Experiment No.: 1

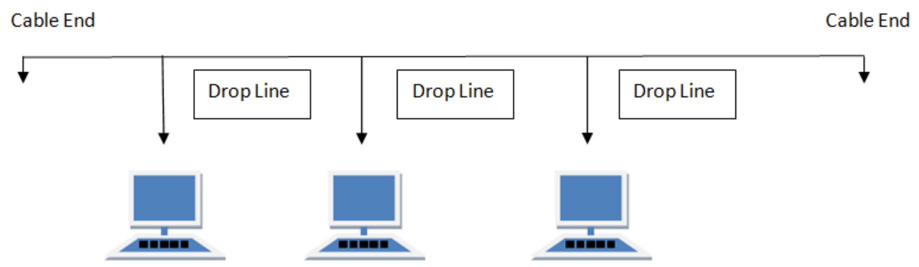
Aim. Study of different types of network topologies.

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

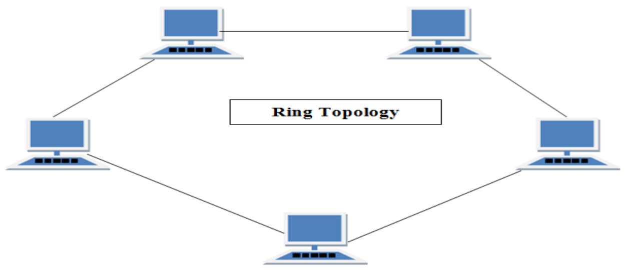
1. It is cost effective.
2. Cable required is least compared to other network topology.
3. Used in small networks.
4. It is easy to understand.
5. Easy to expand joining two cables together.

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

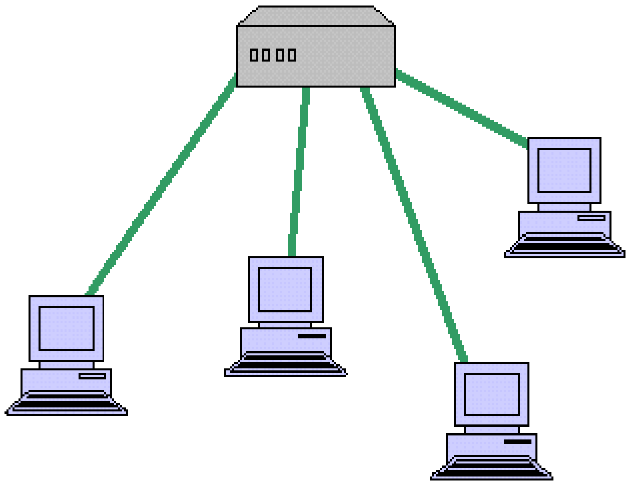
1. Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
2. Cheap to install and expand

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



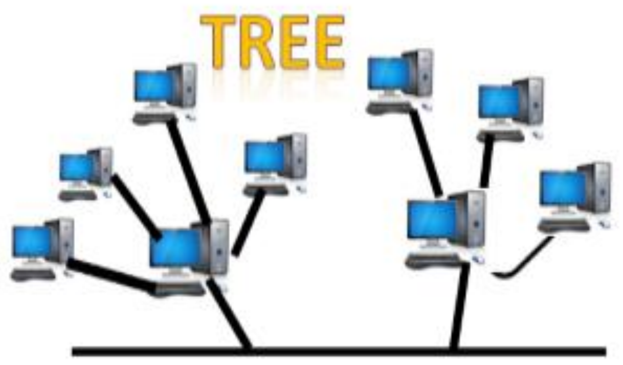
Advantages of Star topology

- **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.

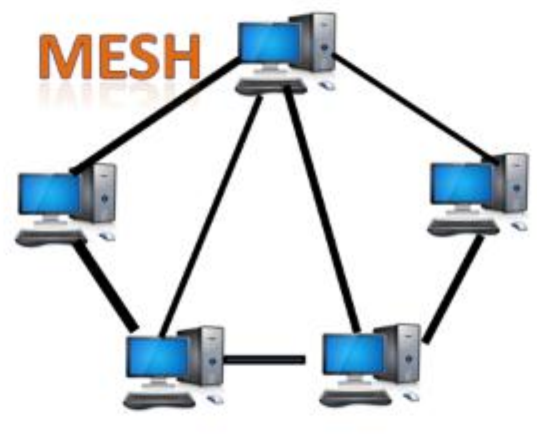
Advantages of Tree topology

- **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

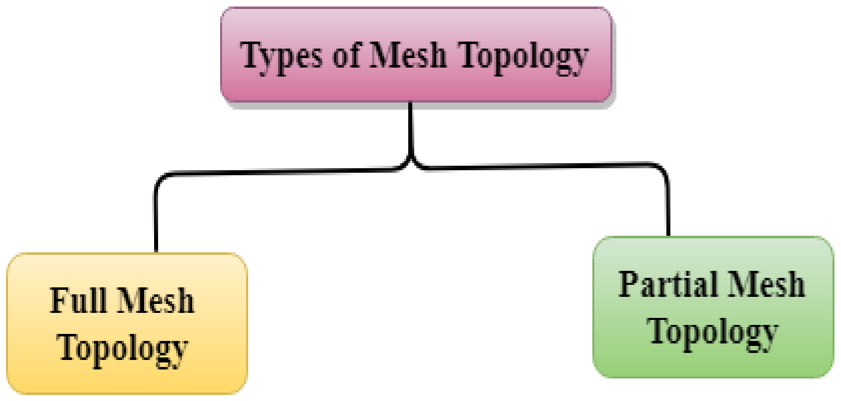


- Mesh technology is an arrangement of the network in which computers are interconnected with each other through various redundant connections.
- There are multiple paths from one computer to another computer.
- It does not contain the switch, hub or any central computer which acts as a central point 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.

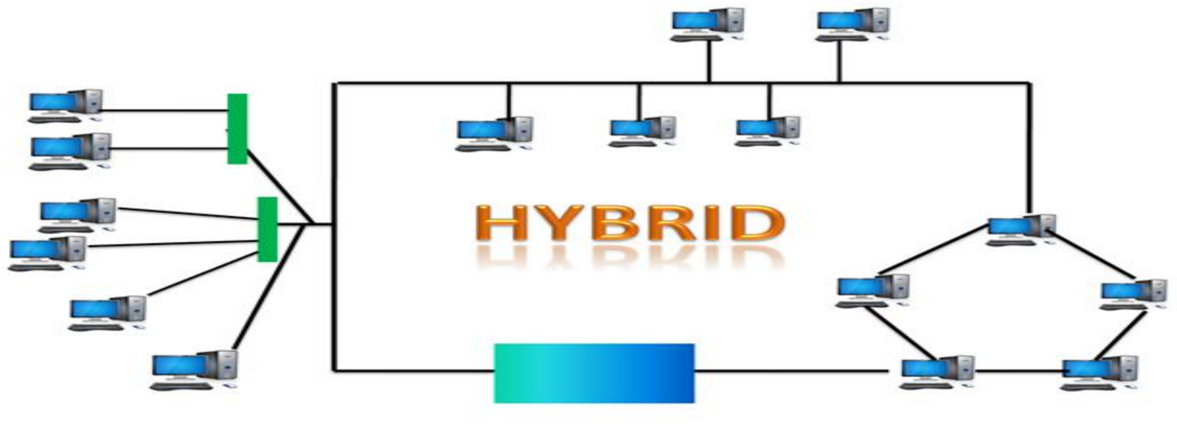
Fast Communication: Communication is very fast between the nodes.

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.

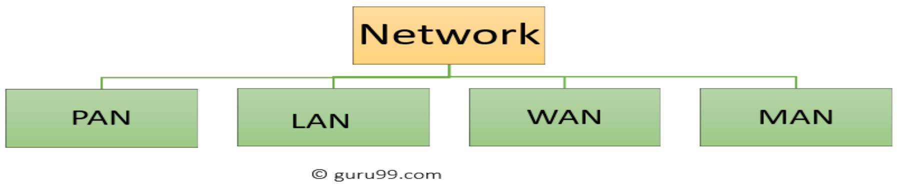
Disadvantages of Hybrid topology

- **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:

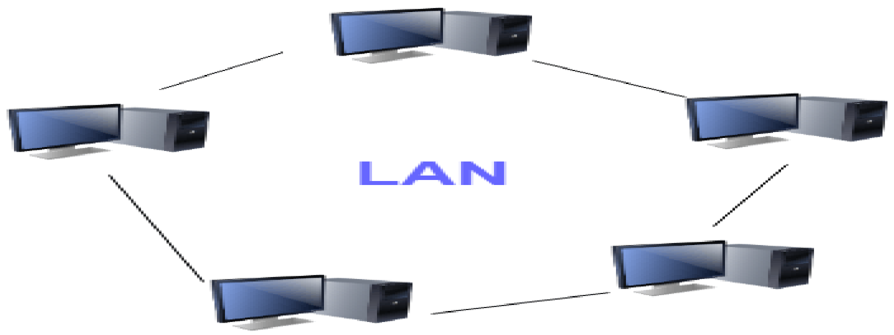
- PAN networks are relatively secure and safe
- It offers only short-range solution up to ten meters
- Strictly restricted to a small area

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.

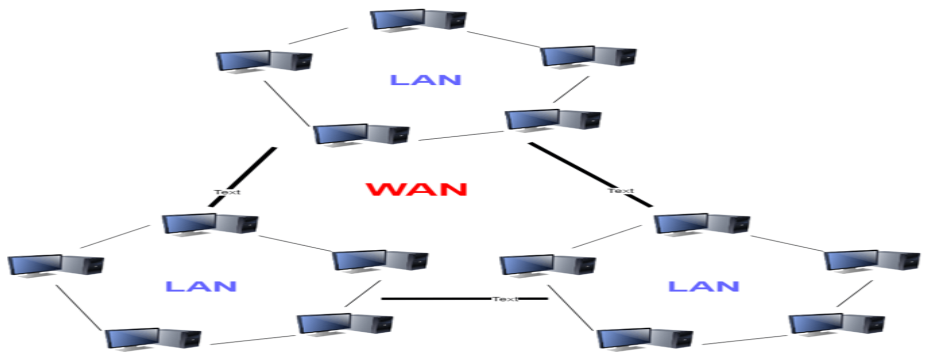
- It will be easy to manage data at only one place, which makes data more secure.
- Local Area Network offers the facility to share a single internet connection among all the LAN users.

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:

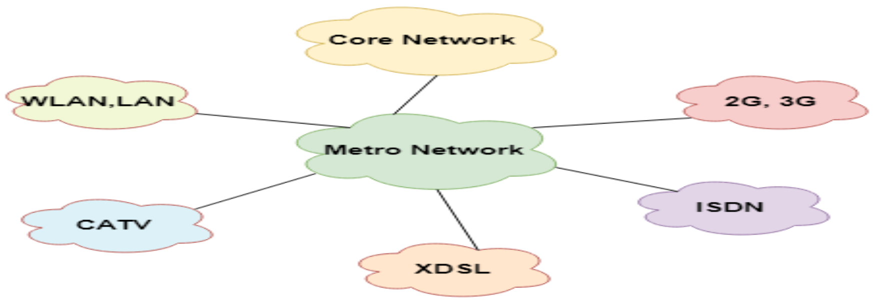
- WAN helps you to cover a larger geographical area. Therefore business offices situated at longer distances can easily communicate.
- Contains devices like mobile phones, laptop, tablet, computers, gaming consoles, etc.
- WLAN connections work using radio transmitters and receivers built into client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

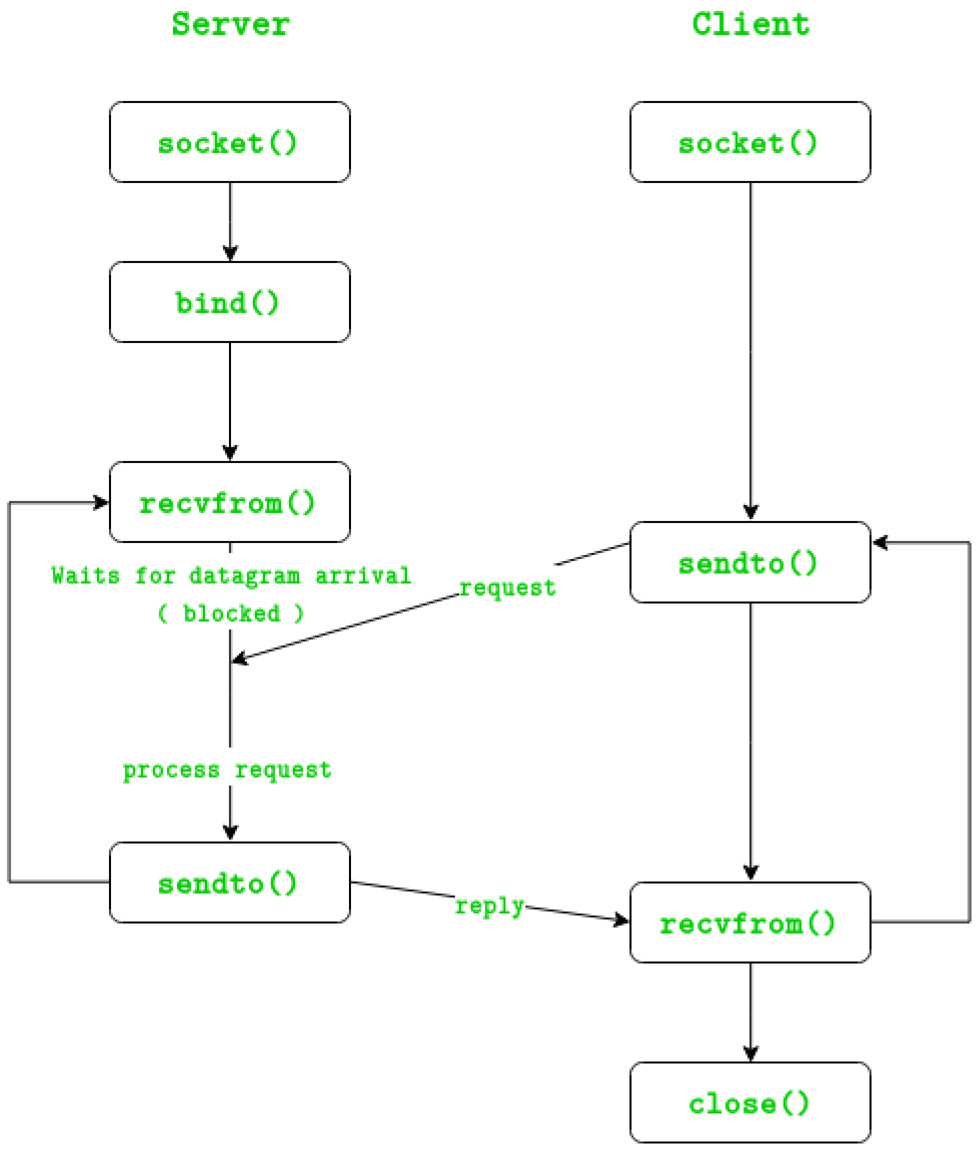
/* CLIENT PROGRAM FOR UDP CONNECTION */

```
#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);
        }
```

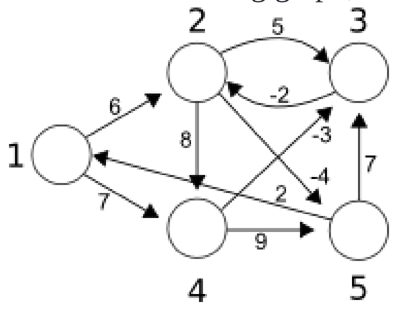
Experiment No.: 9

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.

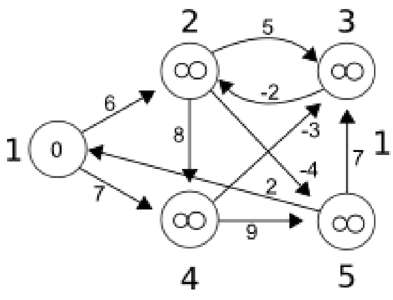
That said, you usually calculate **the shortest path to all nodes** and then pick the ones you're intrested in.

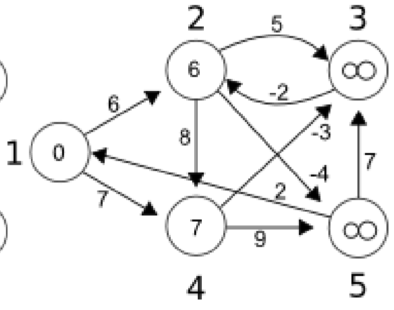
The Algorithm

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*:

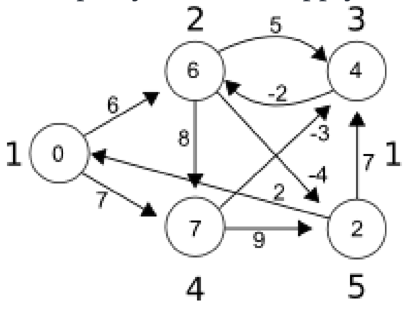


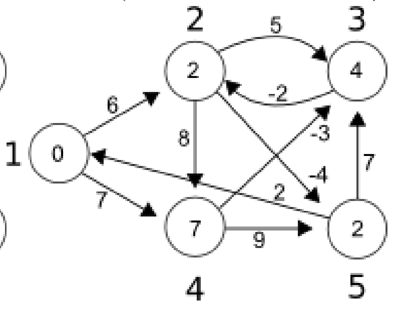


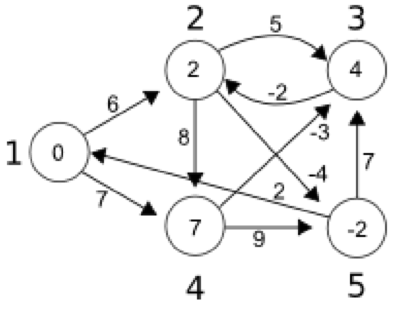
Take every edge and try to *relax* it:

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
```