GLOBAL INSTITUTE OF TECHNOLOGY

(Approved by AICTE and Affiliated to RTU, Kota)

LABORATORY MANUAL

(2019-2020)

NP LAB

II Year & IV Semester

Computer Science & Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDEX

SYLLABUS

4CS4-23: Network Programming Lab

0L+0T+3P **Credit: 1.5 Max. Marks: 75(IA:45, ETE:30)**

List of Experiments:

1. Study of Different Type of LAN& Network Equipments.
2. Study and Verification of standard Network topologies i.e. Star, Bus, Ring etc.
3. LAN installations and Configurations.
4. Write a program to implement various types of error correcting techniques.
5. Write a program to implement various types of farming methods.
6. Write two programs in C: hello client and hello server
a. The server listens for, and accepts, a single TCP connection; it reads all
The data it can from that connection, and prints it to the screen; then it
Closes the connection
b. The client connects to the server, sends the string “Hello, world!”, then
Close the connection
7. Write an Echo Client and Echo server using TCP to estimate the round trip
Time from client to the server. The server should be such that it can accept
Multiple connections at any given time.
8. Repeat Exercises 6 & 7 for UDP.
9. Repeat Exercise 7 with multiplexed I/O operations.
10. Simulate Bellman-Ford Routing algorithm in NS2.

Global Institute Technology, Jaipur (Approved by AICTE and Affiliated to RTU, Kota)	**NETWORK PROGRAMMING LAB** **4CS4-23** COMPUTER SCIENCE & ENGINEERING	Year : II Sem : IV

0L+0T+3P **Credit: 1.5 Max. Marks: 75(IA:45, ETE:30)**

List of Lab Experiments

Experiment No:-1	Study of different type of NETWORK TOPOLOGIES
Experiment No:-2	Study of LAN,MAN,WAN and connecting devices
Experiment No:-3	Write a programs in C: hello client (The server listens for, and accepts, a single TCP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection)
Experiment No:-4	Write a programs in C: hello server (The client connects to the server, sends the string “Hello, world!”, then closes the connection)
Experiment No:-5	Write a programs in C for TCP and UDP chat server
Experiment No:-6	Write a programs in C: hello client (The server listens for, and accepts, a single UDP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection)
Experiment No:-7	Write a programs in C: hello server (The client connects to the server, sends the string “Hello, world!”, then closes the connection)
Experiment No:-8	Write an Echo server using TCP to estimate the round trip time From client to the server. The server should be such that it can accept multiple connections at any given time , with multiplexed I/O operations
Experiment No:-9	Simulate Bellman-Ford Routing algorithm in NS2 LAN installation and configuration
Experiment No:-10	Simulation of sliding window protocol.
Experiment No:-11	Simulation of File transfer protocol.

Course Objectives &Course Outcomes

Course Objectives: The student should be made:

1. Analyze the different layers in networks.
2. Define, use, and differentiate such concepts as OSI-ISO,TCP/IP.
3. How to send bits from physical layer to data link layer
4. Sending frames from data link layer to Network layer
5. Different algorithms in Network layer
6. Analyze the presentation layer, application layer
7. They can understand how the data transferred from source to destination
8. They can come to know that how the routing algorithms worked out in network layer

Course Outcomes: Upon Completion of the course, the students will be able to

1. To get an idea of how the process executes in UNIX.
2. To know the concept of inter process communication.
3. To implement the network programming in UNIX.
4. To make a client server communication through TCP and UDP protocols.
5. To expose on advanced socket programming, domain name system, http in UNIX environment.

Content beyond Syllabus

Write a programs in C for UDP chat server
Program using fork ( ) system call.
Simulation of sliding window protocol.
Simulation of File transfer protocol.

Introduction to Socket Programming

Aim: Introduction to Socket Programming

Keywords: sockets, client-server, network programming-socket functions, OSI layering, byte-ordering

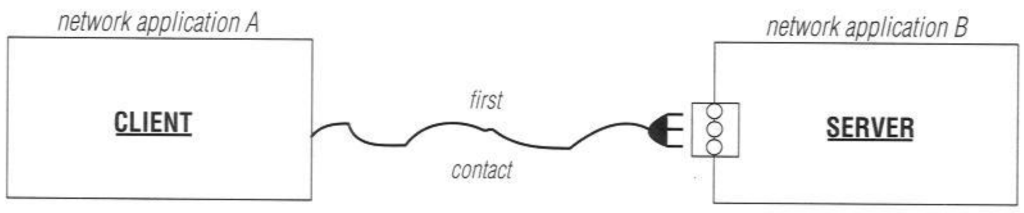
Outline:

1.) Introduction

In this Lab you will be introduced to socket programming at a very elementary level. Specifically, we will focus on TCP socket connections which are a fundamental part of socket programming since they provide a connection oriented service with both flow and congestion control. What this means to the programmer is that a TCP connection provides a reliable connection over which data can be transferred with little effort required on the programmers part; TCP takes care of the reliability, flow control, congestion control for you. First the basic concepts will be discussed, then we will learn how to implement a simple TCP client and server.

2.) The Client / Server Model

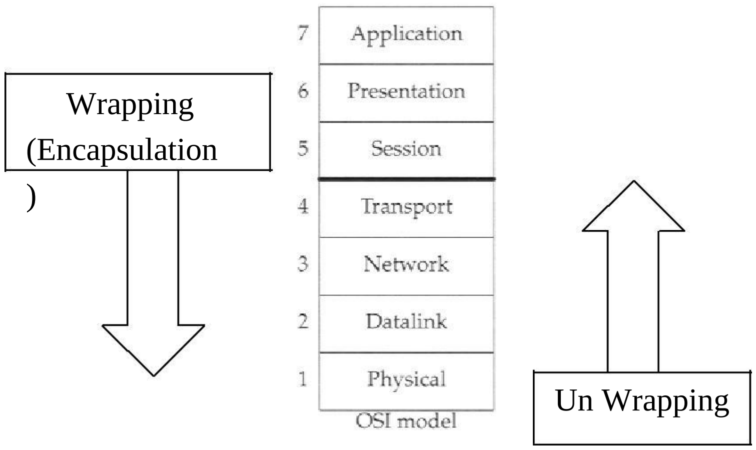
It is possible for two network applications to begin simultaneously, but it is impractical to require it. Therefore, it makes sense to design communicating network applications to perform complementary network operations in sequence, rather than simultaneously. The server executes first and waits to receive; the client executes second and sends the first network packet to the server. After initial contact, either the client or the server is capable of sending and receiving data.



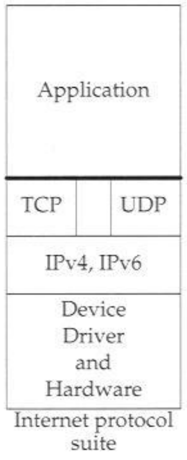
A client initiates communications to a server.

3.) The Socket Interface and Features of a TCP connection

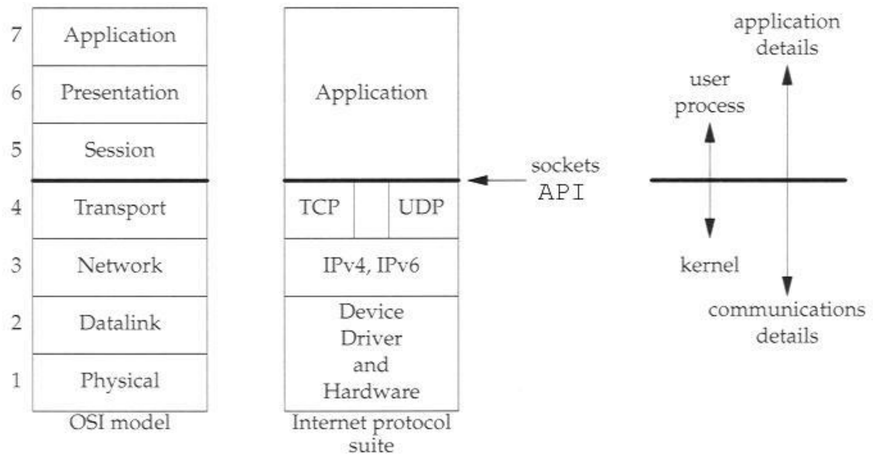
The OSI Layers:



The Internet Layers:



The Internet does not strictly obey the OSI model but rather merges several of the protocols layers together. Where is the socket programming interface in relation to the protocol stack?



<u>Features of a TCP connection:</u>

- Connection Oriented
- Reliability
  1. Handles lost packets
  2. Handles packet sequencing
  3. Handles duplicated packets
- Full Duplex
- Flow Control
- Congestion Control

TCP versus UDP as a Transport Layer Protocol:

TCP	UDP
Reliable, guaranteed	Unreliable. Instead, prompt delivery of Packets.
Connection-oriented	Connectionless
Used in applications that require safety guarantee. (ex. File applications.)	Used in media applications. (ex. video or Voice transmissions.)
Flow control, sequencing of packets, error-control.	No flow or sequence control, user must handle These manually.
Uses byte stream as unit of transfer. (stream sockets)	Uses datagram's as unit of transfer. (datagram sockets)
Allows to send multiple packets with a single ACK.	
Allows two-way data exchange, once the connection is Established. (full-duplex)	Allows data to be transferred in one direction At once. (half-duplex)
e.g. Telnet uses stream sockets. (everything you write on one side appears exact in same order on the other side)	e.g. TFTP (trivial file transfer protocol) uses Datagram sockets.

Sockets versus File I/O

Working with sockets is very similar to working with files. The socket() and accept() functions both return handles (file descriptor) and reads and writes to the sockets requires the use of these handles (file descriptors). In Linux, sockets and file descriptors also share the same file descriptor table. That is, if you open a file and it returns a file descriptor with value say 8, and then immediately open a socket, you will be given a file descriptor with value 9 to reference that socket. Even though sockets and files share the same file descriptor table, they are still very different. Sockets have addresses associated with them whereas files do not, notice that this distinguishes sockets form pipes, since pipes do not have addresses with which they associate. You cannot randomly access a socket like you can a file with lseek(). Sockets must be in the correct state to perform input or output.

File I/O	*Network I/O*
open a file	open a socket
	name the socket
	associate with another socket
read and write	send and receive between sockets
close the file	close the socket

4.) Byte Ordering

Port numbers and IP Addresses (both discussed next) are represented by multi-byte data types which are placed in packets for the purpose of routing and multiplexing. Port numbers are two bytes (16 bits) and IP4 addresses are 4 bytes (32 bits), and a problem arises when transferring multi-byte data types between different architectures. Say Host A uses"*big-endian*" architecture and sends a packet across the network to Host B which uses a "*little-endian*" architecture. If Host B looks at the address to see if the packet is for him/her (choose a gender!), it will interpret the bytes in the opposite order and will wrongly conclude that it is not his/her packet. ***The Internet uses big-endian*** and we call it the *network-byte-order*, and it is really not important to know which method it uses since we have the following functions to convert host-byte-ordered values into network-byte-ordered values and vice versa:

To convert port numbers (16 bits):

Host -> Network
Unit16_t hosts (uint16_t host port number)

Network -> Host
Unit16_t hosts (uint16_t net port number)

To convert IP4 addresses (32 bits):

    Host -> Network
    Unit32_t host (uint32_t host port number)

    Network -> Host
    Unit32_t host (uint32_t net port number)

5.) Address Structures, Ports, and Address conversion functions

<u>Overview of IP4 addresses:</u>

IP4 addresses are 32 bits long. They are expressed commonly in what is known as dotted decimal notation. Each of the four bytes which makes up the 32 address are expressed as an integer value (0 – 255) and separated by a dot. For example, 138.23.44.2 is an example of an IP4 address in dotted decimal notation. There are conversion functions which convert a 32 bit address into a dotted decimal string and vice versa which will be discussed later.

Often times though the IP address is represented by a domain name, for example, hill.ucr.edu. Several functions described later will allow you to convert from one form to another (Magic provided by DNS!).

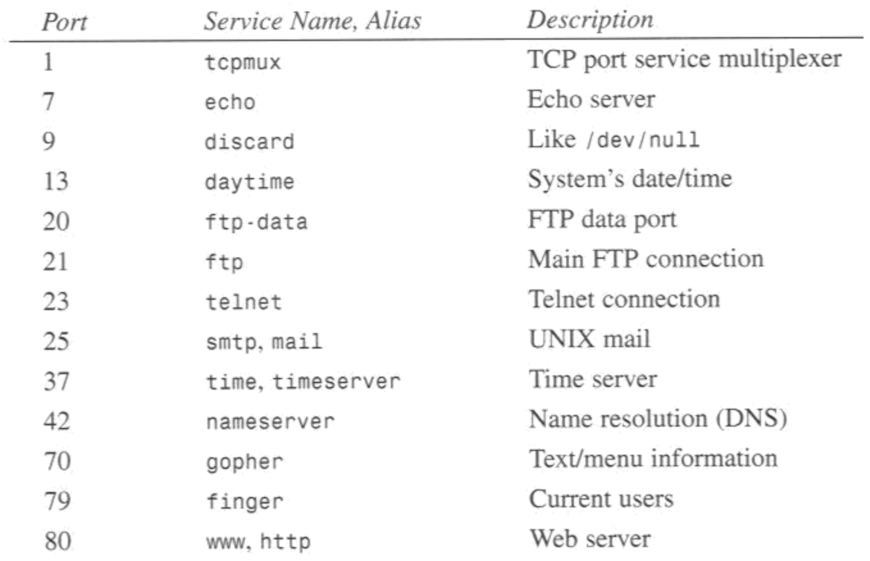
The importance of IP addresses follows from the fact that each host on the Internet has a unique IP address. Thus, although the Internet is made up of many networks of networks with many different types of architectures and transport mediums, it is the IP address which provides a cohesive structure so that at least theoretically, (there are routing issues involved as well), any two hosts on the Internet can communicate with each other.

<u>Ports:</u>

Sockets are UNIQUELY identified by Internet address, end-to-end protocol, and port number.
That is why when a socket is first created it is vital to match it with a valid IP address and a port number.
In our labs we will basically be working with TCP sockets.

Ports are software objects to multiplex data between different applications. When a host receives a packet, it travels up the protocol stack and finally reaches the application layer. Now consider a user running an ftp client, a telnet client, and a web browser concurrently. To which application should the packet be delivered? Well part of the packet contains a value holding a port number, and it is this number which determines to which application the packet should be delivered.

So when a client first tries to contact a server, which port number should the client specify? For many common services, standard port numbers are defined.



Port	Service Name, Alias	Description
1	`tcpmux`	TCP port service multiplexer
7	`echo`	Echo server
9	`discard`	Like `/dev/null`
13	`daytime`	System's date/time
20	`ftp-data`	FTP data port
21	`ftp`	Main FTP connection
23	`telnet`	Telnet connection
25	`smtp, mail`	UNIX mail
37	`time, timeserver`	Time server
42	`nameserver`	Name resolution (DNS)
70	`gopher`	Text/menu information
79	`finger`	Current users
80	`www, http`	Web server

Ports 0 – 1023, are reserved and servers or clients that you create will not be able to **bind** to these ports unless you have root privilege.

Ports 1024 – 65535 are available for use by your programs, but beware other network applications maybe running and using these port numbers as well so do not make assumptions about the availability of specific port numbers. Make sure you read Stevens for more details about the available range of port numbers!

Address Structures:

Socket functions like connect(), accept(), and bind() require the use of specifically defined address structures to hold IP address information, port number, and protocol type. This can be one of the more confusing aspects of socket programming so it is necessary to clearly understand how to use the socket address structures. The difficulty is that you can use sockets to program network applications using different protocols. For example, we can use IP4, IP6, Unix local, etc. Here is the problem: Each different protocol uses a different address structure to hold its addressing information, yet they all use the same functions connect(), accept(), bind() etc. So how do we pass these different structures to a given socket function that requires an address structure? Well it may not be the way you would think it should be done and this is because sockets where developed a long time ago before things like a void pointer where features in C. So this is how it is done:

There is a generic address structure: strut socked

This is the address structure which must be passed to all of the socket functions requiring an address structure. ***This means that you must type cast*** your specific protocol dependent address structure to the generic address structure when passing it to these socket functions.

Protocol specific address structures usually start with *stockade_* and end with a ***suffix*** depending on that protocol. For example:

Struct sockaddr_in	(IP4, think of in as internet)
Struct sockaddr_in6	(IP6)
Struct sockaddr_un	(UNIX local)
Struct sockaddr_dl	(Data link)

We will be only using the IP4 address structure: struct sockaddr_in.
So once we fill in this structure with the IP address, port number, etc we will pass this to one of our socket functions and we will need to type cast it to the generic address structure. For example:

Struct sockaddr_in my Address Struct;

//Fill in the address information into my Address Struct here, (will be explained in detail shortly)

Connect (socket file descriptor, (struct $\underbrace{\text{stockade *) \&my Address}}$ Struct, size of (my
Address Struct));

Here is how to fill in the sockaddr_in structure:

```
struct sockaddr_in{

        sa_family_t  sin family              /*Address/Protocol Family*/ (we'll use PF_INET)
        unit16_t           sin port          /* 16-bit Port number         --Network Byte Ordered--
*/
        struct invade        sin_addr        /*A struct for the 32         bit IP Address  */
        unsigned char sin zero[8]            /*Just ignore this it         is just padding*/
};
```

```
Struct in_addr {
        unit32_t         s_addr   /*32 bit IP Address   --Network Byte Ordered-- */
};
```

For the Subfamily variable sin family always use the constant: PF_INET or AF_INET
***Always initialize address structures with zero () or memo set () before filling them in ***
***Make sure you use the byte ordering functions when ne Cesar for the port and IP
address variables otherwise there will be strange things a happening to your packets***

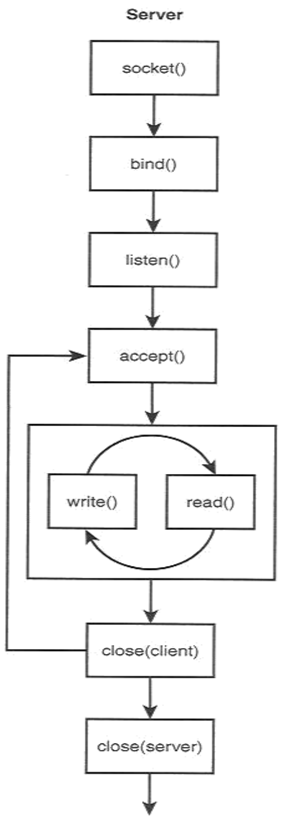
To convert a string dotted decimal IP4 address to a NETWORK BYTE ORDERED 32 bit value use the functions:

- inet_addr()
- initiation()

To convert a 32 bit NETWORK BYTE ORDERED to a IP4 dotted decimal string use:

- inet_ntoa()

6.) Outline of a TCP Server:



Step 1: Creating a socket:

Int socket (int family, int type, int protocol);

Creating a socket is in some ways similar to opening a file. This function creates a file descriptor and returns it from the function call. You later use this file descriptor for reading, writing and using with other socket functions

Parameters:
Family: AF_INET or PF_INET (These are the IP4 family)
Type: SOCK_STREAM (for TCP) or SOCK_DGRAM (for UDP)
Protocol: IPPROTO_TCP (for TCP) or IPPROTO_UDP (for UDP) or use 0

Step 2: Binding an address and port number

int bind(int socket_file_descriptor, const struct stockade * Local Address, socklen_t Address Length);

We need to associate an IP address and port number to our application. A client that wants to connect to our server needs both of these details in order to connect to our server. Notice the difference between this function and the connect () function of the client. The connect function specifies a remote address that the client wants to connect to, while here, the server is specifying to the bind function a local IP address of one of its Network Interfaces and a local port number.

The parameter *socket_file_descriptor* is the socket file descriptor returned by a call to *socket()* function. The return value of *bind ()* is 0 for success and –1 for failure.

**Again make sure that you cast the structure as a generic address structure in this function **

You also do not need to find information about the IP addresses associated with the host you are working on. You can specify: INNADDR_ANY to the address structure and the bind function will use on of the available (there may be more than one) IP addresses. This ensures that connections to a specified port will be directed to this socket, regardless of which Internet address they are sent to. This is useful if host has multiple IP addresses, then it enables the user to specify which IP address will be b_nded to which port number.

<u>Step 3:</u> Listen for incoming connections

Binding is like waiting by a specific phone in your house, and Listening is waiting for it to ring.

int listen(int socket_file_descriptor, int backlog);

The backlog parameter can be read in Stevens' book. It is important in determining how many connections the server will connect with. Typical values for backlog are 5 – 10.

The parameter *socket_file_descriptor* is the socket file descriptor returned by a call to *socket()* function. The return value of *listen ()* is 0 for success and –1 for failure.

<u>Step 4:</u> Accepting a connection.

int accept (int socket_file_descriptor, struct stockade * Client Address, socklen_t *add Len);

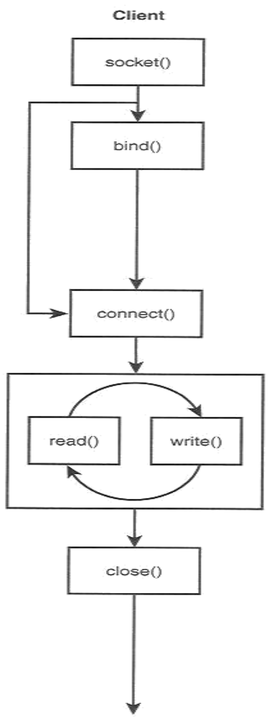
Accept () returns a new socket file descriptor for the purpose of reading and writing to the client. The original file descriptor is used usually used for listening for new incoming connections. Servers will be discussed in much more detail in a later lab.

It desuetude the next connection request on the queue for this socket of the server. If queue is empty, this function blocks until a connection request arrives. (Read the reference book *TCP/IP Implementation in C* for more details.)

Again, make sure you type cast to the generic socket address structure

Note that the last parameter is a pointer. You are not specifying the length, the kernel is and returning the value to your application, the same with the Client Address. After a connection with a client is established the address of the client must be made available to your server, otherwise how could you communicate back with the client? Therefore, the accept () function call fills in the address structure and length of the address structure for your use. Then accept () returns a new file descriptor, and it is this file descriptor with which you will read and write to the client.

7.) Outline of a TCP Client



Step 1: Create a socket: Same as in the server.

Step 2: Binding a socket: This is unnecessary for a client, what bind does is (and will be discussed in detail in the server section) is associate a port number to the application. If you skip this step with a TCP client, a temporary port number is automatically assigned, so it is just better to skip this step with the client.

Step 3: Connecting to a Server:

  int connect(int socket_file_descriptor, const struct stockade *Server Address, socklen_t Address Length);

Once you have created a socket and have filled in the address structure of the server you want to connect to, the next thing to do is to connect to that server. This is done with the connect function listed above.

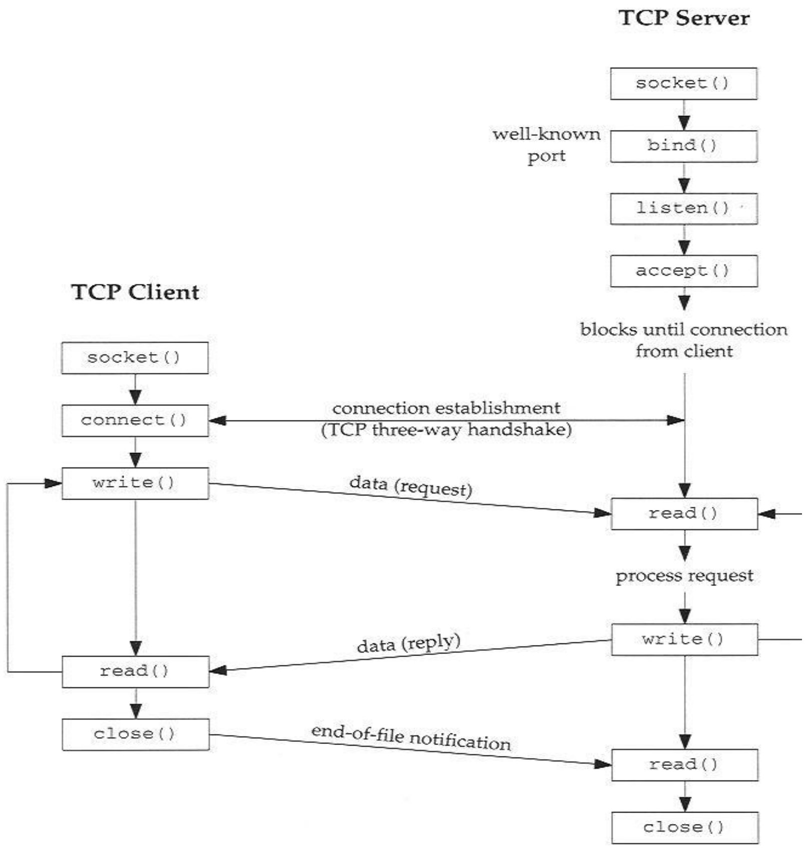
**This is one of the socket functions which requires an address structure so remember to type cast it to the generic socket structure when passing it to the second argument **

Connect performs the three-way handshake with the server and returns when the connection is established or an error occurs.

Once the connection is established you can begin reading and writing to the socket.

Step 4: Read and Writing to the socket will be discussed shortly
Step 5: Closing the socket will be discussed shortly

8.) Outline of a client-server network interaction:



Communication of 2 pairs via sockets necessitates existence of this 4-tuple:

- Local IP address
- Local Port#
- Foreign IP address
- Foreign Port#

!!!! When a server receives (accepts) the client's connection request => it *fork*s a copy of itself and lets the child handle the client. (make sure you remember these Operating Systems concepts)

Therefore on the server machine, listening socket is distinct from the connected socket.

Read/write: These are the same functions you use with files but you can use them with sockets as well. However, it is extremely important you understand how they work so please read Stevens carefully to get a full understanding.

Writing to a socket:

int write(int file descriptor, const void * buff, size message length);

The return value is the number of bytes written, and –1 for failure. The number of bytes written may be less than the message length. What this function does is transfer the data from you application to a buffer in the kernel on your machine, it does not directly transmit the data over the network. This is extremely important to understand otherwise you will end up with many headaches trying to debug your programs.

TCP is in complete control of sending the data and this is implemented inside the kernel. Due to network congestion or errors, TCP may not decide to send your data right away, even when the function call returns. TCP has an elaborate sliding window mechanism which you will learn about in class to control the rate at which data is sent. Read pages 48-49, 77-78 in Stevens very carefully.

Reading from a socket:

int read(int file descriptor, char *buffer, size buffer length);

The value returned is the number of bytes read which may not be buffer length! It returns –1 for failure. As with write(), read() only transfers data from a buffer in the kernel to your application , you are not directly reading the byte stream from the remote host, but rather TCP is in control and buffers the data for your application.

Shutting down sockets:

After you are finished reading and writing to your socket you most call the close system call on the socket file descriptor just as you do on a normal file descriptor otherwise you waste system resources.

The close () function: int close (int file descriptor);

The shut down () function: You can also shutdown a socket in a partial way which is often used when forking off processes. You can shutdown the socket so that it won't send anymore or you could also shutdown the socket so that it won't read anymore as well. This function is not so important now but will be discussed in detail later. You can look at the man pages for a full description of this function.

12.) Summary of Functions

For specific and up-to-date information about each of the following functions, please use the online man pages and Steven’s Unix Network Programming Vol. I.

Socket creation and destruction:

- socket()
- close()
- shutdown()

Client:

- connect()
- bind()

Server:

- accept()
- bind()
- listen()

Data Transfer:

- send()
- recv()
- write()
- read()

Miscellaneous:

- zeros()
- mindset()

Host Information:

- unnamed()
- get host by name()
- get host by addr()

Address Conversion:

- initiation()
- inet_addr()
- inet_ntoa()

Experiment No.: 1

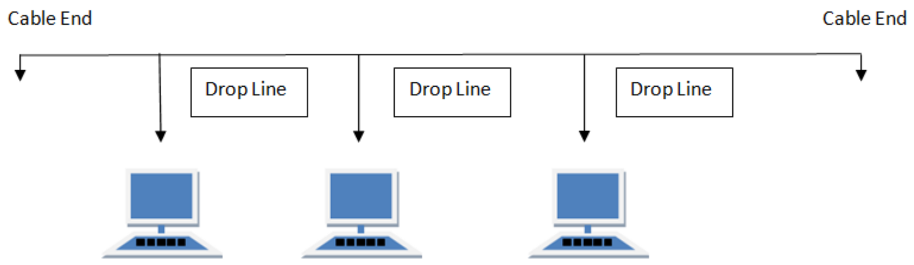
Aim. Study of different types of network topologies.

Types of Network Topology

Network Topology is the schematic description of a network arrangement, connecting various nodes (sender and receiver) through lines of connection.

BUS Topology

Bus topology is a network type in which every computer and network device is connected to single cable. When it has exactly two endpoints, then it is called **Linear Bus topology**.



Features of Bus Topology

1. It transmits data only in one direction.
2. Every device is connected to a single cable

Advantages of Bus Topology

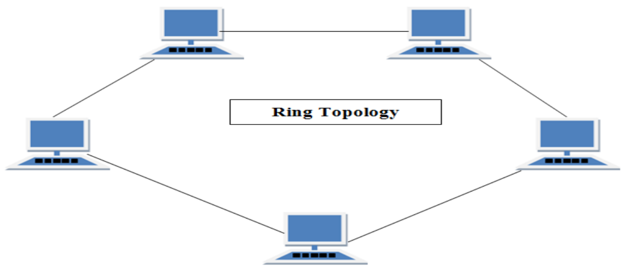
1. It is cost effective.
2. Cable required is least compared to other network topology.
3. Used in small networks.
4. It is easy to understand.
5. Easy to expand joining two cables together.

Disadvantages of Bus Topology

1. Cables fails then whole network fails.
2. If network traffic is heavy or nodes are more the performance of the network decreases.
3. Cable has a limited length.
4. It is slower than the ring topology.

RING Topology

It is called ring topology because it forms a ring as each computer is connected to another computer, with the last one connected to the first. Exactly two neighbors for each device.



Features of Ring Topology

1. A number of repeaters are used for Ring topology with large number of nodes, because if someone wants to send some data to the last node in the ring topology with 100 nodes, then the data will have to pass through 99 nodes to reach the 100th node. Hence to prevent data loss repeaters are used in the network.
2. The transmission is unidirectional, but it can be made bidirectional by having 2 connections between each Network Node, it is called **Dual Ring Topology**.
3. In Dual Ring Topology, two ring networks are formed, and data flow is in opposite direction in them. Also, if one ring fails, the second ring can act as a backup, to keep the network up.
4. Data is transferred in a sequential manner that is bit by bit. Data transmitted, has to pass through each node of the network, till the destination node.

Advantages of Ring Topology

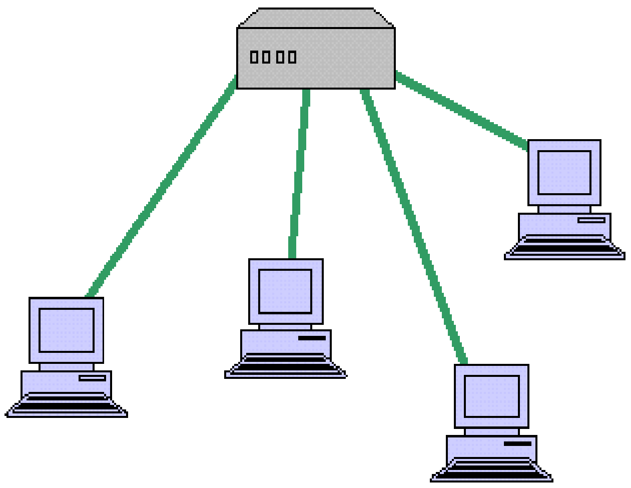
1. Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
2. Cheap to install and expand

Disadvantages of Ring Topology

1. Troubleshooting is difficult in ring topology.
2. Adding or deleting the computers disturbs the network activity.
3. Failure of one computer disturbs the whole network.

Star Topology

- All computers/devices connect to a central device called hub or switch.
- Each device requires a single cable
- point-to-point connection between the device and hub.
- Most widely implemented
- Hub is the single point of failure



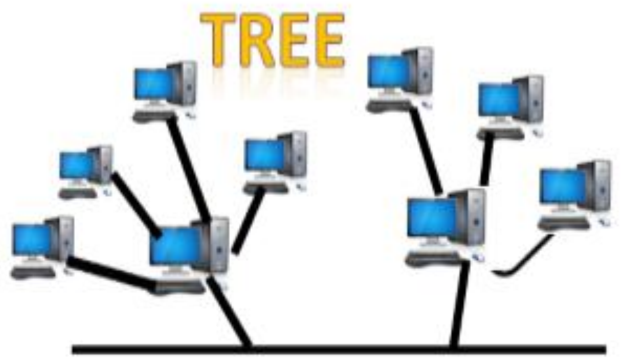
Advantages of Star topology

- **Efficient troubleshooting:** Troubleshooting is quite efficient in a star topology as compared to bus topology. In a bus topology, the manager has to inspect the kilometers of cable. In a star topology, all the stations are connected to the centralized network. Therefore, the network administrator has to go to the single station to troubleshoot the problem.
- **Network control:** Complex network control features can be easily implemented in the star topology. Any changes made in the star topology are automatically accommodated.
- **Limited failure:** As each station is connected to the central hub with its own cable, therefore failure in one cable will not affect the entire network.
- **Familiar technology:** Star topology is a familiar technology as its tools are cost-effective.
- **Easily expandable:** It is easily expandable as new stations can be added to the open ports on the hub.
- **Cost effective:** Star topology networks are cost-effective as it uses inexpensive coaxial cable.
- **High data speeds:** It supports a bandwidth of approx 100Mbps. Ethernet 100BaseT is one of the most popular Star topology networks.

Disadvantages of Star topology

- **A Central point of failure:** If the central hub or switch goes down, then all the connected nodes will not be able to communicate with each other.
- **Cable:** Sometimes cable routing becomes difficult when a significant amount of routing is required.

Tree topology



- Tree topology combines the characteristics of bus topology and star topology.
- A tree topology is a type of structure in which all the computers are connected with each other in hierarchical fashion.
- The top-most node in tree topology is known as a root node, and all other nodes are the descendants of the root node.
- There is only one path exists between two nodes for the data transmission. Thus, it forms a parent-child hierarchy.

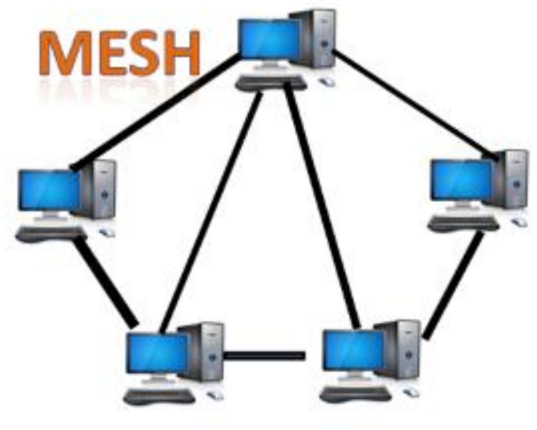
Advantages of Tree topology

- **Support for broadband transmission:** Tree topology is mainly used to provide broadband transmission, i.e., signals are sent over long distances without being attenuated.
- **Easily expandable:** We can add the new device to the existing network. Therefore, we can say that tree topology is easily expandable.
- **Easily manageable:** In tree topology, the whole network is divided into segments known as star networks which can be easily managed and maintained.
- **Error detection:** Error detection and error correction are very easy in a tree topology.
- **Limited failure:** The breakdown in one station does not affect the entire network.
- **Point-to-point wiring:** It has point-to-point wiring for individual segments.

Disadvantages of Tree topology

- **Difficult troubleshooting:** If any fault occurs in the node, then it becomes difficult to troubleshoot the problem.
- **High cost:** Devices required for broadband transmission are very costly.
- **Failure:** A tree topology mainly relies on main bus cable and failure in main bus cable will damage the overall network.
- **Reconfiguration difficult:** If new devices are added, then it becomes difficult to reconfigure.

Mesh topology

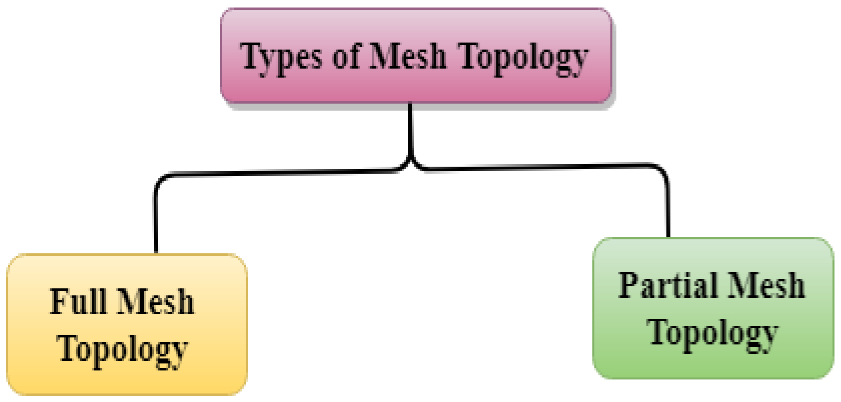


- Mesh technology is an arrangement of the network in which computers are interconnected with each other through various redundant connections.
- There are multiple paths from one computer to another computer.
- It does not contain the switch, hub or any central computer which acts as a central point of communication.
- The Internet is an example of the mesh topology.
- Mesh topology is mainly used for WAN implementations where communication failures are a critical concern.
- Mesh topology is mainly used for wireless networks.
- Mesh topology can be formed by using the formula:
  Number of cables = (n*(n-1))/2;

Where n is the number of nodes that represents the network.

Mesh topology is divided into two categories:

- Fully connected mesh topology
- Partially connected mesh topology



- **Full Mesh Topology:** In a full mesh topology, each computer is connected to all the computers available in the network.
- **Partial Mesh Topology:** In a partial mesh topology, not all but certain computers are connected to those computers with which they communicate frequently.

Advantages of Mesh topology:

Reliable: The mesh topology networks are very reliable as if any link breakdown will not affect the communication between connected computers.

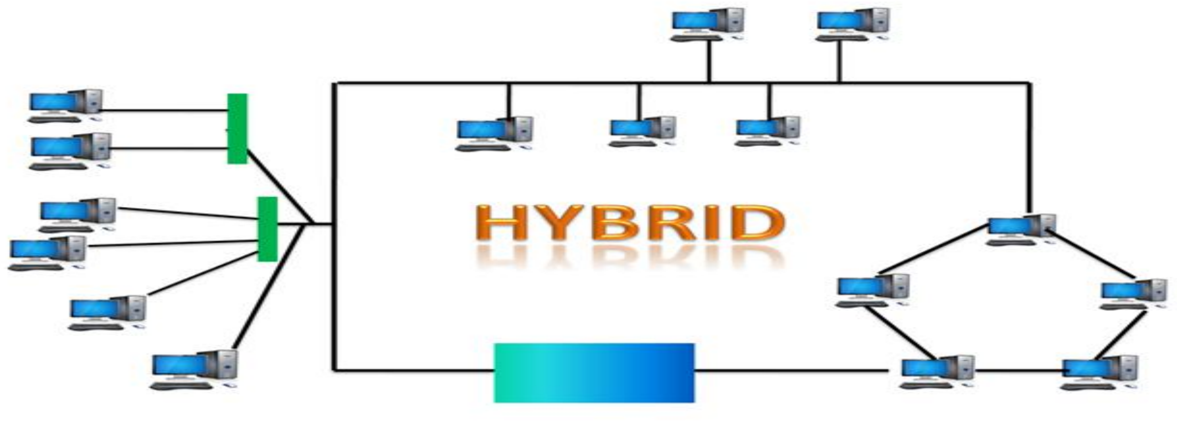
Fast Communication: Communication is very fast between the nodes.

Easier Reconfiguration: Adding new devices would not disrupt the communication between other devices.

Disadvantages of Mesh topology

- **Cost:** A mesh topology contains a large number of connected devices such as a router and more transmission media than other topologies.
- **Management:** Mesh topology networks are very large and very difficult to maintain and manage. If the network is not monitored carefully, then the communication link failure goes undetected.
- **Efficiency:** In this topology, redundant connections are high that reduces the efficiency of the network.

Hybrid Topology



- The combination of various different topologies is known as **Hybrid topology**.
- A Hybrid topology is a connection between different links and nodes to transfer the data.
- When two or more different topologies are combined together is termed as Hybrid topology and if similar topologies are connected with each other will not result in Hybrid topology. For example, if there exist a ring topology in one branch of ICICI bank and bus topology in another branch of ICICI bank, connecting these two topologies will result in Hybrid topology.

Advantages of Hybrid Topology

- **Reliable:** If a fault occurs in any part of the network will not affect the functioning of the rest of the network.
- **Scalable:** Size of the network can be easily expanded by adding new devices without affecting the functionality of the existing network.
- **Flexible:** This topology is very flexible as it can be designed according to the requirements of the organization.
- **Effective:** Hybrid topology is very effective as it can be designed in such a way that the strength of the network is maximized and weakness of the network is minimized.

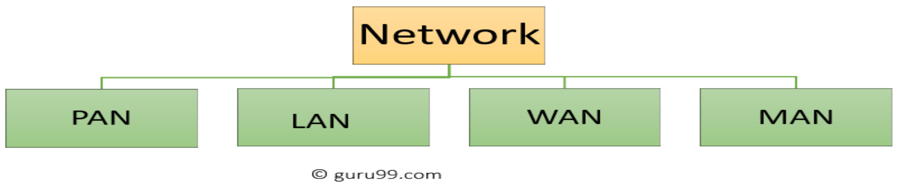
Disadvantages of Hybrid topology

- **Complex design:** The major drawback of the Hybrid topology is the design of the Hybrid network. It is very difficult to design the architecture of the Hybrid network.
- **Costly Hub:** The Hubs used in the Hybrid topology are very expensive as these hubs are different from usual Hubs used in other topologies.
- **Costly infrastructure:** The infrastructure cost is very high as a hybrid network requires a lot of cabling, network devices, etc.

Experiment No.: 2

Aim: Study of LAN, MAN, WAN and connecting devices

There are various types of computer networks available. We can categorize them according to their size as well as their purpose. The size of a network should be expressed by the geographic area and number of computers, which are a part of their networks. It includes devices housed in a single room to millions of devices spread across the world.



Some of the most popular network types are:

- PAN
- LAN
- MAN
- WAN

Let's study all of these networks in detail.

PAN (Personal Area Network)?

PAN is a computer network formed around a person. It generally consists of a computer, mobile, or personal digital assistant. PAN can be used for establishing communication among these personal devices for connecting to a digital network and the internet

Characteristics of PAN

- It is mostly personal devices network equipped within a limited area.
- Allows you to handle the interconnection of IT devices at the surrounding of a single user.
- PAN includes mobile devices, tablet, and laptop.
- It can be wirelessly connected to the internet called WPAN.
- Appliances use for PAN: cordless mice, keyboards, and Bluetooth systems

Advantages of PAN

Here, are important pros/benefits of using PAN network:

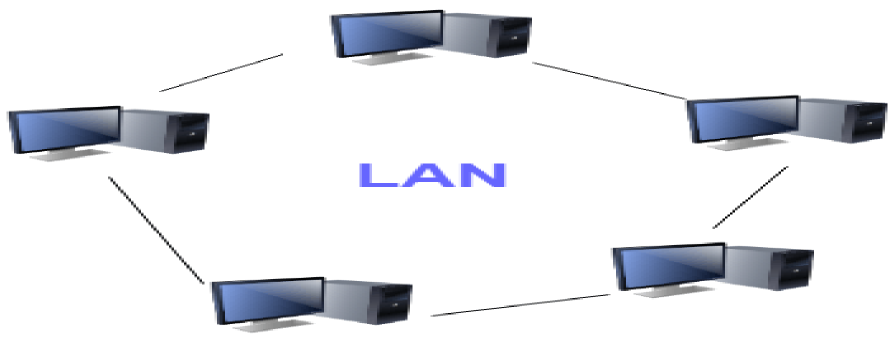
- PAN networks are relatively secure and safe
- It offers only short-range solution up to ten meters
- Strictly restricted to a small area

Disadvantages of PAN

Here are important cons/ drawback of using PAN network:

- It may establish a bad connection to other networks at the same radio bands.
- Distance limits.

LAN (LOCAL AREA NETWORK)



A **L**ocal **A**rea **N**etwork (LAN) is a group of computer and peripheral devices which are connected in a limited area such as school, laboratory, home, and office building. It is a widely useful network for sharing resources like files, printers, games, and other application. The simplest type of LAN network is to connect computers and a printer in someone's home or office. In general, LAN will be used as one type of transmission medium.

It is a network which consists of less than 5000 interconnected devices across several buildings.

Characteristics of LAN

Here are important characteristics of a LAN network:

- It is a private network, so an outside regulatory body never controls it.
- LAN operates at a relatively higher speed compared to other WAN systems.
- There are various kinds of media access control methods like token ring and Ethernet.

Advantages of LAN

Here are pros/benefits of using LAN:

- Computer resources like hard-disks, DVD-ROM, and printers can share local area networks. This significantly reduces the cost of hardware purchases.
- You can use the same software over the network instead of purchasing the licensed software for each client in the network.
- Data of all network users can be stored on a single hard disk of the server computer.
- You can easily transfer data and messages over networked computers.

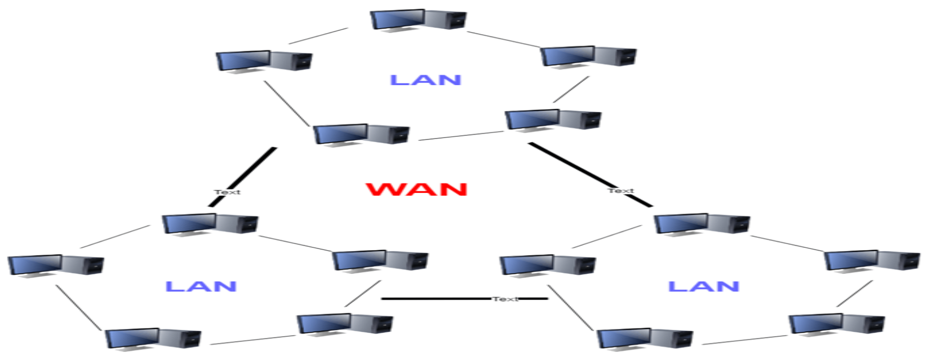
- It will be easy to manage data at only one place, which makes data more secure.
- Local Area Network offers the facility to share a single internet connection among all the LAN users.

Disadvantages of LAN

Here are the important cons/ drawbacks of LAN:

- LAN will indeed save cost because of shared computer resources, but the initial cost of installing Local Area Networks is quite high.
- The LAN admin can check personal data files of every LAN user, so it does not offer good privacy.
- Unauthorized users can access critical data of an organization in case LAN admin is not able to secure centralized data repository.
- Local Area Network requires a constant LAN administration as there are issues related to software setup and hardware failures

WAN (WIDE AREA NETWORK)



WAN (Wide Area Network) is another important computer network that which is spread across a large geographical area. WAN network system could be a connection of a LAN which connects with other LAN's using telephone lines and radio waves. It is mostly limited to an enterprise or an organization.

Characteristics of LAN:

- The software files will be shared among all the users; therefore, all can access to the latest files.
- Any organization can form its global integrated network using WAN.

Advantages of WAN

Here are the benefits/ pros of using WAN:

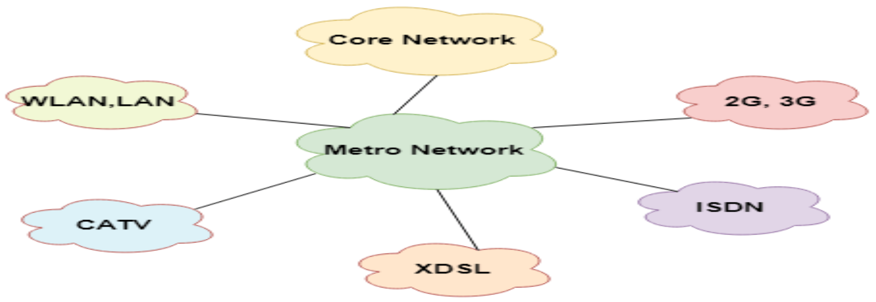
- WAN helps you to cover a larger geographical area. Therefore business offices situated at longer distances can easily communicate.
- Contains devices like mobile phones, laptop, tablet, computers, gaming consoles, etc.
- WLAN connections work using radio transmitters and receivers built into client devices.

Disadvantage of WAN

Here are drawbacks/cons of using WAN:

- The initial setup cost of investment is very high.
- It is difficult to maintain the WAN network. You need skilled technicians and network administrators.
- There are more errors and issues because of the wide coverage and the use of different technologies.
- It requires more time to resolve issues because of the involvement of multiple wired and wireless technologies.
- Offers lower security compared to other types of networks.

MAN (METROPOLITAN AREA NETWORK)



A Metropolitan Area Network or MAN is consisting of a computer network across an entire city, college campus, or a small region. This type of network is large than a LAN, which is mostly limited to a single building or site. Depending upon the type of configuration, this type of network allows you to cover an area from several miles to tens of miles.

Characteristics of MAN

Here are important characteristics of the MAN network:

- It mostly covers towns and cities in a maximum 50 km range
- Mostly used medium is optical fibers, cables
- Data rates adequate for distributed computing applications.

Advantages of MAN

Here are pros/benefits of using MAN system:

- It offers fast communication using high-speed carriers, like fiber optic cables.
- It provides excellent support for an extensive size network and greater access to WANs.
- The dual bus in MAN network provides support to transmit data in both directions concurrently.
- A MAN network mostly includes some areas of a city or an entire city.

Disadvantages of MAN

Here are drawbacks/ cons of using the MAN network:

- You need more cable to establish MAN connection from one place to another.
- In MAN network it is tough to make the system secure from hackers

Other Types of Networks

Apart from above mentioned here, are some other important types of networks:

- WLAN (Wireless Local Area Network)
- Storage Area Network
- System Area Network
- Home Area Network
- POLAN- Passive Optical LAN
- Enterprise private network
- Campus Area Network
- Virtual Area Network

Let's see all of them in detail:

1) WLAN

WLAN (Wireless Local Area Network) helps you to link single or multiple devices using wireless communication within a limited area like home, school, or office building. It gives users an ability to move around within a local coverage area which may be connected to the network. Today most modern day's WLAN systems are based on IEEE 802.11 standards.

2) Storage-Area Network (SAN)

A Storage Area Network is a type of network which allows consolidated, block-level data storage. It is mainly used to make storage devices, like disk arrays, optical jukeboxes, and tape libraries.

3) System-Area Network

System Area Network is used for a local network. It offers high-speed connection in server-to-server and processor-to-processor applications. The computers connected on a SAN network operate as a single system at quite high speed.

4) Passive Optical Local Area Network

POLAN is a networking technology which helps you to integrate into structured cabling. It allows you to resolve the issues of supporting Ethernet protocols and network apps.

POLAN allows you to use optical splitter which helps you to separate an optical signal from a single-mode optical fiber. It converts this single signal into multiple signals.

5) Home Area Network (HAN)

A Home Area Network is always built using two or more interconnected computers to form a local area network (LAN) within the home. For example, in the United States, about 15 million homes have more than one computer.

This type of network helps computer owners to interconnect with multiple computers. This network allows sharing files, programs, printers, and other peripherals.

6) Enterprise Private Network

Enterprise private network (EPN) networks are building and owned by businesses that want to securely connect numerous locations in order to share various computer resources.

7) Campus Area Network (CAN)

A Campus Area Network is made up of an interconnection of LANs within a specific geographical area. For example, a university campus can be linked with a variety of campus buildings to connect all the academic departments.

8) Virtual Private Network

A VPN is a private network which uses a public network to connect remote sites or users together. The VPN network uses "virtual" connections routed through the internet from the enterprise's private network or a third-party VPN service to the remote site.

It is a free or paid service that keeps your web browsing secure and private over public WiFi hotspots.

Summary:

- Type of computer networks can categorize according to their size as well as their purpose
- PAN is a computer network which generally consists of a computer, mobile, or personal digital assistant
- LAN (local area network) is a group of computer and peripheral devices which are connected in a limited area
- WAN (Wide Area Network) is another important computer network that which is spread across a large geographical area
- A metropolitan area network or MAN is consisting of a computer network across an entire city, college campus, or a small region
- WLAN is a wireless local area network that helps you to link single or multiple devices using. It uses wireless communication within a limited area like home, school, or office building.
- SAN is a storage area network is a type of network which allows consolidated, block-level data storage
- System area network offers high-speed connection in server-to-server applications, storage area networks, and processor-to-processor applications
- POLAN is a networking technology which helps you to integrate into structured cabling
- Home network (HAN) is a always built using two or more interconnected computers to form a local area network (LAN) within the home
- Enterprise private network (EPN) networks are build and owned by businesses that want to securely connect various locations
- Campus area network (CAN) is made up of an interconnection of LANs in a specific geographical area
- A VPN is a private network which uses a public network to connect remote sites or users together

Experiment No.: 3

Aim: Write a programs in C: hello_client (The server listens for, and accepts, a single TCP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection)

```
                    /* CLIENT PROGRAM FOR TCP CONNECTION */

#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket create and varification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

Compilation –

Server side:

```
gcc server.c -o server
./server
```

Client side:

```
gcc client.c -o client
./client
```

Output –

Server side:

```
Socket successfully created..

Socket successfully binded..
```

```
Server listening..
server acccept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```

Client side:

```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

Experiment No.: 4

Aim: Write a programs in C: hello_server for TCP
(The client connects to the server, sends the string “Hello, world!”, then closes the connection)

```
//server
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int sockfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(sockfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(sockfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");

    // Function for chatting between client and server
    func(connfd);

    // After chatting close the socket
    close(sockfd);
}
```

Compilation –

Server side:

```
gcc server.c -o server
./server
```

Client side:

```
gcc client.c -o client
./client
```

Output –

Server side:

```
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hi
    To client : hello
From client: exit
    To client : exit
Server Exit...
```

Client side:

```
Socket successfully created..
connected to the server..
```

```
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

Experiment No.: 5

Aim: Write a program to implement TCP Chat Server and UDP chat Server.

/* TCP Chat Server*/

// Program for chatappserver.c

```
#include<sys/socket.h>

#include<sys/types.h>

#include<stdio.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#include<string.h>

#include<unistd.h>

#define SER_PORT 1200

int main()

{

int a,sersock,newsock,n;

char str[25],str2[25];

struct sockaddr_in seraddr;

struct sockaddr_in cliinfo;

socklen_t csize=sizeof(cliinfo);

seraddr.sin_family=AF_INET;

seraddr.sin_port=htons(SER_PORT);

seraddr.sin_addr.s_addr=htonl(INADDR_ANY);

if((sersock=socket(AF_INET,SOCK_STREAM,0))<0)
```

```
{
error("\n socket");
exit(0);
}
if(bind(sersock,(struct sockaddr *)&seraddr,sizeof(seraddr))<0)
{
error("\nBIND");
exit(0);
}
if(listen(sersock,1)<0)
{
error("\n LISTEN");
}
if((newsock=accept(sersock,(struct sockaddr *)&cliinfo,&csize))<0)
{
error("\n ACCEPT");
exit(0);
}
else
printf("\n now connected to %s\n",inet_ntoa(cliinfo.sin_addr));
read(newsock,str,sizeof(str));
do
{
```

```
printf("\n client msg:%s",str);
printf("\n server msg:");
scanf("%s",str2);
write(newsock,str2,sizeof(str2));
listen(newsock,1);
read(newsock,str,sizeof(str));
n=strcmp(str,"BYE");
a=strcmp(str2,"BYE");
}
while(n!=0||a!=0);
close(newsock);
close(sersock);
return 0;
}
```

// Programfor chatappclient.c

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<unistd.h>
#define SER_PORT 1200
```

```
int main(int count,char*arg[])

{

int a,clisock;

char str[20],str2[20];

struct sockaddr_in cliaddr;

cliaddr.sin_port=htons(SER_PORT);

cliaddr.sin_family=AF_INET;

cliaddr.sin_addr.s_addr=inet_addr(arg[1]);

clisock=socket(AF_INET,SOCK_STREAM,0);

if(clisock<0)

{

perror("\n SOCKET");

exit(0);

}

if(connect(clisock,(struct sockaddr*)&cliaddr,sizeof(cliaddr))<0)

{

perror("\n CONNECT");

exit(0);

}

printf("\nclient connected to %s",arg[1]);

printf("\nCLIENT");

scanf("%s",&str);

if(write(clisock,str,sizeof(str))<0)
{
printf("\n data could not be sent");
}
do
{
listen(clisock,1);
read(clisock,str2,sizeof(str2));
printf("\nserver msg:%s",str2);
printf("\nclient msg:");
scanf("%s",&str);
a=strcmp(str2,"BYE");
write(clisock,str2,sizeof(str2));
}
while(a!=0);
close(clisock);
return 0;
}
```

/* UDP Chat Server */

/* udpserver.c */

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>
```

```
#include <errno.h>
#include <string.h>
#include <stdlib.h>

int main()
{
int sock;
int addr_len, bytes_read;
char recv_data[1024];
struct sockaddr_in server_addr , client_addr;


if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
{
perror("Socket");
exit(1);
        }

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(5000);
        server_addr.sin_addr.s_addr = INADDR_ANY;
bzero(&(server_addr.sin_zero),8);


if (bind(sock,(struct sockaddr *)&server_addr,sizeof(struct sockaddr))==-1)
        {
perror("Bind");
exit(1);
        }

        addr_len = sizeof(struct sockaddr);

            printf("\nUDPServer Waiting for client on port 5000");
fflush(stdout);

while (1)
{

bytes_read = recvfrom(sock,recv_data,1024,0,(struct sockaddr *)&client_addr, &addr_len);


              recv_data[bytes_read] = '\0';

printf("\n(%s,%d)said:",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
printf("%s", recv_data);
            fflush(stdout);
        }
return 0;
}
```

/* CLIENT PROGRAM FOR UDP CONNECTION */

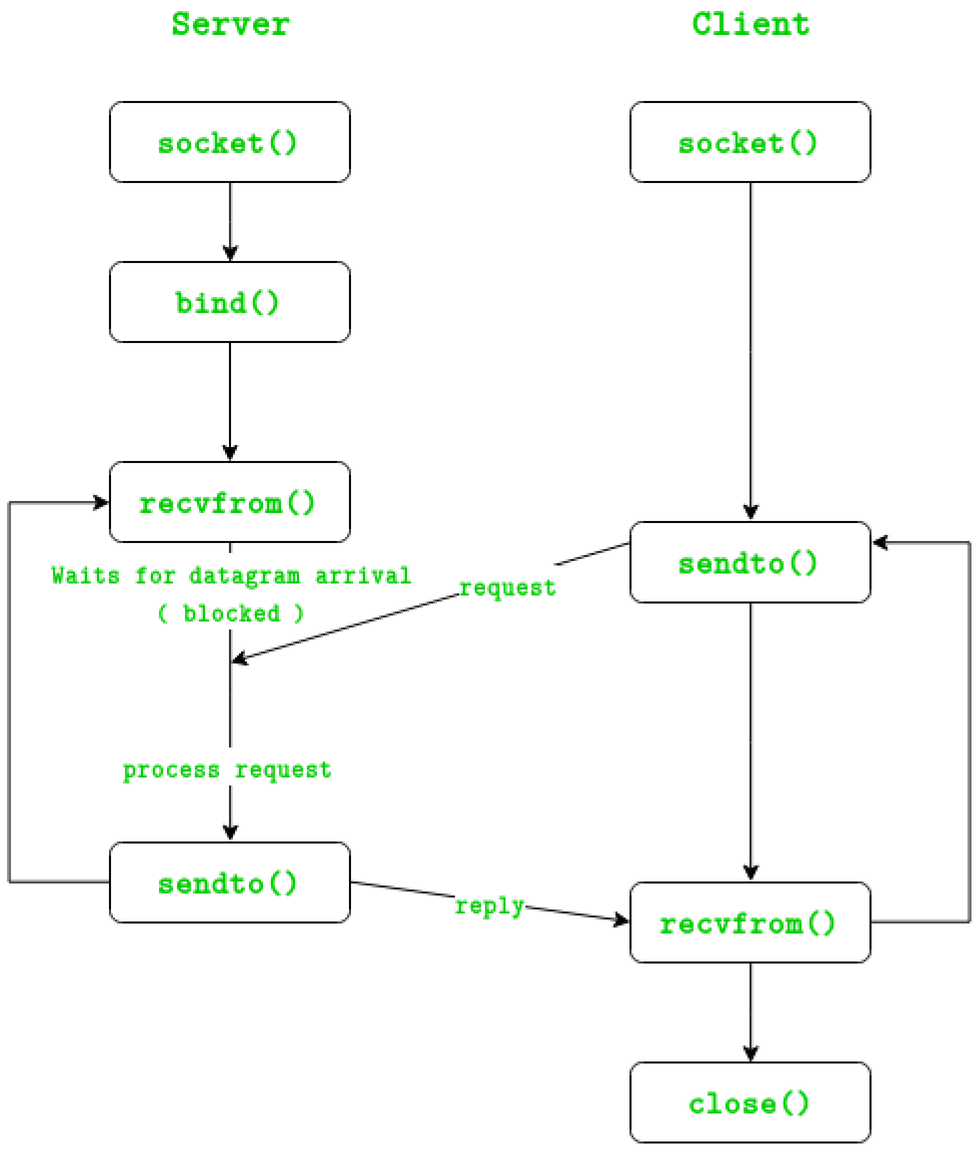
```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>

int main()
{
int sock;
struct sockaddr_in server_addr;
struct hostent *host;
char send_data[1024];

host= (struct hostent *) gethostbyname((char *)"127.0.0.1");


if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
{
perror("socket");
exit(1);
}

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(5000);
server_addr.sin_addr = *((struct in_addr *)host->h_addr);
bzero(&(server_addr.sin_zero),8);

while (1)
  {

printf("Type Something (q or Q to quit):");
gets(send_data);
if ((strcmp(send_data , "q") == 0) || strcmp(send_data , "Q") == 0)
break;

else
sendto(sock, send_data, strlen(send_data), 0,
            (struct sockaddr *)&server_addr, sizeof(struct sockaddr));

   }

}
```

Experiment No.: 6

Aim: Write a programs in C: hello_client (The server listens for, and accepts, a single UDP connection; it reads all the data it can from that connection, and prints it to the screen; then it closes the connection)

Theory

In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.



The entire process can be broken down into following steps :

UDP Server :

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

UDP Client :

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

Necessary Functions :

```
int socket(int domain, int type, int protocol)

Creates an unbound socket in the specified domain.

Returns socket file descriptor.
```

Arguments :
domain – Specifies the communication
domain (AF_INET for IPv4/ AF_INET6 for IPv6)
type – Type of socket to be created
(SOCK_STREAM for TCP / SOCK_DGRAM for UDP)
protocol – Protocol to be used by socket.
0 means use default protocol for the address family.

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

Assigns address to the unbound socket.
```

Arguments :
sockfd – File descriptor of socket to be binded
addr – Structure in which address to be binded to is specified
addrlen – Size of *addr* structure

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,

            const struct sockaddr *dest_addr, socklen_t addrlen)

Send a message on the socket
```

Arguments :
sockfd – File descriptor of socket
buf – Application buffer containing the data to be sent
len – Size of *buf* application buffer
flags – Bitwise OR of flags to modify socket behaviour
dest_addr – Structure containing address of destination
addrlen – Size of *dest_addr* structure

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                struct sockaddr *src_addr, socklen_t *addrlen)
```

Receive a message from the socket.

Arguments :
sockfd – File descriptor of socket
buf – Application buffer in which to receive data
len – Size of *buf* application buffer
flags – Bitwise OR of flags to modify socket behaviour
src_addr – Structure containing source address is returned
addrlen – Variable in which size of *src_addr* structure is returned

```
int close(int fd)
```

Close a file descriptor

Arguments :
fd – File descriptor
In the below code, exchange of one hello message between server and client is shown to demonstrate the model.

```
// Client side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE 1024

// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from client";
    struct sockaddr_in     servaddr;

    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    int n, len;

    sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &servaddr,
            sizeof(servaddr));
    printf("Hello message sent.\n");

    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                MSG_WAITALL, (struct sockaddr *) &servaddr,
                &len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);

    close(sockfd);
    return 0;
}
```

Output :

```
$ ./server
Client : Hello from client
Hello message sent.
$ ./client
Hello message sent.
Server : Hello from server
```

Experiment No.: 7

Aim: Write a programs in C: hello_server
(The client connects to the server, sends the string “Hello, world!”, then closes the UDP connection)

```
// Server side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE 1024

// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family    = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    // Bind the socket with the server address
    if ( bind(sockfd, (const struct sockaddr *)&servaddr,
            sizeof(servaddr)) < 0 )
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n;
    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                MSG_WAITALL, ( struct sockaddr *) &cliaddr,
                &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);
    sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
            len);
    printf("Hello message sent.\n");

    return 0;
}
```

Output :

```
$ ./server

Client : Hello from client

Hello message sent.

$ ./client

Hello message sent.

Server : Hello from server
```

Experiment No.: 8

Aim: Write an Echo_server using TCP to estimate the round trip time
from client to the server. The server should be such that it can accept multiple connections at any given time , with multiplexed I/O operations

Code for Client (For echo server)

```
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <unistd.h>

#define MAXCOUNT 1024

int main(int argc, char* argv[])
{
int sfd;
char msg[MAXCOUNT];
char blanmsg[MAXCOUNT];
struct sockaddr_in saddr;

memset(&saddr,0,sizeof(saddr));
sfd = socket(AF_INET,SOCK_STREAM,0);
   saddr.sin_family = AF_INET;
   inet_pton(AF_INET,"127.0.0.1",&saddr.sin_addr);
   saddr.sin_port = htons(5004);

connect(sfd,(struct sockaddr*) &saddr, sizeof(saddr));
for(; ;) {
memset(msg,0,MAXCOUNT);
memset(blanmsg,0,MAXCOUNT);
fgets(msg,MAXCOUNT,stdin);
send(sfd,msg,strlen(msg),0);
recv(sfd,blanmsg,sizeof(blanmsg),0);
printf("%s",blanmsg);
fflush(stdout);
   }
exit(0);
}
```

Here is the code for the server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#define MAXCOUNT 1024

int main(int argc, char* argv[])
{
int sfd,nsfd,n,i,cn;
char buf[MAXCOUNT];
socklen_t caddrlen;
struct sockaddr_in caddr,saddr; //Structs for Client and server Address in the Internet

sfd = socket(AF_INET,SOCK_STREAM,0);
memset(&saddr,0,sizeof(saddr)); //Clear the Server address structure

   saddr.sin_family = AF_INET; //Internet Address Family
   saddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
   saddr.sin_port = htons(5004);

bind(sfd, (struct sockaddr*) &saddr,sizeof(saddr));
listen(sfd,1);

for(; ;) {
caddrlen = sizeof(caddr);
nsfd = accept(sfd,(struct sockaddr*) &caddr,&caddrlen);
cn = recv(nsfd,buf,sizeof(buf),0);
if(cn == 0) {
exit(0);
        }
```

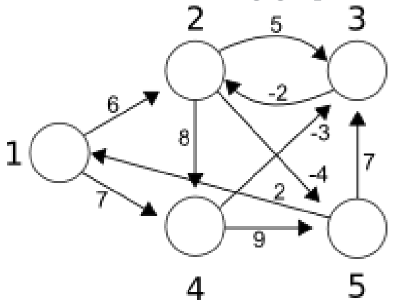
Experiment No.: 9

Aim: Program to simulate Bellman Ford Routing Algorithm

Algorithm:

The Problem

Given the following graph, calculate the length of the shortest path from **node 1** to **node 2**.



It's obvious that there's a direct route of length *6*, but take a look at path: *1 -> 4 -> 3 -> 2*. The length of the path is *7 – 3 – 2 = 2*, which is less than *6*. BTW, you don't need negative edge weights to get such a situation, but they do clarify the problem.

This also suggests a property of shortest path algorithms: to find the shortest path form *x*to *y*, you need to know, beforehand, the shortest paths to *y*'s neighbours. For this, you need to know the paths to *y*'s neighbours' neighbours… In the end, you must calculate the shortest path to the connected component of the graph in which *x* and *y* are found.

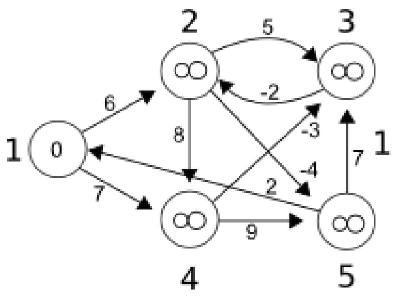
That said, you usually calculate **the shortest path to all nodes** and then pick the ones you're intrested in.

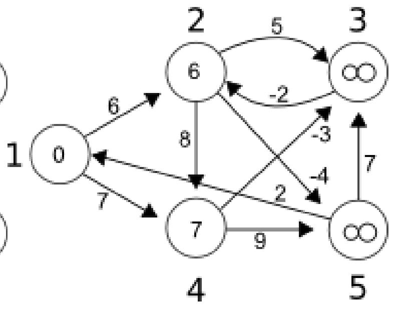
The Algorithm

The Bellman-Ford algorithm is one of the classic solutions to this problem. It calculates the shortest path to all nodes in the graph from a single source.

The basic idea is simple:
Start by considering that the shortest path to all nodes, less the source, is infinity. Mark the length of

the path to the source as *0*:

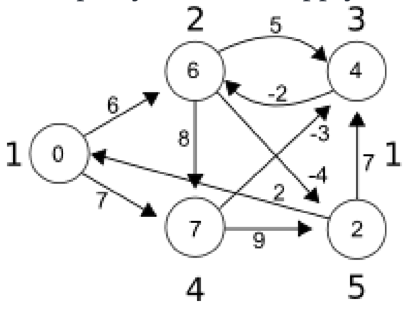


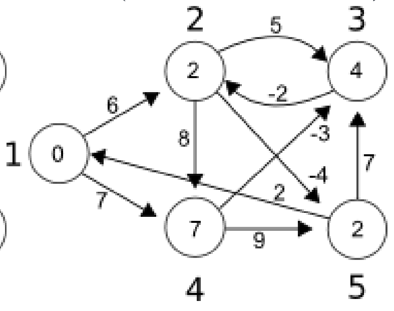


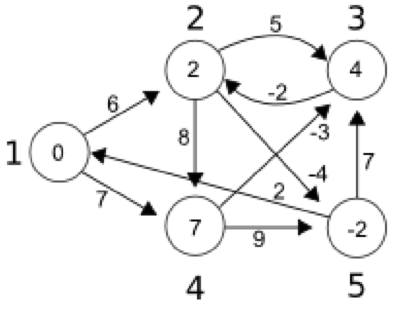
Take every edge and try to *relax* it:

Relaxing an edge means checking to see if the path to the node the edge is pointing to can't be shortened, and if so, doing it. In the above graph, by checking the **edge 1 -> 2** of length *6*, you find that the length of the shortest path to **node 1** plus the length of the**edge 1 -> 2** is less then infinity. So, you replace infinity in **node 2** with *6*. The same can be said for edge *1 -> 4* of length *7*. It's also worth noting that, practically, you can't relax the edges whose start has the shortest path of length infinity to it.

Now, you apply the previous step *n – 1* times, where n is the number of nodes in the graph. In this example, you have to apply it *4* times (that's *3* more times).







Here, **d[i]** is the shortest path to node **i**, **e** is the number of edges and **edges[i]** is the **i**-th edge.

It may not be obvious why this works, but take a look at what is certain after each step. After the first step, any path made up of at most *2* nodes will be optimal. After the step *2*, any path made up of at most *3* nodes will be optimal… After the *(n – 1)*-th step, any path made up of at most *n* nodes will be optimal.

The Programme

The following programme just puts the **bellman_ford** function into context. It runs in**O(VE)** time, so for the example graph it will do something on the lines of **5 * 9 = 45**relaxations. Keep in mind that this algorithm works quite well on graphs with few edges, but is very slow for dense graphs (graphs with almost *n^2* edges)

```
#include <stdio.h>

typedef struct {
        int u, v, w;
} Edge;

int n; /* the number of nodes */
int e; /* the number of edges */
Edge edges[1024]; /* large enough for n <= 2^5=32 */
int d[32]; /* d[i] is the minimum distance from node s to node i */

#define INFINITY 10000

void printDist() {
        int i;

        printf("Distances:\n");

        for (i = 0; i < n; ++i)
                printf("to %d\t", i + 1);
        printf("\n");

        for (i = 0; i < n; ++i)
                printf("%d\t", d[i]);

        printf("\n\n");
}
```

```
void bellman_ford(int s) {
	int i, j;

	for (i = 0; i < n; ++i)
		d[i] = INFINITY;

	d[s] = 0;

	for (i = 0; i < n - 1; ++i)
		for (j = 0; j < e; ++j)
			if (d[edges[j].u] + edges[j].w < d[edges[j].v])
				d[edges[j].v] = d[edges[j].u] + edges[j].w;
}

int main(int argc, char *argv[]) {
	int i, j;
	int w;

	FILE *fin = fopen("dist.txt", "r");
	fscanf(fin, "%d", &n);
	e = 0;

	for (i = 0; i < n; ++i)
		for (j = 0; j < n; ++j) {
			fscanf(fin, "%d", &w);
			if (w != 0) {
				edges[e].u = i;
				edges[e].v = j;
				edges[e].w = w;
				++e;
			}
		}
	fclose(fin);

	/* printDist(); */

	bellman_ford(0);

	printDist();

	return 0;
```

}
And here’s the input file used in the example (dist.txt):

```
5
0 6 0 7 0
0 0 5 8 -4
0 -2 0 0 0
0 0 -3 9 0
2 0 7 0 0
```

Experiment No.: 10

Aim: Program to simulate of sliding window protocol.

Algorithm: Step 1: Start the program.
Step 2: Include the necessary header files.
Step 3: To create the socket using socket() function.
Step 4: Enter the number of frames.
Step 5: And the corresponding message is send to receiver.
Step 6: Acknowledgement is received by receiver.
Step 7: If u send another message,ACK 2 message is received.
Step 8: Send the acknowledgement to sender.
Step 9: Print out with the necessary details.
Step 10: Stop the program.

SOURCE CODE:

Server:
```
#include<string.h>
#include<stdio.h>
#include<netdb.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main(int argc,char ** argv)
{
struct sockaddr_in saddr,caddr;
int r,len,ssid,csid,pid,pid1,i,n;
char wbuffer[1024],rbuffer[1024];
float c;
if(argc<2)
fprintf(stderr,"Port number not specified\n");
ssid=socket(AF_INET,SOCK_STREAM,0);
if(ssid<0)
perror("Socket failed\n");
bzero((char *)&saddr,sizeof(saddr));
saddr.sin_family=AF_INET;
saddr.sin_port=htons(atoi(argv[1]));
saddr.sin_addr.s_addr=INADDR_ANY;
if(bind(ssid,(struct sockaddr *)&saddr,sizeof(saddr))<0)
perror("Socket Bind\n");
listen(ssid,5);
len=sizeof(caddr);
csid=accept(ssid,(struct sockaddr *)&caddr,&len);
if(csid<0)
perror("Socket Accept\n");
fprintf(stdout,"TYPE MESSAGE TO CLIENT\n");
pid=fork();
if(pid==0)
{
while(1)
{
bzero(rbuffer,1024);
n=read(csid,rbuffer,1024);
if(n==0)
perror("Socket read\n");
else
fprintf(stdout,"MESSAGE FROM CLIENT: %s\n",rbuffer);
}
exit(0);
}
else
{
while(1)
{
bzero(wbuffer,1024);
fgets(wbuffer,1024,stdin);
n=write(csid,wbuffer,1024);
if(n==0)
perror("Socket Write");
}
}
return(0);
}
```

CLIENT:

```
#include<stdio.h>
#include<netdb.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main(int argc,char ** argv)
{
struct sockaddr_in saddr;
struct hostent *server;
int n,ssid,csid,pid,pi;
char wbuffer[1024],rbuffer[1024];
char str[15];
if(argc<3)
fprintf(stderr,"Parameter inadequate\n");
csid=socket(AF_INET,SOCK_STREAM,0);
if(csid<0)
perror("Socket Failed\n");
bzero((char *)&saddr,sizeof(saddr));
server=gethostbyname(argv[1]);
saddr.sin_family=AF_INET;
saddr.sin_port=htons(atoi(argv[2]));
bcopy((char *)server->h_addr,(char *)&saddr.sin_addr.s_addr,server->h_length);
ssid=connect(csid,(struct sockaddr *)&saddr,sizeof(saddr));
if(ssid<0)
perror("Socket Connect\n");
fprintf(stdout,"ENTER MESSAGE TO SERVER:\n");
pid=fork();
if(pid==0)
{
while(1)
{
bzero(wbuffer,1024);
fgets(wbuffer,1024,stdin);
n=write(csid,wbuffer,sizeof(wbuffer));
if(n==0)
perror("Socket Write");
}
exit(0);
}
else
{
while(1)
{
bzero(rbuffer,1024);
n=read(csid,rbuffer,sizeof(rbuffer));
if(n==0)
perror("Socket Read\n");
else
fprintf(stdout,"MESSAGE FROM SERVER: %s\n",rbuffer);
}
return(0);
}
}
```

OUTPUT:

SERVER:

```
[05mecse090@networkserver ~]$ cc chatserv.c
[05mecse090@networkserver ~]$./a.out 9898
TYPE MESSAGE TO CLIENT
MESSAGE FROM CLIENT: hi
```

```
hai
```

CLIENT:

```
[05mecse090@networkserver ~]$ cc chatcli.c
[05mecse090@networkserver ~]$cc chatcli.c
[05mecse090@networkserver ~]$ ./a.out 127.0.0.1
ENTER MESSAGE TO SERVER:
hi
MESSAGE FROM SERVER: hai
```

RESULT:

Thus the c program for the simulation of sliding window protocol has beenexecuted and the output is verified successfully.

Experiment No.: 11

Aim: Program to simulate file transfer protocol.

Algorithm:
Client
```
Step 1: start the program
Step 2: Declare the variables and structure for sockets
Step 3: And then get the port number
Step 4: Create a socket using socket functions
Step 5: The socket is binded at the specified port
Step 6: Using the object, the port and address are declared
Step 7: Get the source file and the destination file from the user
Step 8: Use the send command for sending the two strings
Step 9: Receive the bytes sent from the server
Step 10: Print it in the console
Step 11: Close the socket
```
Server
```
Step 1: Start the program
Step 2: Declare the variables and structure for sockets
Step 3: And then get the port number
Step 4: Create a socket using socket functions
Step 5: Use the connect command for socket connection
Step 6: Use bind option to bind the socket address
Step 7: Use accept command to receive the connection from the client
Step 8: Receive command from the client
Step 9: Send the file to the client socket
Step 10: Close the connection
```

PROGRAM:

SERVER:

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
int main()
{
int sd,nsd,i,port=1234;
char content[100]="\0",fname[100]="\0";
struct sockaddr_in ser,cli;
FILE *fp;
if((sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1)
{
printf("ERROR::SOCKET CREATION PROBLEM--CHECK THE PARAMETERS.\n");
return 0;
}
bzero((char *)&ser,sizeof(ser));
printf("THE PORT ADDRESS IS: %d\n",port);
ser.sin_family=AF_INET;
ser.sin_port=htons(port);
ser.sin_addr.s_addr=htonl(INADDR_ANY);
if(bind(sd,(struct sockaddr *)&ser,sizeof(ser))==-1)
{
printf("\nERROR::BINDING PROBLEM, PORT BUSY--PLEASE CSS IN THE SER AND
CLI\n");
return 0;
}
i=sizeof(cli);
listen(sd,1);
printf("\nSERVER MODULE\n");
printf("********************\n");
nsd=accept(sd,(struct sockaddr *)&cli,&i);
if(nsd==-1)
{
printf("\nERROR::CLIENT ACCEPTIN PROBLEM--CHECK THE DEIPTOR
PARAMETER.\n\n");
return 0;
}
printf("\nCLIENT ACCEPTED");
i=recv(nsd,fname,30,0);
fname[i]='\0';
fp=fopen(fname,"rb");
while(1)
{
i=fread(&content,1,30,fp);
content[i]='\0';
send(nsd,content,30,0);
strcpy(content,"\0");
if(i<30)
break;
}
send(nsd,"EOF",4,0);
printf("\nFILE TRANSFERED TO DESTINATION\n\n");
fclose(fp);
close(sd);
close(nsd);
return 0;
}
```

CLIENT:

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/socket.h>
int main()
{
int sd,i,port=1234;
char content[100]="\0",fname[100]="\0",file[100]="\0";
struct sockaddr_in ser;
FILE *fp;
if((sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))==-1)
{
printf("\nERROR::SOCKET CREATION PROBLEM--CHECK THE
PARAMETER.\n\n");
return 0;
}
bzero((char *)&ser,sizeof(ser));
printf("\nTHE PORT ADDRESS IS: %d\n",port);
ser.sin_family=AF_INET;
ser.sin_port=htons(port);
ser.sin_addr.s_addr=htonl(INADDR_ANY);
if(connect(sd,(struct sockaddr *)&ser,sizeof(ser))==-1)
{
printf("\nERROR::CANT CONNECT TO SERVER--CHECK PARAMETERS.\n\n");
return 0;
}
printf("\nTHIS IS THE CLIENT MODULE. THIS MODULE CAN ASK THE SERVER A
FILE");
printf("\n***********************************************************************\n\n");
printf("\nENTER THE PATHNAME OF SOURCE FILE::\n");
scanf("%s",fname);
printf("\nENTER THE PATHNAME OF DESTINATION FILE::\n");
scanf("%s",file);
send(sd,fname,30,0);
fp=fopen(file,"wb");
while(1)
{
i=recv(sd,content,30,0);
content[i]='\0';
if(!strcmp(content,"EOF"))
break;
//fwrite(&content,strlen(content),1,fp);
printf("%s",content);
strcpy(content,"\0");
}
printf("\n\nFILE RECEIVED\n\n");
fclose(fp);
close(sd);
return 0;
}
```

OUTPUT:

SERVER:

```
[3itb41@TELNET ~]$ cd ftpser.c
[3itb41@TELNET serv]$ ./a.out

THE PORT ADDRESS IS: 1234
SERVER MODULE
********************
CLIENT ACCEPTED
FILE TRANSFERED TO DESTINATION
```

CLIENT:

```
[3itb41@TELNET cli]$ cc ftpcli.c
[3itb41@TELNET cli]$ ./a.out
THE PORT ADDRESS IS: 1234
THIS IS THE CLIENT MODULE. THIS MODULE CAN ASK THE SERVER A FILE
*****************************************************************
ENTER THE PATHNAME OF SOURCE FILE::
/home/3itb41/file1.html
ENTER THE PATHNAME OF DESTINATION FILE::
fp1.html
<HTML>
<BODY>
Hi
</BODY>
</HTML>
FILE RECEIVED
```

RESULT:

Thus the c program for transferring files from one machine to another machine using TCP is executed and the output is verified successfully.

Viva Questions
Experiment No.1

1 What is Network Topology?

Network Topology is a physical layout of the computer network and it defines how the computers, devices, cables etc are connected to each other.

2 Describe star topology

Star topology consists of a central hub that connects to nodes. This is one of the easiest to setup and maintain.

3 Which topology there is a central controller or hub?

In star topology a main hub is present to which all other nodes of the network are connected. Every data or information being transmitted or received in this topology has to pass through the hub. The hub directs the data to its destination.

4 What is Physical or logical arrangement of network

Topology in networks is the structure or pattern in which each and every node in the network is connected. There are many topologies in networking like bus, tree, ring, star, mesh, and hybrid.

5 What is topology requires multipoint connection

In bus topology, there is a single cable to which all the network nodes are connected. So whenever a node tries to send a message or data to other nodes, this data passes through all other nodes in the network.

Experiment No.:2

1 What is a Network?

A network is a set of devices connected to each other using a physical transmission medium.

2 What is a Node?

Two or more computers are connected directly by an optical fiber or any other cable. A node is a point where a connection established. It is a network component that is used to send, receive and forward the electronic information.

3 What is WAN?

WAN stands for Wide Area Network. It is an interconnection of computers and devices that are geographically dispersed. It connects networks that are located in different regions and countries.

4 What is VPN?

VPN means Virtual Private Network, a technology that allows a secure tunnel to be created across a network such as the Internet. For example, VPNs allow you to establish a secure dial-up connection to a remote server.

5 What is backbone network?

A backbone network is a centralized infrastructure that is designed to distribute different routes and data to various networks. It also handles management of bandwidth and various channels.

Experiment No.: 3

1 What is TCP/IP?

TCP/IP is short for Transmission Control Protocol / Internet Protocol. This is a set of protocol layers that is designed to make data exchange possible on different types of computer networks, also known as heterogeneous network.

2 What is DNS?

DNS is Domain Name System. The main function of this network service is to provide host names to TCP/IP address resolution.

3 What protocols fall under the Application layer of the TCP/IP stack?

The following are the protocols under TCP/IP Application layer: FTP, TFTP, Telnet and SMTP.

4 What is client/server?

Client/server is a type of network wherein one or more computers act as servers. Servers provide a centralized repository of resources such as printers and files. Clients refers to workstation that access the server.

5 Describe Ethernet.

Ethernet is one of the popular networking technologies used these days. It was developed during the early 1970s and is based on specifications as stated in the IEEE. Ethernet is used in local area networks.

Experiment No.: 4

1 What Is A Transaction Server?

With a transaction server, the client invokes remote procedures that reside on the server with an SQL database engine. These remote procedures on the server execute a group of SQL statements. The network exchange consists of a single request/reply message. The SQL statements either all succeed or fail as a unit.

2 Explain the Building Blocks of Client/server?

The client side building block runs the client side of the application.
The server side building block runs the server side of the application.

3 What are the Characteristics of Client/server?

Service, Shared resources, Transparency of location, Mix-and-match, Message based exchanges; Encapsulation of services, Scalability, Integrity, and Client/Server computing is the ultimate "Open platform". It gives the freedom to mix-and-match components of almost any level. Clients and servers are loosely coupled systems that interact through a message-passing mechanism.

4 What is meant by 3-tier Architecture?

In 3-tier Client/Server systems, the application logic (or process) lives in the middle tier and it is separated from the data and the user interface. In theory, the 3-tier Client/Server systems are more scalable, robust and flexible.
Example: TP monitor, Web.

5 What Are Called Fat Clients And Fat Servers?

If the bulk of the application runs on the Client side, then it is Fat clients. It is used for decision support and personal software.

If the bulk of the application runs on the Server side, then it is Fat servers. It tries to minimize network interchanges by creating more abstract levels of services.

Experiment No.: 5

1 What Is The Role Of TCP/IP In Data Transmission From Source To Destination?

Yes, lots of them, far too many to list here. Uri Razz maintains a TCP/IP bibliography (the "TCP/IP Resources List") that is posted to the comp.protocols.tcp-ip newsgroup on a monthly basis.

2 TCP/IP Has How Many Layers?

5 layers: Network layer, Internet layer, Transport layer and Application layer.

3 What Is TCP/IP?

TCP/IP is a name given to the collection (or suite) of networking protocols that have been used to construct the global Internet. The protocols are also referred to as the DOD (Dee-oh-Dee) or Arpanet protocol suite because their early development was funded by the Advanced Research Projects Agency (ARPA) of the US Department of Defense (DOD).

4 What Is Subnet Mask?

A subnet mask is combined with an IP address in order to identify two parts: the extended network address and the host address. Like an IP address, a subnet mask is made up of 32 bits.

5 What Do Mean By Tunnel Model?

This is a mode of data exchange wherein two communicating computers do not use IPSec themselves. Instead, the gateway that is connecting their LANs to the transit network creates a virtual tunnel that uses the IPSec protocol to secure all communication that passes through it.

Experiment No.:6

1 What Are The Major Difference Between UDP And TCP/IP Protocol?

The first thing is UDP is connection less where as TCP is connection

Oriented The broadcasting and multicasting software is available in the UDP only Why because it is connection less hence we can broadcast the packets very easily, no need to wait for connection like in tcp, which takes more delay.

2 Write Udp/socket diagram Applications?

To implement the udp service we must create socket by socket system call which takes argument as SOCK_DGRAM which is used for to pass the data in the form of datagram's.

3 What Do You Mean About ISP, And What Is Work?

ISP: (internet service providers) who provides internet services

Work of ISP: providing internet services i.e. ips

1. web server services
2. Virtual hosting(manage web server ,domain and users internet related works)

4 Is The Ip Address Of Computer And Modems Is Same Or Not?

No. The IP address of a system is the logical address where as the address of the MODEM is the MAC (Media Access Control) address, it is the physical address provided by the vendor.

5 How Many Types Of Transmission Are There?

Two types of transmissions

1. Serial
2. parallel

Serial means sending one bit at a time on a single wire used over long distances more efficient used to send data to external systems

Parallel means sending bits at a time on different wires used over short distances efficient but not serial used to send data in internal transfers

Experiment No.: 7

1. What is IP address?

The 32 bits Internet address is that which defines a host or router. Uniquely and universally on the internet. The portion that identifies the network is of the IP address called the net id. The portion identifies the host or router on the network of the IP address that is called the host id. An IP address defines connection to a network of a device.

2. Explain the three types of addresses in TCP/IP?

Three types of addresses used by computers using the TCP/IP

- Physical address,
- Internetwork address
- Port address.

The physical address, is the address as defined by its LAN or WAN of a node. The IP address defines a host on the Internet uniquely. the port address is an identifier which identifies a process on a host.

3. What is the function of routing table?

Each host or router contains a routing table to route IP packets. In next hop routing the

TCP/IP questions

Packets make only the address of the next hop which is listed in the routing table. All hosts on a network share one entry in the routing table in network specific routing in host specific in the routing table full IP address of a host is given routing. A router is assigned to receive packets with no match in the routing table in default routing.

4. What are the fields included in routing table?

The routing table consists of seven fields: These are a mask, a address of destination, a address of next-hop, flags, reference count, use, and interface. The routing module applies the mask row by row, to the received address of destination till a match is found. Classless addressing requires geographical and hierarchical routing for preventing immense routing tables.

5. What is Fragmentation?

It is the division of a datagram into smaller units to accommodate of a data link protocol's MTU.
The fields in the IP header which is related to fragmentation are the identification
number, the flags fragmentation, and the offset fragmentation. The IP datagram header is consists of a fixed, 20- byte section and also a variable options section with a maximum of 40 bytes.

Experiment No.: 8

1 How does the race condition occur?
It occurs when two or more processes are reading or writing some shared data and the final result depends on who runs precisely when.

2 What is multiprogramming?
Multiprogramming is a rapid switching of the CPU back and forth between processes.

3 Name the seven layers of the OSI Model and describe them briefly.
Physical Layer - covers the physical interface between devices and the rules by which bits are passed from one to another.
Data Link Layer - attempts o make the physical link reliable and provides the means to activate, maintain, and deactivate the link.
Network Layer - provides for the transfer of information between end systems across
some sort communications network.
Transport Layer - provides a mechanism for the exchange of data between end systems.
Session Layer - provides the mechanism for controlling the dialogue between applications in end systems.
Presentation Layer - defines the format of the data to be exchanged between applications and offers application programs a set of data transformation services.
Application Layer - provides a means for application programs to access the OSI environment.

4 What is the difference between TCP and UDP?
TCP and UDP are both transport-level protocols. TCP is designed to provide reliable communication across a variety of reliable and unreliable networks and internets.
UDP provides a connectionless service for application-level procedures. Thus, UDP is basically an unreliable service; delivery and duplicate protection are not guaranteed.

5 What does a socket consists of?
The combination of an IP address and a port number is called a socket.

6 What are some advantages and disadvantages of Java Sockets?

Advantages of Java Sockets:
Sockets are flexible and sufficient. Efficient socket based programming can be easily implemented for general communications.
Sockets cause low network traffic. Unlike HTML forms and CGI scripts that generate and transfer whole web pages for each new request, Java applets can send only necessary updated information.
Disadvantages of Java Sockets:
Security restrictions are sometimes overbearing because a Java applet running in a Web browser is only able to establish connections to the machine where it came from, and to nowhere else on the network
Despite all of the useful and helpful Java features, Socket based communications allows only to send packets of raw data between applications.

Experiment No.: 9

1 Explain What Is Difference between Baseband and Broadband Transmission

In the baseband transmission, the entire bandwidth of the cable is consumed by a single signal. In broadband transmission, signals are sent on multiple frequencies, allowing multiple signals to be sent simultaneously.

In baseband transmission we transmit digital signal without converting it into analog, here a low pass channel is used.

In broadband transmission we transmit digital signal by converting it into analog. Here a band pass channel is used.

2 Explain Ping Utility?

PING stands Packet Internet Gopher. This is a utility for ensuring connectivity between computer. ICMP protocol works behind this utility. Under it , sending node sends packets to destination node and reply is received if there is proper communication between two.

PING: Packet Internet Gropper

It’s a diagnostic utility, which diagnose devices connectivity.

It use ICMP: Internet Control Messaging protocol to send echo requests (usually 4 packets) and receive echo replies (4 packets)

3 Explain What Is NetBIOS And NetBIOS?

NETBIOS is a programming interface that allows I/O requests to be sent to and received from a remote computer and it hides the networking hardware from applications.

NETBEUI is NetBIOS extended user interface. A transport protocol designed by Microsoft and IBM for the use on small subnets.

NETBIOS is a programming interface that allows I/O request to be sent to and received from a remote computer and it hides the networking hardware from applications.

NETBEUI is a NetBIOS extended user interface. A transport protocol designed by Microsoft and IBM for the use on small subnets.

4 Explain Can We Have Too Many Apes And What Is The Impact?

We can absolutely have too many APs and this can actually be more troublesome than too few APs. When devices on the same channel are co-located without enough channel separation the result is wasted equipment and reduced performance.

5 What Is a Wireless LAN (WLAN)?

Very simply, a WLAN is a wireless or radio frequency extension of a LAN. Some WLANs however, such as ad-hoc networks have no wired components. Generally these networks are based on the IEEE 802.11 protocol suite but can also consist of proprietary communication protocols.

Experiment No.: 10

1 What is Go – Back – *N* ARQ

Go – Back – *N* ARQ provides for sending multiple frames before receiving the acknowledgment for the first frame. It uses the concept of sliding window, and so is also called sliding window protocol. The frames are sequentially numbered and a finite number of frames are sent. If the acknowledgment of a frame is not received within the time period, all frames starting from that frame are retransmitted.

2 What is Selective Repeat ARQ?

This protocol also provides for sending multiple frames before receiving the acknowledgment for the first frame. However, here only the erroneous or lost frames are retransmitted, while the good frames are received and buffered.

3 What is Sliding window protocol?

Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. The sliding window is also used in Transmission Control Protocol. In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames. Sliding window method is also known as windowing.

4 Why Selective Repeat Protocol?

The go-back-n protocol works well if errors are less, but if the line is poor it wastes a lot of bandwidth on retransmitted frames. An alternative strategy, the selective repeat protocol, is to allow the receiver to accept and buffer the frames following a damaged or lost one. Selective Repeat attempts to retransmit only those packets that are actually lost (due to errors)

5 What are Retransmission Requests?

Implicit – The receiver acknowledges every good packet, packets that are not ACKed before a time-out are assumed lost or in error. Notice that this approach must be used to be sure that every packet is eventually received.

Explicit – An explicit NAK (selective reject) can request retransmission of just one packet. This approach can expedite the retransmission but is not strictly needed

Experiment No.:11

1 What Is Ftp?

FTP stands for File Transfer Protocol. An FTP server allows clients to connect to it either anonymously or with a username and password combination. After successful authentication, files can be transferred back and forth between the server and client. The files are neither encrypted nor compressed.

2 How to Deny Specific Users Access to the Ftp Server?

To deny specific user’s access to the FTP server, add their usernames to the /etc/vsftpd/ fetuses file. By default, system users such as root and nobody are included in this list.

3 Important Configuration File for vsftpd Server?

The FTP server uses the /etc/vsftpd/vsftpd.conf configuration file. Using this file, you can set options for displaying a custom banner message after users log in, setting the default file permissions for uploaded files, and setting the port on which to listen for incoming connections.

4 What Is Passive Mode?

Passive mode, like active mode, is initiated by the FTP client application. When requesting data from the server, the FTP client indicates it wants to access the data in passive mode and the server provides the IP address and a random, unprivileged port (greater than 1024) on the server. The client then connects to that port on the server to download the requested information.

5 Explain Directive "session support”?

When enabled, vsftpd attempts to maintain login sessions for each user through Pluggable Authentication Modules (PAM).

6 Explain Ftp Spoofing Attack?

In the context of network security, a spoofing attack is a situation in which one person or program successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage.