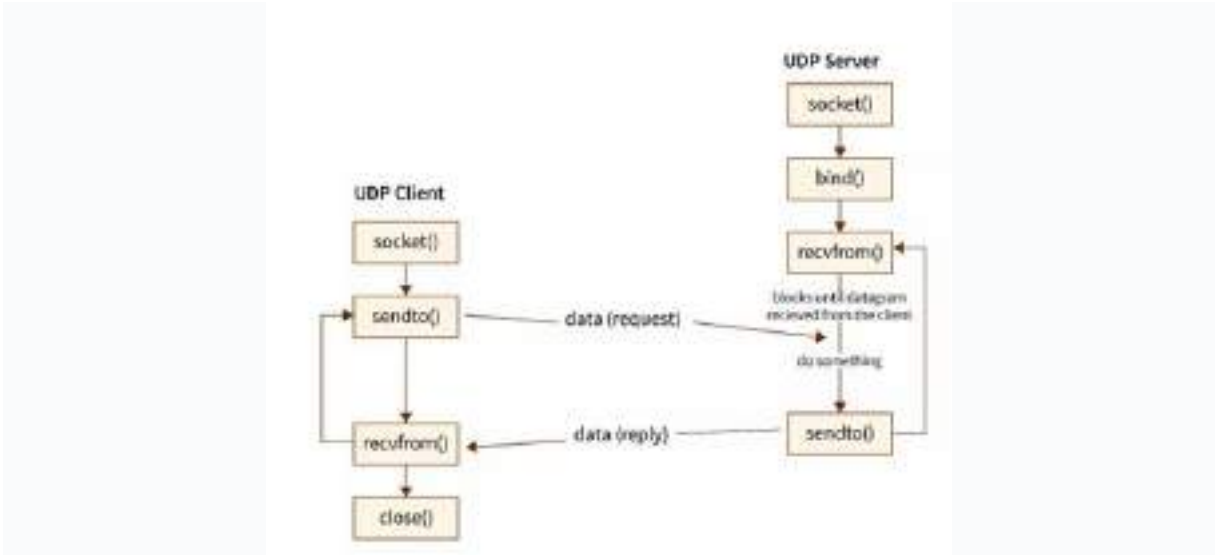
UDP stands for User Datagram Protocol. UDP is a connection-less and unreliable protocol of transport layer. UDP does not establish a connection between two communicating parties before transmitting the data. Following are the steps given that is to be followed for establishing UDP socket connection on the client-side

- Use socket() function for creating socket;
- recvfrom() and sendto() functions are used for transmitting data between two communicating parties.

Steps to be followed for establishing UDP socket connection at the server-side.

- Create a socket using the socket() function.
- Use the bind() function for the binding socket to an address.
- Transmit data with the help of the recvfrom() function and sendto().

Below image to show UDP socket connection



File: MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

File: MyServer**.java**

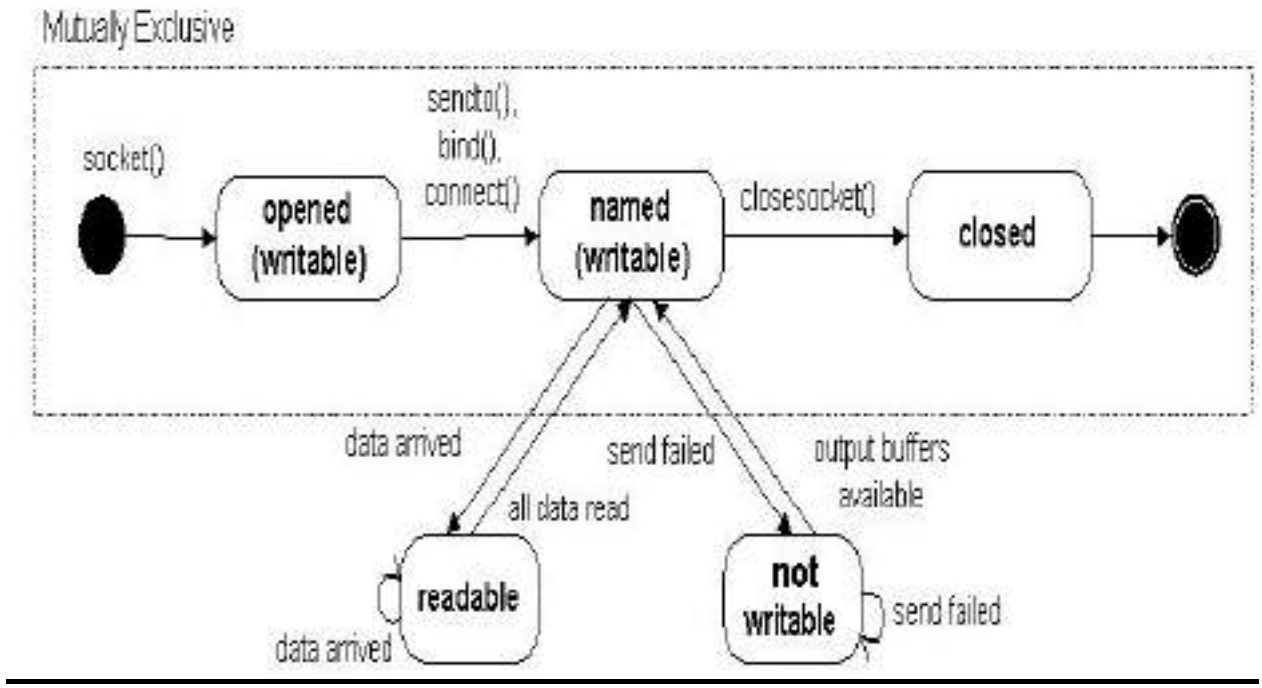
```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
```

```
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e)
{System.out.println(e);}
}
}
```

Socket States:

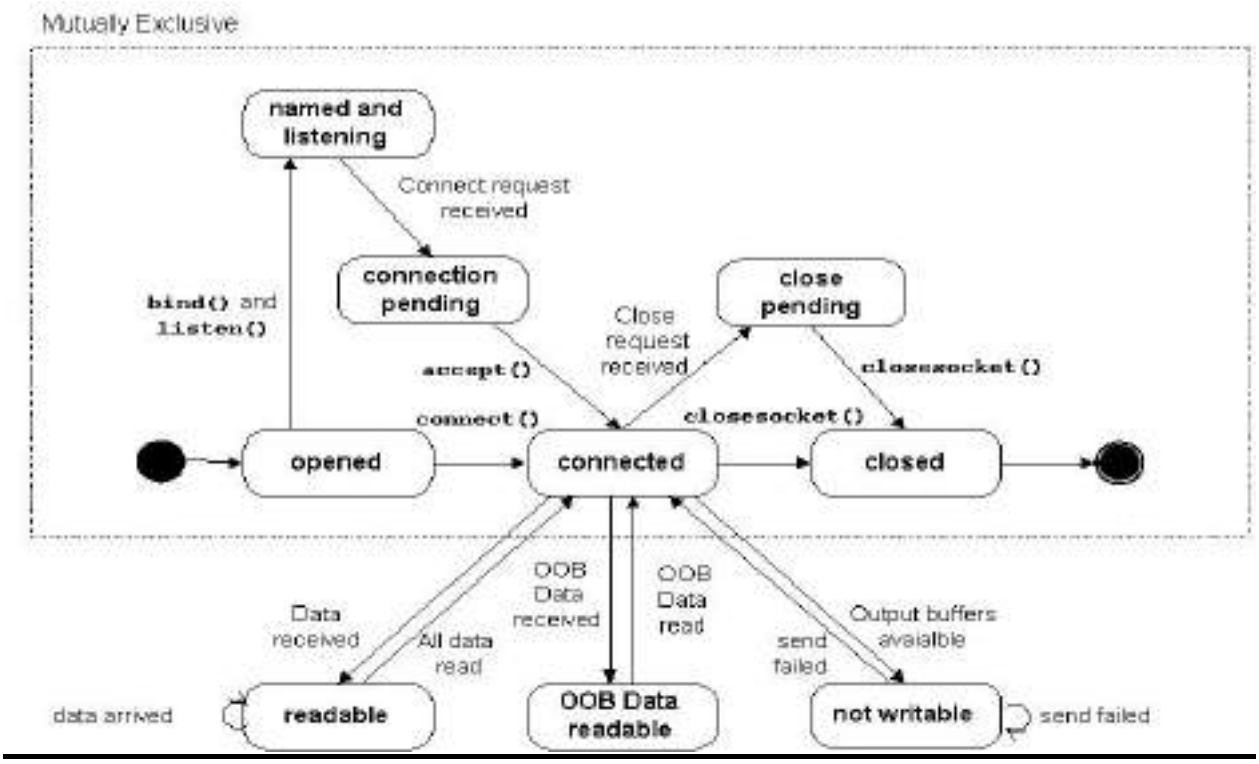
The state of a socket determines which network operations will succeed, which operations will block, and which operations with will fail (the socket state even determines the error code). Sockets have a finite number of states, and the WinSock API clearly defines the conditions that trigger a transition from one state to another.

DATAGRAM SOCKET STATES*:*



Socket State	Meaning
opened	**socket()** returned an unnamed socket (an unnamed socket is one that is not bound to a local address and port). The socket can be named explicitly with bind or implicitly with **sendto()** or **connect()**.
named	A named socket is one that is bound to a local address and a port. The socket can now send and/or receive.
readable	The network system received data and is ready to be read by the application using **recv()** or **recvfrom()**.
not writable	The network system does not have enough buffers to accommodate outgoing data.
closed	The socket handle is invalid.

STREAM SOCKET STATES:



Socket State	Meaning
opened	**socket()** returned an unnamed socket (an unnamed socket is one that is not bound to a local address and port). The socket can be named explicitly with **bind()** or implicitly with **conenct()**.
named and listening	The socket is named (bound to a local address and port) and is ready to accept incoming connection requests.
connection pending	The network system received an incoming connection requests and is waiting for the application to respond.
connected	An association (virtual circuit) has been established between a local and remote host. Sending and receiving data is now possible
readable	The network system received data and is ready to be read by the application using **recv()** or **recvfrom()**.
OOB readable	Out Of Band data received by the network system received data and is ready to be read by the application (using **recv()** or **recvfrom()**.)

Not writable	The network system does not have enough buffers to accommodate outgoing data.
close pending	The virtual circuit is close.
closed	The socket handle is invalid.

Socket Options

Various types of options are available in a socket. There are various ways to get and set the options that affect a socket. They include,

- getsockopt and setsockopt functions
- fcntl function
- ioctl function

(a) getsockopt and setsockopt

Syntax

```
int getsockopt (int sockfd, int level, int optname, void *optval, socklen_t *optlen);

int setsockopt (int sockfd, int level, int optname, const void *optval, socklen_t optlen);

Return Value : Both return 0 on OK, -1 on error.
```

sockfd – refer to an open socket descriptor

level –specifies the code in the system that interprets the option. (i.e) general socket code or protocol specific code (IPv4, IPv6,TCP)

optname – name of the option

optval – It is a pointer to a variable from which the new value of the option is fetched by setsockopt or into which the current value of the option is stored by getsockopt.

Optlen – specifies the size of this variable.

(b) fcntl

fcntl stands for “file control”. This function performs various descriptor control operations.

Syntax

```
int fcntl (int fd, int cmd, ….int arg);

Return value: 0 if OK, -1 on error
```

(c) ioctl

ioctl stands for “IO control”.

Syntax

```
int ioctl (int fd, int request, ….void *arg);

Return value: 0 if OK, -1 on error
```

Generic Socket Options

These options are protocol independent options. These options are as follows.

(1) SO_BROADCAST

- This option enables or disables the ability of the process to send broadcast messages.
- Broadcasting is supported only for datagram sockets and only on networks that support the concept of broadcast message.
- An application must set this socket option before sending any broadcast message.
- If the destination address is a broadcast address and this socket option is not set, EACCESS is returned.

(2) SO_DEBUG

- This option is supported only by TCP.
- When this option is enabled for a TCP socket, the kernel keeps track of the detailed information about all the packets send and received by the TCP for the socket.
- These are kept in a circular buffer and can be examined with the trpt program.

(3) SO_DONTROUTE

- This option specifies that the outgoing packets are to bypass the normal routing mechanisms of the underlying protocol.
- According to the destination address given, the packets will be routed.
- So, a local interface will be identified and then the packet is routed.
- If the local interface cannot be identified, ENETUNREACH is returned.

- This option can also be applied to individual datagrams using MSG_DONTROUTE flag.
- This option is often used by the routing daemons to bypass the routing table and force a packet to be sent out a particular interface.

(4) SO_ERROR

- When an error occurs on a socket, the protocol module sets a variable named so_error for that socket to one of the standard unix Exxx values. This is called the pending error for the socket.
- This option can be fetched but cannot be set.
- The process can be notified about the error in one of the two ways.
    - If the process is blocked in a call to select on the socket, for either readability or writability.
    - If the process is using signal driven I/O, the SIGIO signal is generated for either the process or the process group.

(5) SO_KEEPALIVE

- The purpose of this option is to detect if the peer host crashes or become unreachable.
- When the keepalive option is set for a TCP socket and no data has been exchanged across the socket in either direction for 2 hours, TCP automatically sends a keep-alive probe to the peer.
- This probe is a TCP segment to which the peer must respond. One of the three scenarios result.
    - The peer responds with the expected ACK (If the peer is active)
    - The peer responds with an RST, which tells the local TCP that the peer host has been crashed and rebooted. So, the sockets pending error is set to ECONNRESET and the socket is closed.
    - There is no response from the peer to the keep-alive probe. TCP sends 8 additional probes, 75 seconds apart, trying to get a response from the peer. It will give up if there is no response within 11 minutes and 15 seconds from the time of sending the first probe. If there is no response, the sockets pending error is set to ETIMEDOUT and the socket are closed. If the peer host is unreachable, the pending error is set to EHOSTUNREACH.
- This option is normally used by servers, although clients can also use this option.

- Servers use this option because after establishment of connection, there may be a situation where the server may wait for the client request.
- But, if the client hosts crashes, powered off or connection drops, the server never knows about it and waits for the input that can never arrive. This is called a half open connection. The keep-alive option will detect these half open connections and terminate them.

(6) SO_LINGER

- This option specifies how the close function operates for a connection oriented protocol.
- By default, close returns immediately. But, if there is any data still remaining in the socket send buffer, the system will try to deliver the data to the peer.
- But, the SO_LINGER changes this default case.
- It requires the following structure to be passed between the user process and the kernel.

```
Struct linger
{
        int l_onoff;      /* 0 = off, non-zero = on */
        int l_linger      /* linger time */
}
```

- When this socket option is set, any one of the following three scenarios takes place, depending on the values of the two structure members.
    - If l_onoff=0, the option is turned off. So, the value of l_linger is ignored and the TCP default applies (i.e) close returns immediately.
    - If l_onoff=nonzero and l_linger = 0, TCP aborts the connection when it is closed. (ie) TCP discards any data still remaining in the socket send buffer and sends a RST to its peer.
    - If l_onoff=nonzero and l_linger = nonzero, then the kernel will linger when the socket is closed. (i.e) If there is any data still remaining in the socket send buffer, the process is put to sleep until either,
        - All data is send and acknowledged by the peer TCP.
        - The linger time expires.
- Assume that the client writes data to the socket and then calls close. The following diagrams depict the various scenarios.

(a) Default situation

- By default, close returns immediately.

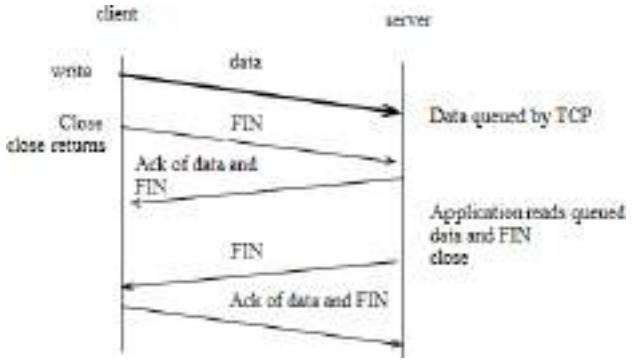


Fig 3.1 Default operation of close

We now need to look at exactly when *close* on a socket returns and what the actions and consequences are. In these cases, we assume that the client writes data to the socket and then calls close.

Fig. 3.1 shows the default scenario. Assume that when the client's data arrives, the server is temporarily busy, so the data is added to the socket receive buffer by its TCP. Similarly, the next segment, the client's FIN is also added.

But by default, the client's close returns immediately. As we see here, the client's close can return before the server reads the remaining data in its socket receive buffer. Therefore it is possible for the server host to crash before the server application reads this remaining data, and the client application will never know.

(b) SO_LINGER socket option is set and l_linger set to a positive value

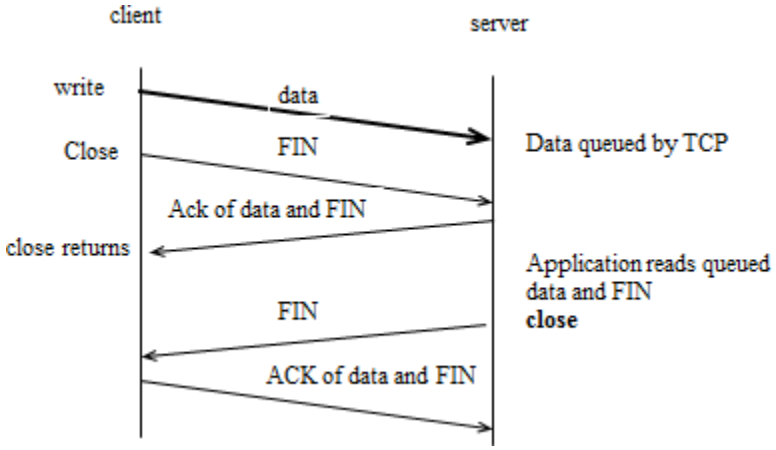


Fig 3.2 l_linger set to a positive value

In this scenario, the client sets the SO_LINGER option, specifying some positive linger time. When this occurs, the client's close does not return until all the client's data and its FIN have been acknowledged by the server TCP as shown in Fig 3.2.

The server host can crash before the server application reads its remaining data, and the client application will never know.

(c) SO_LINGER socket option set with l_linger set to small positive value

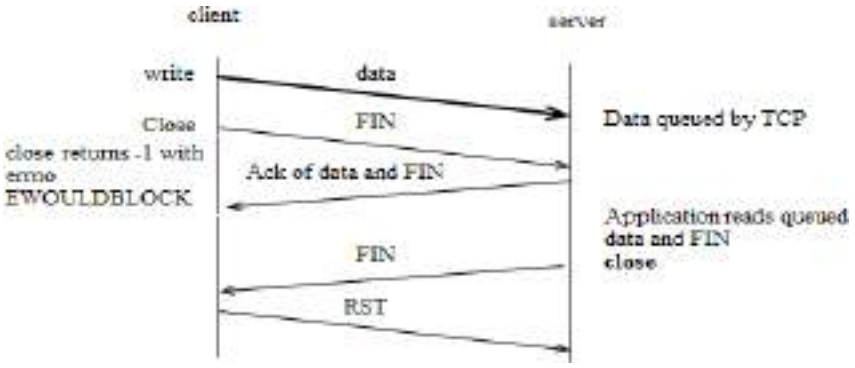


Fig 3.3 l_linger set to small positive value

Fig. 3.3 shows what can happen if the SO_LINGER option is set to a value that is too low. The basic principle here is that a successful return from close, with the SO_LINGER option set, only tells us that the data we sent (and our FIN) have been acknowledge by the peer TCP. It does not tell us whether the peer application has read the data. If we do not set the SO_LINGER option, we do not know whether the peer TCP has acknowledged the data.

(d) Using shutdown to show the peer has received the data

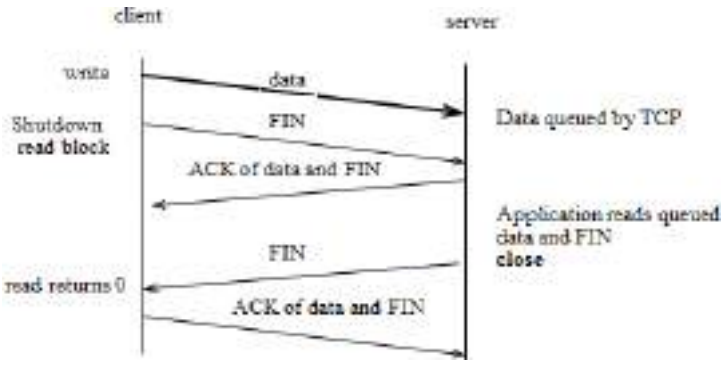


Fig. 3.4 Using shutdown

One way for the client to know that the server has read its data is to call shutdown (with SHUT_WR) instead of close and wait for the peer to close its end of the connection as shown in Fig. 3.4.

(e) Application ACK

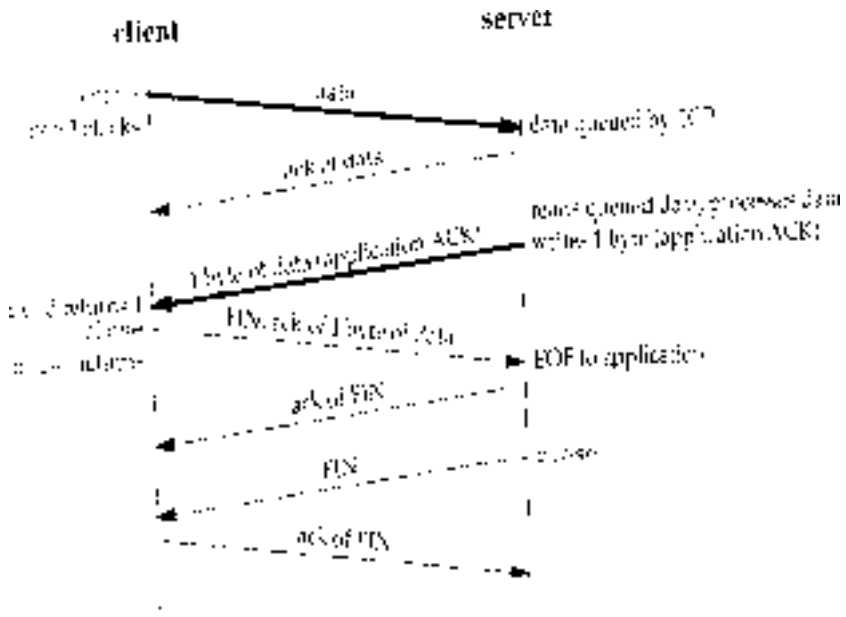


Fig. 3.5 Application ACK

Another way to know that the peer application has read our data is to use an application-level acknowledgment which requires coding in the server and client. In this case, the client waits for a 1 byte acknowledgment for each packet sent. Fig 3.5 shows the possible packet exchange.

(7) SO_OOBINLINE

- When this option is enabled, the out of band data will be placed in the normal input queue.
- When this occurs, the MSG_OOB flag to the receive functions cannot be used to read the out of band data.

(8) SO_RCVBUF and SO_SNDBUF

- Every socket has a send buffer and a receive buffer.
- Receive buffer - It is used to hold the received data until it is read by the application.
- Send buffer – It is used to hold the data to be send.

- The socket receive buffer has a limit in its window size. And, the peer can send data only upto the window size limit. The window size will be advertised to its peer while sending the SYN segment during connection establishment. This is TCPs flow control.
- If the peer ignores the advertised window and if it sends data beyond the window, the receiving TCP discards it.
- The default size of TCP send and receive buffers is 4096 bytes. But, newer systems use larger values from 8192 to 61440 bytes.
- The default size of UDP send buffer is 9000 bytes.
- The default size of UDP receive buffer is 40000 bytes.
- The main goal of this option is that these two options let us change the default sizes.

(9) SO_RCVLOWAT and SO_SNDLOWAT

- Every socket has a receive low water mark and send low water mark.
- These are used by the select function.
- Receive low water mark – It is the amount of data that must be in the socket receive buffer for the select to return readable.
- Send low water mark – It is the amount of available space that must exist in the socket send buffer for select to return writable.
- Default receive low water mark is 1.
- Default send low water mark is 2048.
- These two socket options, let us change these two low water marks.

(10) SO_RCVTIMEO and SO_SNDTIMEO

- These two socket options allow us to place a timeout on socket receive and send.
- This let us specify the timeout in seconds and microseconds.
- The timeout can be disabled by setting its value to 0 seconds and 0 microseconds.
- Both timeouts are disabled by default.
- The receive timeout affects the five input functions namely read, readv, recv, recvfrom and recvmsg.
- The send timeout affects the five output functions namely write, writev, send, sendto and sendmsg.

(11) S0_REUSEADDR and SO_REUSEPORT

- SO_REUSEADDR serves four different purposes.
    - It allows a listening server to start and bind its well known port even if previously established connections exist that use this port as their local port. This condition is typically encountered as follows.
        - A listening server is started.
        - A connection request arrives and a child process is spawned to handle that client.
        - The listening server terminates, but the child continues to service the client on the existing connection.
        - The listening server is restarted.
    - It allows a new server to be started on the same port as an existing server that is bound to the wildcard address as long as each instance binds a different local IP address.
    - It allows a single process to bind the same port to multiple sockets, as long as each bind specifies a different local IP address.
    - It allows completely duplicate bindings : A bind of an IP address and port, when the same IP address and port are already bound to another socket, if the transport protocol supports it.
- This feature is supported only for UDP sockets.
- This feature is used with multicasting to allow the same application to be run multiple times on the same host.
- SO_REUSEADDR does the following.
    - It allows completely duplicate bindings, but only if each socket that wants to bind the same IP address and port specify this socket option.
    - It is considered equivalent to SO_REUSEPORT if the IP address being bound is a multicast address.
- Limitation : It is not supported by all systems.

(12) SO_TYPE

- This option returns the socket type.
- The integer value returned is a value SOCK_STREAM or SOCK_DGRAM or SOCK_RAW.

(13) SO_USELOOPBACK

- This option applies only to sockets in the routing domain.

- By default, this set to ON.
- When this option is enabled, the socket receives a copy of everything sent on the socket.

IPv4 Socket Options

- These socket options are processed by IPv4. These options include the following.

(1) IP_HDRINCL
    - If this option is set for a raw IP socket, we must build our own IP header for all the datagrams we send on the raw socket.
    - Normally kernel builds the IP header for all datagrams, but some applications require to build their own IP header.
    - When this option is set, we build a complete IP header, with the following exceptions.
        - IP always calculates and stores the IP header checksum.
        - If we set the IP identification field to 0, the kernel will set the field.
        - If the source IP address is INADDR_ANY, IP sets it to the primary IP address of the outgoing interface.
        - Setting IP options is implementation dependent.
        - Some fields must be in host byte order and some in network byte order. This is implementation dependent.

(2) IP_OPTIONS
    - Setting this option allows us to set IP options in the IPv4 header.
    - This requires intimate knowledge of the format of IP options in the IP header.

(3) IP_RECVDSTADDR
    - This option causes the destination IP address of a received UDP datagram to be returned as ancillary data by recvmsg.

(4) IP_RECVIF
    - This option causes the index of the interface on which a UDP datagram is received to be returned as ancillary data by recvmsg.

(5) IP_TOS
    - This option let us set the type of service in the IP header for a TCP, UDP socket.
    - TOS can be,
        - T – Throughput
        - R – Reliability

        - D – Delay
        - C - Cost

(6) IP_TTL

- TTL stands for Time to Live
- This option let us set and fetch the default TTL.

3.2 ICMPv6 Socket Option

- This socket option is processed by ICMPv6.

(1) ICMP_FILTER

- This option let us fetch and set an icmp6_filter structure that specifies which of the 256 possible ICMPv6 message types will be passed to the process on a raw socket.

IPv6 Socket Option

- These socket options are processed by IPv6. These options include the following.

(1) IPv6_CHECKSUM

- This option specifies the byte offset into the user data where the checksum field is located.
- If this value is non-negative, the kernel will,
    - Compute and store a checksum for all outgoing packets.
    - Verify the received checksum on input, discarding packets with an invalid checksum.
- If the value is -1 (default), the kernel will not calculate and store the checksum for outgoing packets on this raw socket and will not verify the checksum for received packets.

(2) IPv6_DONTFRAG

- Setting this option disables the automatic insertion of a fragment header for UDP and raw sockets.
- When this option is set, output packets larger than Maximum Transfer Unit (MTU) of the outgoing interface will be dropped.

(3) IPv6_NEXTHOP

- This option specifies the next hop address for a datagram as a socket address structure and is a privileged operation.

(4) IPv6_PATHMTU

- This option cannot be set, only retrieved.
- When this option is retrieved, the current MTU as determined by PATH_MTU discovery is returned.

(5) IPv6_RECVDSTOPTS

- Setting this option specifies that any received IPv6 destination options are to be returned as ancillary data by recvmsg.

(6) IPv6_RECVHOPLIMIT

- Setting this option specifies that the received hop limit field is to be returned as ancillary data by recvmsg.

(7) IPv6_RECVHOPOPTS

- Setting this option specifies that any received IPv6 hop-by-hop options are to be returned as ancillary data by recvmsg.

(8) IPv6_RECVPATHMTU

- Setting this option specifies that the path MTU of a path is to be returned as ancillary data by recvmsg.

(9) IPv6_RECVPKTINFO

- Setting this option specifies that the following two pieces of information about a received IPv6 datagram are to be returned as ancillary data by recvmsg.
  - The destination IPv6 address
  - Arriving interface index

(10) IPv6_RECVRTHDR

- Setting this option specifies that a received IPv6 routing header is to be returned as an ancillary date by recvmsg.

(11) IPv6_RECVTCLASS

- Setting this option specifies that the received traffic class is to be returned as ancillary data by recvmsg.

(12) IPv6_UNICAST_HOPS

- Setting this option specifies the default hop limit for outgoing datagrams sent on the socket, while fetching the socket option returns the value of the hop limit that the kernel will use for the socket.

(13) IPv6_USE_MIN_MTU

- Setting this option avoids fragmentation.
  - When this option is set to 1, path MTU discovery is not performed and packets are sent using minimum MTU.

- When this option is set to 0, causes path MTU discovery to occur for all destinations.
- When this option is set to -1, path MTU discovery is performed.

(14) IPv6_V6ONLY

- Setting this option restricts it to IPv6 communication only.

(15) IPv6_XXX

- UDP socket uses, recvmsg and sendmsg
- TCP socket uses, getsockopt and setsockopt

TCP Socket Options

(1) TCP_MAXSEG

- This socket option allows us to fetch or set the Maximum Segment Size (MSS) for a TCP connection.
- The value returned is the maximum amount of data that the TCP will send to the other end.
- The MSS is set while sending the SYN segment to the peer during connection establishment.
- The maximum amount of data that our TCP will send per segment can also change during the life of the connection if TCP supports path MTU discovery.
- If the route of the peer changes, this value will go up or down.

(2) TCP_NODELAY

- If this option is set, it disables TCP's Nagle algorithm.
- By default, this algorithm is enabled.
- Nagles algorithm avoids the syndrome caused in the sender side (i.e) if the sending side sends data too slowly, by sending each byte as a packet and waiting for the acknowledgment.

Nagle's Algorithm

(1) It sends the first byte as it is as a packet and waits for an acknowledgment.
(2) When it receives the ACK, it does not send the further byte as it is, provided it waits until a certain number of bytes gets accumulated or till the ACK for the previous is arrived.

- The purpose of Nagle's algorithm is to reduce the number of small packets in WAN.
- Small packet is any packet smaller than MSS.
- The two common generators of small packets are the Rlogin and Telnet clients, since they send each keystroke as a separate packet.
- In a fast LAN, we normally donot notice a Nagle's algorithm because the time required for a small packet to be acknowledged is typically a few milliseconds, far less than the time between two successive characters that we type.
- But in a WAN, it takes nearly a second to acknowledge a small packet, so we can notice a delay in the character echoing and this delay is often exaggerated by the Nagle's algorithm.
- Consider the following example,
    - We type the six character string "hello!" with exactly 250 ms between each character.
    - The Round Trip Time (RTT) to the server is 600 ms and the server immediately sends back the echo of each character.
    - Assuming the Nagle's algorithm is disabled, we have the 12 packets as shown below.

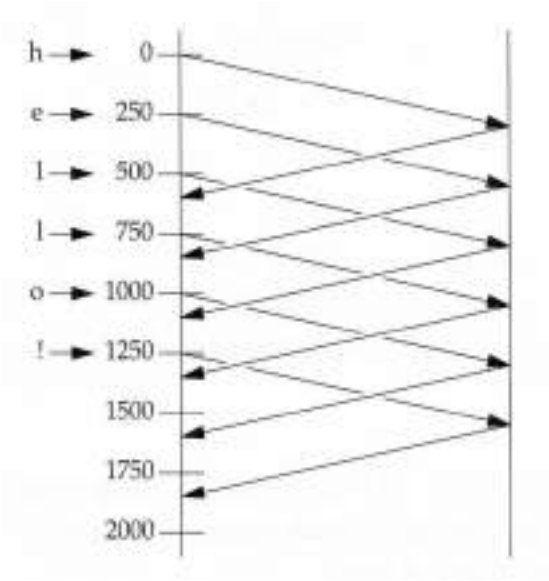


Fig 3.6 Nagle's algorithm (disabled)

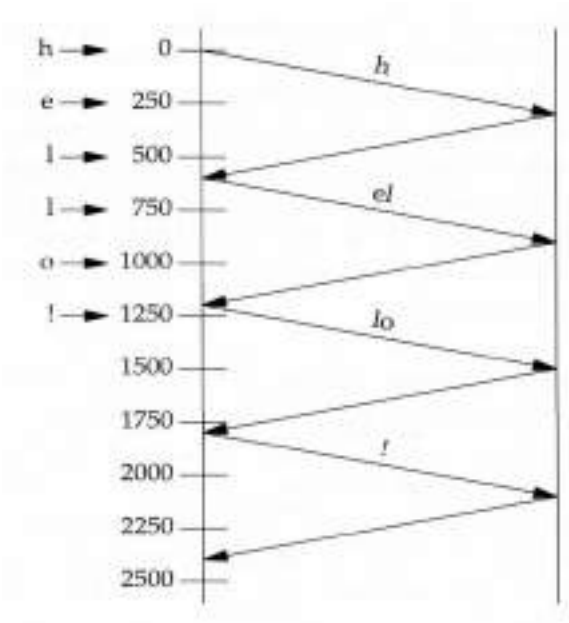


Fig. 3.7 Nagle's Algorithm (Enabled)

- The purpose of the Nagle algorithm is to reduce the number of small packets on a WAN.
- The algorithm states that if a given connection has outstanding data then no small packets will be sent on the connection in response to a user write operation until the existing data is acknowledged.
- Two common generators of small packets are the rlogin and telnet clients, since they normally send each keystroke as a separate packet.
- Fig. 3.6 shows the algorithm disabled. Fig. 3.7 shows the algorithm enabled.
- The characters are typed with 250 milliseconds between each.
- Round Trip Time (RTT) is 600 milliseconds.