# 1. Understanding and usage of Networking commands

## PING command:

**Ping stands for Packet Internet Gropher.**

**Ping** is a [computer network](http://en.wikipedia.org/wiki/Computer_network) administration utility used to test the reachability of a [host](http://en.wikipedia.org/wiki/Host_(network)) on an [Internet](http://en.wikipedia.org/wiki/Internet_Protocol) [Protocol](http://en.wikipedia.org/wiki/Internet_Protocol) (IP) network and to measure the [round-trip time](http://en.wikipedia.org/wiki/Round-trip_time) for messages sent from the originating host to a destination computer.

*Ping* operates by sending [Internet Control Message Protocol](http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) (ICMP) *echo request* [packets](http://en.wikipedia.org/wiki/Packet_(information_technology)) to the target host and waiting for an ICMP response.

## SYNTAX

**Ping –option host-address**

Host-address : this is the address of the host which we want to test reachability

-t option : used to send echo packets continuously until we stop. Ex: Ping –t host-address

-a option : used to resolve addresses to hostnames.

-n count: used to send count times echo requests.

-f option: used to set the flag of don’t fragment. If this is used packet will not fragment.

-i TTL: To spcify the packet life time

Ex: # ping –n 5 [www.example.com](http://www.example.com/)

PING [www.example.com](http://www.example.com/) (192.0.43.10) 56(84) bytes of data.

64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=1 ttl=250 time=80.5 ms 64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=2 ttl=250 time=80.4 ms 64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=3 ttl=250 time=80.3 ms 64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=4 ttl=250 time=80.3 ms 64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=5 ttl=250 time=80.4 ms

--- [www.example.com](http://www.example.com/) ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 4006ms rtt min/avg/max/mdev = 80.393/80.444/80.521/0.187 ms

## NETSTAT command:

**netstat** (**net**work **stat**istics) is a [command-line](http://en.wikipedia.org/wiki/Command_line_interface) [tool](http://en.wikipedia.org/wiki/Computer_software) that displays [network connections](http://en.wikipedia.org/wiki/Transmission_Control_Protocol) (both incoming and outgoing), routing tables, and a number of network interface statistics. It is available on [Unix](http://en.wikipedia.org/wiki/Unix), [Unix-like](http://en.wikipedia.org/wiki/Unix-like), and [Windows NT](http://en.wikipedia.org/wiki/Windows_NT)-based [operating systems](http://en.wikipedia.org/wiki/Operating_systems).

It is used for finding problems in the network and to determine the amount of traffic on the network as a performance measurement.

## Syntax:

*netstat [-a] [-n] [-p][-i][-r]*

|  |  |
| --- | --- |
| -a | Show the state of all sockets and all routing table entries; normally, sockets used by server processes are not shown and only interface, host, network, and default routes are  shown. |
| -n | Show network addresses as numbers. netstat normally displays addresses as symbols.  This option may be used with any of the display formats. |
| -p | Show the address resolution (ARP) tables. |
| -i | Show the state of the interfaces that are used for TCP/IP traffic. |
| -r | Show the routing tables. |

Example: C:\Users\stanleycsehod>netstat Active Connections

Proto Local Address Foreign Address State

TCP 192.168.5.16:49171 program:http CLOSE\_WAIT

|  |  |  |
| --- | --- | --- |
| TCP | 192.168.5.16:49172 | cdn-87-248-221-254:http CLOSE\_WAIT |
| TCP | 192.168.5.16:49173 | cdn-87-248-221-254:http CLOSE\_WAIT |
| TCP | 192.168.5.16:49174 | program:http CLOSE\_WAIT |

C:\Users\stanleycsehod>netstat -a Active Connections

|  |  |  |  |
| --- | --- | --- | --- |
| Proto | Local Address | Foreign Address | State |
| TCP | 0.0.0.0:135 | csehod:0 | LISTENING |
| TCP | 0.0.0.0:445 | csehod:0 | LISTENING |
| TCP | 0.0.0.0:5357 | csehod:0 | LISTENING |
| TCP | 0.0.0.0:8080 | csehod:0 | LISTENING |

1. IfConfig

**ifconfig** (short for interface configuration) is a system administration utility in [Unix-like](http://en.wikipedia.org/wiki/Unix-like) operating systems to configure, control, and query [TCP/IP](http://en.wikipedia.org/wiki/TCP/IP) [network interface](http://en.wikipedia.org/wiki/Network_card) parameters from a [command line](http://en.wikipedia.org/wiki/Command_line_interface) [interface](http://en.wikipedia.org/wiki/Command_line_interface) (CLI) or in system configuration scripts.

The "ifconfig" command allows the operating system to setup network interfaces and allow the user to view information about the configured network interfaces.

Common uses for ifconfig include setting an interface's [IP address](http://en.wikipedia.org/wiki/IP_address) and [netmask](http://en.wikipedia.org/wiki/Netmask), and disabling or enabling a given interface.[[1]](http://en.wikipedia.org/wiki/Ifconfig#cite_note-0) At boot time, many UNIX-like operating systems initialize their network interfaces with shell-scripts that call ifconfig. As an interactive tool, system administrators routinely use the utility to display and analyze network interface parameters.

USAGE for windows:

ipconfig [/all/renew/release/flushdns/displaydns/registerdns]

/all Display full configuration information.

/release Release the IPv4 address for the specified adapter.

/renew Renew the IPv4 address for the specified adapter.

/flushdns Purges the DNS Resolver cache.

/registerdns Refreshes all DHCP leases and re-registers DNS names

/displaydns Display the contents of the DNS Resolver Cache.

C:\Users\stanleycsehod>ipconfig Windows IP Configuration

Ethernet adapter Local Area Connection:

Connection-specific DNS Suffix . :

Link-local IPv6 Address : fe80::2141:9753:d0d5:5032%11

IPv4 Address. : 192.168.5.16

Subnet Mask : 255.255.255.0

Default Gateway : 192.168.5.1

Tunnel adapter isatap.{7BA6B200-CB41-408C-8160-37E3AFAD2C4F}: Media State Media disconnected

Connection-specific DNS Suffix . :

Tunnel adapter Local Area Connection\* 9:

Connection-specific DNS Suffix . :

IPv6 Address. : 2001:0:4137:9e76:24ab:2cd9:3f57:faef

Link-local IPv6 Address : fe80::24ab:2cd9:3f57:faef%13

Default Gateway : ::

1. **ARP command:**

ARP stands for Address Resolution Protocol. This protocol is used by network nodes to match IP addresses to MAC addresses.

Displays and modifies entries in the Address Resolution Protocol (ARP) cache, which contains one or more tables that are used to store IP addresses and their resolved Ethernet or Token Ring physical addresses. There is a separate table for each Ethernet or Token Ring network adapter installed on your computer. Used without parameters, **arp** displays help.

You can use the **arp** command to view and modify the ARP table entries on the local computer

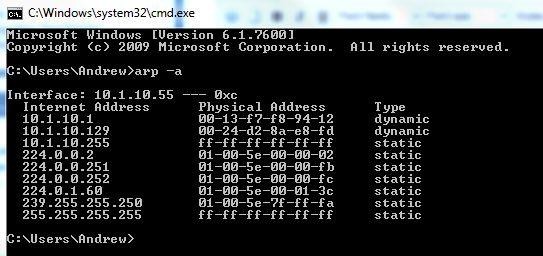
*Syntax (Inet means Internet address)*

**arp** –option options are :

1. **–a :** Displays current ARP cache tables for all interfaces. To display the ARP cache entry for a specific IP address, use **arp -a** with the *InetAddr* parameter, where *InetAddr* is an IP address**-**
2. **-d *InetAddr* :** Deletes an entry with a specific IP address, where *InetAddr* is the IP address. To delete all entries, use the asterisk (\*) wildcard character in place of *InetAddr*.
3. **-s *InetAddr EtherAddr* [*IfaceAddr*] :** Adds a static entry to the ARP cache that resolves the IP address *InetAddr*

to the physical address *EtherAddr*.

1. **/? :** Displays help at the command prompt.

To run the arp command in

Windows click

START> RUN> CMD.

1. **telnet command:**

The **telnet** commands allow you to communicate with a remote computer that is using the Telnet protocol.

The Telnet program runs on your computer and connects your PC to a [server](http://www.webopedia.com/TERM/S/server.html) on the network. You can then enter [commands](http://www.webopedia.com/TERM/C/command.html) through the Telnet program and they will be executed as if you were entering them directly on the server [console](http://www.webopedia.com/TERM/C/console.html). This enables you to control the server and communicate with other servers on the network. To start a Telnet session, you must log in to a server by entering a valid [username](http://www.webopedia.com/TERM/U/username.html) and [password.](http://www.webopedia.com/TERM/P/password.html) Telnet is a common way to [remotely control](http://www.webopedia.com/TERM/R/remote_control.html) [Web servers](http://www.webopedia.com/TERM/W/Web_server.html).

Syntax

*telnet –options remotehost-address] options are:*

|  |  |
| --- | --- |
| -8 | Specifies an 8-bit data path. Negotiating the TELNET BINARY option is attempted for both input and output. |
| -E | Stops any character from being recognized as an escape character. |
| -L | Specifies an 8-bit data path on output. This causes the BINARY option to be negotiated on output. |
| -c | Disables the reading of the user's telnetrc file. |
| -d | Sets the initial value of the debug toggle to TRUE. |
| -r | Specifies a user interface similar to rlogin . In this mode, the escape character is set to the  tilde (~) character, unless modified by the -e option. The rlogin escape character is only recognized when it is preceded by a carriage return. In this mode, the telnet escape character, normally '^]', must still precede a telnet command. The rlogin escape character  can also be followed by '.\r' or '^Z', and, like [rlogin](http://www.computerhope.com/unix/urlogin.htm), closes or suspends the connection, respectively. This option is an uncommitted inter- face and may change in the future. |
| -e escape\_char | Sets the initial escape character to escape\_char. escape\_char may also be a two character sequence consisting of '^' followed by one character. If the second character is '?', the DEL character is selected. Otherwise, the second character is converted to a control character and used as the escape character. If the escape character is the null string (that is, -e ''), it is disabled. |
| -l user | When connecting to a remote system that understands the ENVIRON option, then user will be sent to the remote system as the value for the ENVIRON variable USER. |
| -n file | Opens tracefile for recording trace information. |

Examples

**telnet host.com** - Would open a telnet session to the domain host.com

1. **FTP command:**

**File Transfer Protocol** (**FTP**) is a standard [network protocol](http://en.wikipedia.org/wiki/Network_protocol) used to transfer files from one host to another over a [TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol)-based network, such as the [Internet](http://en.wikipedia.org/wiki/Internet). FTP is built on a [client-server](http://en.wikipedia.org/wiki/Client-server_model) architecture and utilizes separate control and data connections between the client and server.

Syntax:

FTP remote-host-ip-address Username:dasdadas Password:daasa

Connected to host.

ftp> get filename : gets the file from the remote host to local host

ftp> put filename : sends/uploads the specified file from the local host to remote host ftp > dir : list out the files of a remote host directory

ftp > ls : list out the files of a remote host directory ftp > lcd : list out the files of a local host directory

ftp >rename file1, file 2: renames the file1 with file2 of a remote host. ftp>delete filename : deletes the specified file from the remote host

ftp > pwd : to check the present working directory of the remote host ftp> mkdir directory-name : to create a directory at the remote host ftp > cd directory name : to change directory at the remote host

1. **Tftp command:**
   * **Trivial File Transfer Protocol** (**TFTP**) is a [file](http://en.wikipedia.org/wiki/Computer_file) transfer [protocol](http://en.wikipedia.org/wiki/Network_protocol) known for its simplici- ty.[*citation needed*] It is generally used for automated transfer of configuration or boot files be- tween machines in a local environment. Compared to [FTP](http://en.wikipedia.org/wiki/File_Transfer_Protocol), TFTP is extremely limited, providing no authentication, and is rarely used interactively by a user.
   * Due to its simple design, TFTP could be implemented using a very small amount of [memory.](http://en.wikipedia.org/wiki/Computer_memory) It is therefore useful for [booting](http://en.wikipedia.org/wiki/Network_booting) computers such as [routers](http://en.wikipedia.org/wiki/Router) which may not have any [data storage devices](http://en.wikipedia.org/wiki/Data_storage_device)
   * It is also used to transfer small amounts of data between hosts on a [network](http://en.wikipedia.org/wiki/Computer_network), such as [IP](http://en.wikipedia.org/wiki/IP_phone) [phone](http://en.wikipedia.org/wiki/IP_phone) firmware or operating system images when a remote [X Window System](http://en.wikipedia.org/wiki/X_Window_System) [terminal](http://en.wikipedia.org/wiki/Computer_terminal) or any other [thin client](http://en.wikipedia.org/wiki/Thin_client) boots from a network host or [server.](http://en.wikipedia.org/wiki/Server_(computing))
   * It has been implemented on top of the [User Datagram Protocol](http://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP) using port number 69.
   * TFTP typically uses [UDP](http://en.wikipedia.org/wiki/User_datagram_protocol) as its [transport protocol](http://en.wikipedia.org/wiki/Transport_layer), but it is not a requirement.
2. ***Socket programming using UDP and TCP (e.g., simple DNS, date & time client/server, echo client/server, iterative & concurrent servers)***

# Implement a simple DNS

#include <netdb.h> #include <sys/types.h> #include <netinet/in.h> #include<sys/socket.h> #include<arpa/inet.h>

int main(int argc,char \*argv[])

{

struct hostent \*h; if(argc!=2)

{ /\*error check the command line\*/ printf(stderr,"usage:getpid address\n"); exit(1);

}

if((h=gethostbyname(argv[1]))==NULL)

{ /\*get the host info\*/ perror("gethostbyname"); exit(1);

}

printf("Host name : %s\n",h->h\_name);

printf("IP Address : %s\n",inet\_ntoa(\*((struct in\_addr \*)h->h\_addr)));

return 0;

}

Output:

student@student-Veriton-Series:~/Sowjanya$ vi DNS.c student@student-Veriton-Series:~/Sowjanya$ cc DNS.c -o dns student@student-Veriton-Series:~/Sowjanya$ ./dns [www.google.com](http://www.google.com/) Hostname is [www.google.com](http://www.google.com/)

IP address is 142.250.195.68

student@student-Veriton-Series:~/Sowjanya$ ./dns [www.facebook.com](http://www.facebook.com/) Hostname is [www.facebook.com](http://www.facebook.com/)

IP address is 157.240.16.35

student@student-Veriton-Series:~/Sowjanya$ ./dns [www.methodist.edu.in](http://www.methodist.edu.in/) Hostname is methodist.edu.in

IP address is 103.148.157.253

# IP, TCP and UDP

As the last panel indicated, when program a sockets application, have a choice to make between using TCP and using UDP. Each has its own benefits and disadvantages.

* TCP is a stream protocol, while UDP is a datagram protocol. TCP establishes a continuous open con- nection between a client and a server, over which bytes may be written and correct order guaran- teed for the life of the connection. Hover, bytes written over TCP have no built-in structure, so higher- level protocols are required to delimit any data records and fields within the transmitted byte stream.
* UDP, on the other hand, does not require that any connection be established between client and server; it simply transmits a message between addresses. A nice feature of UDP is that its packets are self- delimiting--each datagram indicates exactly where it begins and ends. A possible disadvantage of UDP, hover, is that it provides no guarantee that packets will arrive in-order, or even at all. Higher-level protocols built on top of UDP may, of course, provide handshaking and acknowledgements.
* A useful analogy for understanding the difference between TCP and UDP is the difference between a telephone call and posted letters. The telephone call is not active until the caller "rings" the receiver and the receiver picks up. The telephone channel remains alive as long as the parties stay on the call--but they are free to say as much or as little as they wish to during the call.

# Peers, ports, names, and addresses

Beyond the protocol--TCP or UDP--there are two things a peer (a client or server) needs to know about the machine it communicates with: An IP address and a port. An IP address is a 32-bit data value, usually represented for humans in "dotted quad" notation, e.g., 64.41.64.172. A port is a 16-bit data value, usually simply represented as a number less than 65536--most often one in the tens or hundreds range. An IP address gets a packet *to* a machine, a port lets the machine decide which process/service (if any) to direct it to.

# Client/Server Communication

* + Most network applications can be divided into two pieces: a client and a server.
  + server is a process that is waiting for a client process, receives the request from the client, processes it and sends the response back to client.
  + A Client is a process who sends requests to the server.
  + The server process starts first on any system and waits for the client requests.
  + A client process initiated by the user by interacting with it.
  + There can be standard services which runs on standard/reserved ports or user defined services implemented by the user.
  + Examples of standard or predefined services are Time-of day service, Ping service, FTP service, Echo service etc.
  + Servers can be classified based on
    1. Their servicing discipline (iterative or concurrent);
    2. Their communication methods (connection-oriented or connectionless);
    3. Their information that they keep (stateless or stateful).

**Iterative Server**

* + An iterative server serves the requests one after the other.
  + It is easier to design and implement.
  + Iterative design is suitable when the service time for eachrequest is small (because the mean response time is stillacceptably small).
  + It is suitable for simple services such as the TIMEservice. Iterative design is not suitable when the service time for arequest may be large.

## Example

* + Two clients are using a file transfer service:
    - the 1st client requests to get a file of size 200 Mbytes,
    - the 2nd client requests to get a file of size 20 bytes.
  + If iterative design is used, the 2nd client has to wait a long time.

## Concurrent Server

* + A concurrent server serves multiple clients simultaneously.
  + Concurrency refers to either
* real simultaneous computing (using multiple processors)
* apparent simultaneous computing (by time sharing).Concurrent design is more difficult to implement, but it can give a smaller response time when
* Different clients require very different service time, or the service requires significant I/O, or the server is executed on a computer with multiple processors.

**Connection-Oriented Servers**

* + A connection-oriented server uses TCP for connection-oriented communication.
  + TCP provides reliable transport. The servers need not do so. The servers are simpler.
  + A server uses a separate socket for each connection.

**Connectionless Servers**

* + A connectionless server uses UDP for connectionless communication.
  + UDP does not guarantee reliable transport.
  + A connectionless server may need to realize reliability through timeout and retransmission.
  + The server and client are relatively complicated.

**Study of elementary and advanced socket system calls**

**Sockets:**

* Sockets are a form of IPC provided by 4.3 BSD.
* Sockets are used for communication between the processes on a single system and between processes on different systems.
* Unix domain sockets are used for IPC between processes on a single system.
* Examples of Unix domain sockets are pipes, FIFOs, named pipes etc. Socket Addresses:
* This structure is in <sys/socket.h> header file.
* The generic structure for any family is : Struct sockaddr

{

u\_short sa\_family; char sa\_data[14];

};

* For the internet family, the following structures are used :

Struct in\_addr

{

u\_long s\_addr;//32-bit ip address

};

struct sockaddr\_in

{

short sin\_family; //protocol family u\_short sin\_port;//port number struct in\_addr sin\_addr;//in\_addr

//structure member

char sin\_zero[8];//unused member

};

* Elementary system calls are the systems calls which are used in basic communication.
* They are socket(),listen(),bind(),accept(),send,receive() system calls.

1. **Socket():** Used for creating the end point for establishing the connections.
   * Syntax of socket system call:

int socket(int family, int socket\_type, int protocol)

-family : specifies which protocol family are using for this socket.

AF\_INET for Internet family AF\_UNIX unix internal protocols AF\_NS xerox NS protocols

Note: AF stands for Address family PF stands for Protocol family

socket-type: Type of socket ex: connection oriented (SOCK\_STREAM), connection-less(SOCK- DGRAM)

SOCK\_RAW,SOCK\_SEQPACKET,SOCK\_RDM(Reliable delivery message)

- protocol used for this socket IPPROTO\_TCP for TCP protocol IPPROTO\_UDP for UDP protocol.

- This system call returns integer which is used as a socket descriptor, returns -1 if it can not create socket.

\* Socket system call fills protocol tuple of the 5-tuple association.

1. Bind system call: This system call assigns a name to an unnamed socket. Syntax: int bind(int sockfd, struct sockaddr \*myaddr, int addrlen);

-second argument is a pointer to a protocol specific address.

-third argument is a length of a address structure.

This call fills in the local-address and local-process elements of a 5-tuple association.

5-tuple association is{protocol,local-address,local-process,foreign-address,foreign-process}. Uses of bind:

* + Servers register their address so that clients can communicate with them.
  + A client can register a specific address for itself.
  + A connection-less client needs to assure that the system assigns it some unique address, so that the other end has a valid return address to send responses.

1. Connect system call: A client process sends the connect request to a server with this system call. This is used to establish a connection with server.

Syntax: int connect(int sockfd, struct sockaddr\*servaddr, int addrlen);

-Sockfd is a socket descriptor.

-Second and third arguments are server address and length of it.

This system call establishes actual connection between the local system and the foreign system This system call is blocked until the connection is established.

If server is not ready , this returns error(i.e -1).

* + Client does not have to bind a local address before calling connect.
  + Connection initializes 4 tuples of the 5-tuple association.They are local-address,local- process,foreign-address and foreign-process.
  + A connectionless client also can use the connect system call.
  + In a connection-less protocol, connect system call is to store the server-address specified by the process, so that the system can communicate with it.
  + One advantage of connecting to a socket with a connection-less protocol is that we don’t need to specify the destination address for every datagram.

1. Listen system call:

This system call is used by the server for connection-oriented service to indicate that it is willing to receive connections.

Syntax: int listen(int sockfd, int backlog)

-First argument is socket id

-second argument specifies how many connection requests can be queued by the system while it waits for the server to execute the accept system call. Maximum is 5.

1. Accept system call: When client requests for the connection, server executes accept system call and establishes the connection.
   * This system call returns connection id i.e. virtual circuit id, which is later used for data transfer.
   * Syntax: Int accept(int sockfd, struct sockaddr \*peer, int \*peerlen);
   * First argument is socket id on which server is waiting for clients.
   * Second argument is empty structure, which is filled with the client details like ip-address and port no used by client taken from the connection request.
   * Third argument is length of the structure.
   * Accept takes the first connection request on the queue and creates another socket with the same properties as sockfd.
   * If there are no connection requests, this call blocks until requests arrives.
   * This system call returns upto three values:
   * An integer value which is either error indication or a new socket descriptor. -1 indicates error.
   * The address of the client(peer) process.
   * The size of this address.
2. Send & Recv system calls: These system calls are used to send and reveive data over the connec- tion-oriented service.

Syntaxes: Int send(int sockfd, char \*buff, int nbytes, int flags); Int recv(int sockfd, char \*buff, int nbytes, int flags);

-first argument is the socket id.

-second argument is the message which has to be send

-Third argument specifies the length of the message sending

-fourth argument is about flags, it may be 0 when not used or it can be one of the following: MSG\_OOB send /receive out-of-band data

MSG\_PEEK peek at incoming message MSG\_DONTROUTE bypass routing

-These system calls returns the length of data that was written or read.

1. Sendto & recvfrom system calls: These system calls are used for sending and receiving data over connection less service i.e using UDP protocol.

Syntaxes:

int sendto(int sockfd, char \*buf, int nbytes, int flags, struct sockaddr \*to, int addrlen);

int recvfrom(int sockfd, char \*buf, int nbytes, int flags, struct sockaddr \*from, int addrlen);

* + The first four arguments are same as send and recv system calls.
  + Fifth argument is address of the peer entity.
  + Sixth argument is length of the address structure.

1. Close system call: This is used to close a socket. Syntax: int close(int fd);

where fd is a file/socket descriptor.

Before closing the socket kernel ensures that any data within the kernel that still has to be transmit- ted or acknowledged.

Normally the system returns from the close immediately, but the kernel still tries to send or re- ceive.

Byte ordering routines:

* + The following four functions handle byte order differences between different computer architec- tures and different network protocols.
  + These required two header files:

<sys/types.h> and <netinet/in.h> Syntaxes of these functions:

u\_long htonl(u\_long hostlong); u\_short htons(u\_short hostshort); u\_long ntohl(u\_long netlong); u\_short ntohs(u\_short netshort);

* + htonl – convert host to network, long integer
  + htons – convert host to network, short integer
  + ntohl – convert network to host, long integer
  + ntohs – convert network to host, short integer

\*These four operations operate on unsigned integer values. Short integer means 16 bit data

Long integer means 32 bit data.

System V has following functions for byte operations:

memcpy – to copy n bytes from source to destination

memcmp – to compare nbytes of first string with n bytes of another string memset – to initialize the given string with the given value.

These require **string.h** header file.

void \*memset(void \*dest, int c, size\_t len);

void \*memcpy(void \*dest, const void \*src, size\_t nbytes); Int memcmp(const void \*ptr1, const void \*ptr2, size\_t n bytes) Address Conversion Routines:

* + Internet address is in dotted decimal format.
  + The following functions convert between the dotted decimal format and in\_addr structure.
  + This requires the following header file: <arpa/inet.h>
  + These functions are:

1. **unsigned long inet\_addr(char \*ptr)**; //converts given character string in dotted decimal notation to a 32 bit internet address.
2. char \* **inet\_ntoa(struct in\_addr inaddr)**; //converts given internet adress in dotted decimal notation to a character string.
3. INADDR\_ANY : This is a constant, if we specify address as INADDR\_ANY in bind system call, this tells the system that we will accept a connection on any Internet interface on the system, if the system is multihomed.

Advanced Socket system calls:

* + Readv and writev System calls
  + Sendmsg and recvmsg system calls
  + Getpeername,getsockname system calls
  + Getsockopt and setsockopt system calls
  + Shutdown system call
  + Select system call

1. Readv & writev system calls
   * These are called scatter read and gather write.
   * These are similar to read write system calls for reading & writing.
   * These two variants provide the ability to read into or write from noncontiguous buffers.
   * This requires <sys/types.h> & <sys/uio.h> header files.
   * Syntaxes:

## int writev(int fd, struct iovec iov[], int iovcount);

**int readv(int fd, struct iovec iov[], int iovcount);**

* + First argument is any io descriptor
  + Second argument is iovec structure argument array which holds data to write
  + Third argument is count of no. of buffers.
  + The writev system call writes the buffers specified by iov[0], through iov[iovcount-1].
  + The readv does fills one buffer before preceeding to next buffer in iov array.
  + Both returns no.of bytes read or written.
  + Structure of iovec:

## Struct iovec

* + **{**

## caddr\_t iov\_base;//starting address of buffer

* + **int iov\_len; //size of buffer in bytes**

## }

* + readv & writev can be used for any descriptor, not only for sockets.
  + Multiple writes can be done with a single writev.

1. Sendmsg & recvmsg system calls
   * These are used for sending and receiving messages from only socket descriptors.
   * Syntax:

Int sendmsg(int sockfd, struct msghdr msg[],int flags); Int recvmsg(int sockfd, struct msghdr msg[], int flags);

* + These system calls are used for only socket descriptor.
  + These system calls uses scatter read and scatter write concept.
  + This provides access rights between processes.
  + Message structure is:
  + Struct msghdr {

caddr\_t msg\_name;//optional address int msg\_namelen;//size of address

struct iovec \*msg\_iov;//scatter/gather array int msg\_iovlen; //array size

caddr\_t msg\_accessrights;//access rights for sent/recvd int msg\_accrightslen;

}

* + -msg-name & msg-len are used when socket is not connected
  + -iovec structure is similar to writev & readv.
  + - msg\_accessrights deal with the passing & receiving of access rights between processes.

1. Read & write system call variants:

System calls Descriptor read/write

|  |  |  |
| --- | --- | --- |
| Read&write | any | single buffer |
| Readv/writev | Any | scatter/Gather R/W |

Recv/send Socket single buffer Recvfrom/sendto Socket single buffer Recvmsg/sendmsg Socket Scatter/gather r/w

1. Get peer name system cal
2. This system call returns the name of the peer process that is connected to a given socket.
3. Syntax:

Int getpeername(int sockfd, struct sockaddr \*peer, int \*addrlen);

-This returns the foreign-addr and foreign-process elements of the 5-tuple association.

-last argument is value-result argument.

This system call should be invoked after the connection is established. l

1. Getsockname system call

This system call returns the name associated with a socket.

Int getsockname(int sockfd, struct sockaddr \*peer, int \*addrlen);

-this returns the local-addr and local-process elements of a 5-tuple association.

-This can be invoked after the socket is binded or connected.

1. Select system call
   * This system call is used for I/O Multiplexing.
   * This system call can be used when dealing with multiple descriptors.
   * This function allows the process to instruct the kernel to wait for any one of the multiple events to occur & to wake up the process only when one or morre these events occurs or when a specified amount of time has passed.
   * Ex: we can inform kernel to notify only when

* one of the descriptors in the set {1,4,5} are ready for reading
* One of the descriptors in the set {2,7} are ready for writing
* Any of the descriptor in the set {1,4} have an exception condition pending.
  + Descriptors are not restricted to sockets. Any descriptor can be used for select.

Int select(int maxfdpl,fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval

\*timeout);

* + Time val structure is:

Struct timeval

{

long tv\_sec;//seconds

long tv\_usec; //microseconds

}

* + This uses following macros:

1. FD\_ZERO(fd\_set \*set): to clear all bits in the set
2. FD\_SET(int fd, fd\_set \*set):turn the fd bit on
3. FD\_CLR(int fd, fd\_set \*set):clear the fd bit
4. FD\_ISSET(int fd, fd\_set \*set):test the fd bit in the set
   * Timeout is used to specify how much time has to wait when any one of the descriptor is ready for I/O.
   * There are three possibilities with the time argument:
5. Wait forever: return only when one of the descriptor is ready for I/O. For this specify Timeout ar- gument as NULL.
6. Wait up to a certain time for checking the descriptors ready or not. For this 4th argument is Time val structure.
7. Do not wait at all, return immediately after checking the descriptors. This is called polling. For this Time should be zero.

9. Setsockopt & getsockopt system call:

These two system calls are used for setting options to socket and getting the socket options. Syntaxes:

Int getsockopt(int sockfd, int level, int optname, char \*optval, int \*optlen); Int setsockopt(int sockfd, int level, int optname, char \*optval, int optlen);

-level specifies options for socket or TCP or IP.

-optname specifies option name which is setting/getting

-optval is the value setting for option or getting value into this when we call getsockopt.

* + Socket options are:

Level

SOL\_SOCKET

Option name:

SO\_BROADCAST : Enable /disable broadcasting SO\_DEBUG : enables/disables low level debugging SO\_DONTROUTE : To bypass normal routing

SO\_ERROR : returnes to the caller the contents of the variable so\_error. This is only getting .can not set. SO\_KEEPALIVE: get/set how long connection can be alive.

SO\_SNDBUF to set/get the send buffer size SO\_RCVBUF: to set/get the receive buffer size SO\_SNDTIMEO:to set/get the sender timeout SO\_RCVTIMEO : to set/get the receiver timeout.

SO\_REUSEADDR: enables/disables to reuse the local addresses and local processes of 5-tuple association SO\_TYPE: gets the socket type. Not for setting.

* + Level OptionName IPPROTO\_TCP TCP\_MAXSEG

TCP\_NODELAY

IPPROTO\_IP IP\_OPTIONS

# Implementation of Iterative Daytime server using Connection-Oriented(TCP).

**Description:** This program implements server as a iterative Daytime server, which sends the day and time to the client . This uses the connection oriented concept for implementing, which means first it establishes the connection, uses the connection for communication and closes it after the communication is ended. This uses TCP protocol.

## Server side :

* 1. First creates the socket using socket system call.
  2. Binds the name (i.e. IP address, port number and family) of the server to the socket.
  3. Makes the socket to be ready in passive state by creating queues for storing the client requests using listen system call. In this server is listening for the clients.
  4. Once the connect request comes from the client, it stores in the queue and calls accept system call for establishing the connection. This returns the new connection id, which is used while sending and receiving the messages.
  5. Extracts the system time using time and ctime system call.
  6. Sends the day and time to the client.
  7. Close the communication.

## Client side :

1. First creates the socket.
2. Initializes the server address to sockaddress\_in structure to which connect request has to send.
3. Makes a connection request to the server using connection system call. If the connec- tion is established at the server it gets the connection id, which is used for the commu- nication. Otherwise it returns negative number as an error.
4. If connection is established, receives the day and time message from the server.

Server Program:

#include<netinet/in.h> #include<sys/socket.h> #include<stdio.h> #include<string.h> #include<time.h> #include<stdlib.h> #include<unistd.h>

int main( )

{

struct sockaddr\_in sa;

struct sockaddr\_in cli;int sockfd,conntfd;int len,ch;char str[100]; time\_t tick;

sockfd=socket(AF\_INET,SOCK\_STREAM,0); if(sockfd<0)

{

printf("error in socket\n"); exit(0);

}

else printf("Socket opened"); bzero(&sa,sizeof(sa)); sa.sin\_port=htons(5600); sa.sin\_addr.s\_addr=htonl(0);

if(bind(sockfd,(struct sockaddr\*)&sa,sizeof(sa))<0)

{

printf("Error in binding\n");

}

else

printf("Binded Successfully"); listen(sockfd,50);

for(;;)

{

len=sizeof(ch);

conntfd=accept(sockfd,(struct sockaddr\*)&cli,&len); printf("Accepted");

tick=time(NULL); snprintf(str,sizeof(str),"%s",ctime(&tick)); printf("%s",str);write(conntfd,str,100);

}

}

Server Output:

student@student-Veriton-Series:~/Sowjanya$ vi IterDayTimSerTCP.c student@student-Veriton-Series:~/Sowjanya$ cc IterDayTimSerTCP.c -o daytimeser student@student-Veriton-Series:~/Sowjanya$ ./daytimeser

Socket openedBinded SuccessfullyAcceptedSat Sep 4 11:57:49 2021

**Client Program:** #include<netinet/in.h> #include<sys/socket.h> #include<stdio.h> #include<stdlib.h> #include<unistd.h> #include<string.h>

int main()

{

struct sockaddr\_in sa,cli; int n,sockfd;

int len;char buff[100]; sockfd=socket(AF\_INET,SOCK\_STREAM,0); if(sockfd<0){ printf("\nError in Socket"); exit(0);

}

else printf("\nSocket is Opened"); bzero(&sa,sizeof(sa)); sa.sin\_family=AF\_INET; sa.sin\_port=htons(5600);

if(connect(sockfd,(struct sockaddr\*)&sa,sizeof(sa))<0)

{

printf("\nError in connection failed"); exit(0);

}

else

printf("\nconnected successfully"); if(n=read(sockfd,buff,sizeof(buff))<0)

{

printf("\nError in Reading"); exit(0);

}

else

{printf("\nMessage Read %s",buff);

}}

Client Output:

student@student-Veriton-Series:~/Sowjanya$ vi IterDayTimCliTCP.c student@student-Veriton-Series:~/Sowjanya$ cc IterDayTimCliTCP.c -o daytimecli student@student-Veriton-Series:~/Sowjanya$ ./daytimecli

Socket is Opened connected successfully

Message Read Sat Sep 4 11:57:49 2021 student@student-Veriton-Series:~/Sowjanya$

# Implementation of Iterative Daytime server using connectionless service(UDP).

**Description:** This program implements server as a iterative Daytime server, which sends the day and time to the client . This uses the connection less concept for implementing, which means it doesn’t establishes the connection. This uses UDP protocol.

## Server side :

* 1. First creates the socket using socket system call.
  2. Binds the name (i.e. IP address, port number and family) of the server to the socket.
  3. Server blocks for receiving the message from the client by using recv from system call.
  4. Extracts the time from the system and sends to the client.

## Client side :

1. First creates the socket.
2. Initializes the server address to sockaddress\_in structure to which connect request has to send.
3. Sends the message to server using send to system call by specifying server address along with the message.
4. Then waits for the day and time message from the server with recv from system call.

# Server Program:

#include<stdio.h> #include<stdlib.h> #include<time.h> #include<sys/types.h> #include<sys/socket.h> #include<netinet/in.h> #include<string.h> void errExit(char \*str)

{

puts(str); exit(0);

}

int main()

{

unsigned long t;

char \*st,buffer[512],MAX\_MSG; int n,i,l,ml,sockfd,newsockfd,cliLen;

struct sockaddr\_in serv\_addr,cli\_addr; sockfd=socket(AF\_INET,SOCK\_DGRAM,0); serv\_addr.sin\_family=AF\_INET; serv\_addr.sin\_addr.s\_addr=htonl(INADDR\_ANY); serv\_addr.sin\_port=htons(8000);

bind(sockfd,(struct sockaddr \*)&serv\_addr,sizeof(serv\_addr)); l=sizeof(cli\_addr);

cliLen = sizeof(cli\_addr);

n = recvfrom(sockfd,buffer,MAX\_MSG,0 ,(struct sockaddr\*)&cli\_addr,&cliLen);

if(n < 0)

{

errExit("recvfrom error \n");

}

printf("\nMessage received from client: %s", buffer); t=time(&t);

st=(char \*)ctime(&t); strcpy(buffer,st); n=sizeof(buffer);

if(sendto(sockfd,buffer, n,0,(struct sockaddr \*)&cli\_addr,cliLen) !=n)

{

errExit("sendto error \n");

}

printf ("Time sent…");

}

Server Output:

student@student-Veriton-Series:~/Sowjanya$ vi IterDayTimSerUDP.c

student@student-Veriton-Series:~/Sowjanya$ cc IterDayTimSerUDP.c -o daytimeserUDP

student@student-Veriton-Series:~/Sowjanya$ ./daytimeserUDP

Message received from client: Time sent…student@student-Veriton-

Series:~/Sowjanya$ vi IterEchoSerTCP.c student@student-Veriton-Series:~/Sowjanya$

# Client Program:

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

#define SRV\_IP\_ADRS "127.0.0.1"

#define SRV\_UDP\_PORT 8000

#define MAX\_MSG 100 void errExit(char \*str)

{

puts(str); exit(0);

}

int main( )

{

int sockFd;

struct sockaddr\_in srvAdr; char txmsg[MAX\_MSG]; char rxmsg[MAX\_MSG]; int n;

if((sockFd = socket (AF\_INET,SOCK\_DGRAM,0))< 0)

{ errExit("can't open datagram socket \n");

}

memset ( &srvAdr,0,sizeof(srvAdr)); srvAdr.sin\_family = AF\_INET;

srvAdr.sin\_addr.s\_addr = inet\_addr(SRV\_IP\_ADRS); srvAdr.sin\_port = htons (SRV\_UDP\_PORT); printf("Enter message to send :\n"); fgets(txmsg,MAX\_MSG,stdin);

n = strlen(txmsg)+1;

if(sendto(sockFd,txmsg,n,0,(struct sockaddr \*)&srvAdr,sizeof(srvAdr ))!=n)

{ errExit("sendto error \n");

}

n = recv(sockFd,rxmsg,MAX\_MSG,0); printf("n=%d\n",n);

if(n < 0)

{ errExit("recv error \n");

}

printf("Time Received from the server :%s\n",rxmsg);

}

Client Output:

tudent@student-Veriton-Series:~/Sowjanya$ vi IterDayTimCliTCP.c student@student-Veriton-Series:~/Sowjanya$ vi IterDayTimCliUDP.c student@student-Veriton-Series:~/Sowjanya$ cc IterDayTimCliUDP.c

student@student-Veriton-Series:~/Sowjanya$ cc IterDayTimCliUDP.c -o daytimecliUDP

student@student-Veriton-Series:~/Sowjanya$ ./daytimecliUDP Enter message to send :

DayTime n=100

Time Received from the server :Sat Sep 4 12:40:10 2021 student@student-Veriton-Series:~/Sowjanya$

# Implementation of iterative echo server using connection oriented socket system calls

**Description:** This program implements server as a iterative server, which receives the messages from the client and sends the same message as echo to the client. This uses the connection oriented concept for implementing, which means first it establishes the connection, uses the connection for communication and closes it after the communication is ended.

## Server side :

* + - 1. First creates the socket using socket system call.
      2. Binds the name (i.e. IP address, port number and family) of the server to the socket.
      3. Makes the socket to be ready in passive state by creating queues for storing the client requests using listen system call. In this server is listening for the clients.
      4. Once the connect request comes from the client, it stores in the queue and calls accept system call for establishing the connection. This returnes the new connection id, which is used while sending and receiving the messages.
      5. Receives the message from the client and echoes it back to the client.
      6. Close the system call after completing the communication.

## Client side :

1. First creates the socket.
2. Initializes the server address to sock address\_in structure to which connect request has to send.
3. Makes a connection request to the server using connection system call. If the connec- tion is established at the server it gets the connection id, which is used for the commu- nication. Otherwise it returns negative number as an error.
4. If connection is established, sends and receives the messages using send and recv sys- tem calls.

# Server Program:

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

#define SRV\_TCP\_PORT 5556

#define MAX\_MSG 100 void errExit( char \*str )

{

puts(str); exit(0);

}

int main( )

{

int sockFd,newSockFd;

struct sockaddr\_in srvAdr,cliAdr; int cliLen,n;

char mesg[MAX\_MSG];

if ( ( sockFd= socket(AF\_INET, SOCK\_STREAM ,0) ) < 0 )

{

errExit ("can.t open stream socket \n");

}

memset(&srvAdr ,0,sizeof(srvAdr)); //initializes the srvAdr structure to zero

// Initialize the structure members to Family name, port no. and ip address srvAdr.sin\_family = AF\_INET;

srvAdr.sin\_addr.s\_addr = htonl(INADDR\_ANY); srvAdr.sin\_port = htons(SRV\_TCP\_PORT);

if (bind (sockFd,(struct sockaddr \*)&srvAdr, sizeof (srvAdr )) < 0)

{

errExit ("Can.t bind local address \n");

}

listen(sockFd,5); // Waiting for the clients, max. 5 clients can be in waiting state while (1)

{

printf("server waiting for new connection :\n"); cliLen = sizeof(cliAdr);

newSockFd = accept(sockFd,( struct sockaddr \*) &cliAdr, &cliLen ); if(newSockFd < 0)

{

errExit ("accept error \n");

}

printf("connected to client :%s \n", inet\_ntoa (cliAdr.sin\_addr )); while(1)

{

n =recv(newSockFd, mesg,MAX\_MSG,0); if ( n < 0)

{

errExit ("recv error \n");

}

if(n== 0)

{

close (newSockFd); break;

}

mesg [n] =0;

if(send (newSockFd,mesg,n,0) !=n)

{

errExit("send error \n");

}

printf("Received following message :\n %s \n ",mesg);

}

}

}

Server Output:

student@student-Veriton-Series:~/Sowjanya$ vi IterEchoSerTCP.c

student@student-Veriton-Series:~/Sowjanya$ cc IterEchoSerTCP.c -o IterEchoSerTCP

student@student-Veriton-Series:~/Sowjanya$ ./IterEchoSerTCP server waiting for new connection :

connected to client :127.0.0.1 Received following message :

Hello! How are You?

Received following message :

Methodist College of Engineering & Technology server waiting for new connection :

# Client Program:

#include <stdio.h>

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

#define SRV\_IP\_ADRS "127.0.0.1"

#define SRV\_TCP\_PORT 5556

#define MAX\_MSG 100

void errExit(char \*str)

{

puts(str); exit(0);

}

int main ( )

{

int sockFd;

struct sockaddr\_in srvAdr; char txmsg[MAX\_MSG]; char rxmsg[MAX\_MSG]; int n;

if( (sockFd =socket(AF\_INET,SOCK\_STREAM,0))<0)

{

errExit("can't open stream socket \n");

}

memset(&srvAdr, 0,sizeof(srvAdr)); srvAdr.sin\_family = AF\_INET; srvAdr.sin\_addr.s\_addr = inet\_addr(SRV\_IP\_ADRS); srvAdr.sin\_port = htons(SRV\_TCP\_PORT);

if (connect(sockFd,(struct sockaddr \*)&srvAdr,sizeof(srvAdr))<0)

{

errExit("can't connect to server \n");

}

while(1)

{

printf("Enter message to send ,Enter # to exit :\n"); fgets(txmsg,MAX\_MSG,stdin);

if( txmsg[0] == '#')

{

break;

}

n = strlen(txmsg)+1;

if ( send(sockFd,txmsg,n,0 ) != n)

{

errExit("send error \n");

}

n=recv(sockFd,rxmsg,MAX\_MSG,0); if ( n < 0)

{

errExit("recv error \n");

}

printf("Received following message : \n %s \n",rxmsg );

}

close (sockFd);

}

Client Output:

student@student-Veriton-Series:~/Sowjanya$ vi IterEchoCliTCP.c

student@student-Veriton-Series:~/Sowjanya$ cc IterEchoCliTCP.c -o IterEchoCliTCP student@student-Veriton-Series:~/Sowjanya$ ./IterEchoCliTCP

Enter message to send ,Enter # to exit : Hello! How are You?

Received following message : Hello! How are You?

Enter message to send ,Enter # to exit :

Methodist College of Engineering & Technology Received following message :

Methodist College of Engineering & Technology

Enter message to send ,Enter # to exit : #

student@student-Veriton-Series:~/Sowjanya$

# Implementation of iterative echo server using connectionless socket system calls

**Description:** This program implements server as a iterative server, which receives the messages from the client and sends the same message as echo to the client. This uses the connection less concept for implementing, which means no connection establishes before the communication, in this each message is attached with the address of the server because there is no fixed path, each message uses its own path to reach to destination. This communication is not reliable and it deliv- ers fast.

## Server side :

* 1. First creats the socket using socket system call.
  2. Binds the name (i.e. IP address, port number and family) of the server to the socket.
  3. Server blocks for receiving the message from the client by using recvfrom system call.
  4. Sends the message by echoing to client using sendto system call.

## Client side :

* 1. First creates the socket.
  2. Initializes the server address to sockaddress\_in structure to which connect request has to send.
  3. Sends the message to server using sendto system call byspecifying server address along with the message.
  4. Then waits for the echo message from the server with recvfrom system call.

# Server Program:

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

#define SRV\_UDP\_PORT 8000

#define MAX\_MSG 100 void errExit(char \*str)

{

puts(str); exit(0);

}

int main( )

{

int sockFd;

struct sockaddr\_in srvAdr,cliAdr; int cliLen,n;

char mesg[MAX\_MSG];

if((sockFd = socket(AF\_INET,SOCK\_DGRAM,0)) < 0 )

{

errExit("Can't open datagram socket \n");

}

memset (&srvAdr ,0,sizeof (srvAdr)); srvAdr.sin\_family = AF\_INET; srvAdr.sin\_addr.s\_addr = htonl (INADDR\_ANY); srvAdr.sin\_port = htons(SRV\_UDP\_PORT);

if(bind(sockFd,(struct sockaddr\*)&srvAdr,sizeof(srvAdr))<0)

{

errExit ("can't bind local address \n");

}

printf("server waiting for messages \n"); while(1)

{

cliLen = sizeof(cliAdr);

n = recvfrom(sockFd,mesg,MAX\_MSG,0 ,(struct sockaddr\*)&cliAdr,&cliLen); if(n < 0)

{ errExit("recvfrom error \n");

}

if(sendto(sockFd,mesg,n ,0,(struct sockaddr \*)&cliAdr,cliLen) !=n)

{ errExit("sendto error \n");

}

printf("Received following message from clients\n %s \n

%s",inet\_ntoa(cliAdr.sin\_addr),mesg);

}}

Server Output:

student@student-Veriton-Series:~/Sowjanya$ vi IterEchoSerUDP.c student@student-Veriton-Series:~/Sowjanya$ cc IterEchoSerUDP.c -o

IterEchoSerUDP

student@student-Veriton-Series:~/Sowjanya$ ./IterEchoSerUDP server waiting for messages

Received following message from clients 127.0.0.1

Hello!

# Client Program:

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

#define SRV\_IP\_ADRS "127.0.0.1"

#define SRV\_UDP\_PORT 8000

#define MAX\_MSG 100

void errExit(char \*str)

{

puts(str); exit(0);

}

int main( )

{

int sockFd;

struct sockaddr\_in srvAdr; char txmsg[MAX\_MSG]; char rxmsg[MAX\_MSG]; int n;

if((sockFd = socket (AF\_INET,SOCK\_DGRAM,0))< 0)

{

errExit("can't open datagram socket \n");

}

memset ( &srvAdr,0,sizeof(srvAdr)); srvAdr.sin\_family = AF\_INET; srvAdr.sin\_addr.s\_addr = inet\_addr(SRV\_IP\_ADRS); srvAdr.sin\_port = htons (SRV\_UDP\_PORT); printf("Enter message to send :\n"); fgets(txmsg,MAX\_MSG,stdin);

n = strlen(txmsg)+1;

if(sendto(sockFd,txmsg,n,0,(struct sockaddr \*)&srvAdr,sizeof(srvAdr ))!=n)

{ errExit("sendto error \n");

}

n = recv(sockFd,rxmsg,MAX\_MSG,0); printf("n=%d\n",n);

if(n < 0)

{

errExit("recv error \n");

}

printf("Received following message :%s\n",rxmsg);

}

Client Program:

student@student-Veriton-Series:~/Sowjanya$ vi IterEchoCliUDP.c student@student-Veriton-Series:~/Sowjanya$ cc IterEchoCliUDP.c -o

IterEchoCliUDP

student@student-Veriton-Series:~/Sowjanya$ ./IterEchoCliUDP Enter message to send :

Hello! n=8

Received following message :Hello! student@student**-**Veriton-Series:~/Sowjanya$

# Implementation of concurrent echo server using connection oriented socket system calls

**Description**: This program implements server as a concurrent server, which receives the messages from the client and sends the same message as echo to the client. In this it creates a child process using fork system call for each client request. This child process service the client request. In this way it services multiple clients at the same time. This uses the connection oriented concept for im- plementing, which means first it establishes the connection, uses the connection for communica- tion and closes it after the communication is ended.

## Server side :

* 1. First creates the socket using socket system call.
  2. Binds the name (i.e. IP address, port number and family) of the server to the socket.
  3. Makes the socket to be ready in passive state by creating queues for storing the client requests using listen system call. In this server is listening for the clients.
  4. Once the connect request comes from the client, it stores in the queue and calls accept system call for establishing the connection. This returns the new connection id, which is used while sending and receiving the messages.
  5. Creates the child process using fork system call.
  6. Receives and sends the messages from/to the client in the child process.
  7. Parent process waits for the next client.
  8. Close the system call after completing the communication.

## Client side :

1. First creates the socket.
2. Initializes the server address to sockaddress\_in structure to which connect request has to send.
3. Makes a connection request to the server using connection system call. If the connec- tion is established at the server it gets the connection id, which is used for the commu- nication. Otherwise it returns negative number as an error.
4. If connection is established, sends and receives the messages using send and recv sys- tem calls.

# Server Program:

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

#define SRV\_TCP\_PORT 8888

#define MAX\_MSG 100 void errExit( char \*str )

{

puts(str); exit(0);

}

int main( )

{

int sockFd,newSockFd;

struct sockaddr\_in srvAdr,cliAdr; int cliLen,n;

char mesg[MAX\_MSG]; int pid;

if( (sockFd = socket(AF\_INET, SOCK\_STREAM ,0) ) < 0 )

{

errExit("can't open stream socket \n");

}

memset(&srvAdr ,0 , sizeof(srvAdr)); srvAdr.sin\_family = AF\_INET; srvAdr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

srvAdr.sin\_port = htons(SRV\_TCP\_PORT); if(bind(sockFd,(struct sockaddr\*)&srvAdr,sizeof (srvAdr )) < 0)

{

errExit("Can't bind local address \n");

}

listen(sockFd,5); while(1)

{

printf("server waiting for new connection :\n"); cliLen = sizeof(cliAdr);

newSockFd = accept(sockFd,(struct sockaddr\*) &cliAdr, &cliLen ); if (newSockFd < 0)

{

errExit("accept error \n");

}

printf("connected to client :%s \n",inet\_ntoa (cliAdr.sin\_addr)); pid=fork( );

printf("parent\n");

if(pid == 0) /\*\*\* child process \*\*\*/

{

printf("child");

while(1)

{

n = recv(newSockFd, mesg,MAX\_MSG,0); if(n < 0)

{

errExit("recv error \n");

}

if(n == 0)

{

break;

}

mesg [n] =0;

if(write(newSockFd,mesg,n) != n)

{

errExit ("send error \n");

}

printf("Received and sent following message :\n%s\n",mesg);

}

exit(0);

}

else /\*\*\* Parent process \*\*\*/

{

close(newSockFd);

}

}

printf("parent");

}

Server Output:

student@student-Veriton-Series:~/Sowjanya$ vi ConcEchoSerTCP.c student@student-Veriton-Series:~/Sowjanya$ cc ConcEchoSerTCP.c -o

ConcEchoSerTCP

student@student-Veriton-Series:~/Sowjanya$ ./ConcEchoSerTCP server waiting for new connection :

connected to client :127.0.0.1 parent

server waiting for new connection : parent

childReceived and sent following message : Hello! I am the First Client.

connected to client :127.0.0.1 parent

server waiting for new connection : parent

childReceived and sent following message :

Hello! I am the Second Client. Received and sent following message:

# Client Program:

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

#define SRV\_IP\_ADRS "127.0.0.1"

#define SRV\_TCP\_PORT 8888

#define MAX\_MSG 100

void errExit(char \*str)

{

puts(str); exit(0);

}

int main ( )

{

int sockFd;

struct sockaddr\_in srvAdr; char txmsg[MAX\_MSG]; char rxmsg[MAX\_MSG]; int n;

if( (sockFd =socket(AF\_INET,SOCK\_STREAM,0))<0)

{

errExit("can't open stream socket \n");

}

memset(&srvAdr, 0,sizeof(srvAdr)); srvAdr.sin\_family = AF\_INET; srvAdr.sin\_addr.s\_addr = inet\_addr(SRV\_IP\_ADRS); srvAdr.sin\_port = htons(SRV\_TCP\_PORT);

if (connect(sockFd,(struct sockaddr \*)&srvAdr,sizeof(srvAdr))<0)

{

errExit("can't connect to server \n");

}

while(1)

{

printf("Enter message to send ,Enter # to exit :\n"); fgets(txmsg,MAX\_MSG,stdin);

if( txmsg[0] == '#')

{

break;

}

n = strlen(txmsg)+1;

if ( send(sockFd,txmsg,n,0 ) != n)

{

errExit("send error \n");

}

n=recv(sockFd,rxmsg,MAX\_MSG,0); if ( n < 0)

{

errExit("recv error \n");

}

printf("Received following message : \n %s \n",rxmsg );

}

close (sockFd); }

Client Output:

student@student-Veriton-Series:~/Sowjanya$ vi ConcEchoCliTCP.c

student@student-Veriton-Series:~/Sowjanya$ cc ConcEchoCliTCP.c -o ConcEchoCliTCP

student@student-Veriton-Series:~/Sowjanya$ ./ConcEchoCliTCP Enter message to send ,Enter # to exit :

Hello! I am the First Client. Received following message :

Hello! I am the First Client.

Enter message to send ,Enter # to exit :

#

student@student-Veriton-Series:~/Sowjanya$

student@student-Veriton-Series:~/Sowjanya$ cc ConcEchoCliTCP.c -o ConcEchoCliTCP

student@student-Veriton-Series:~/Sowjanya$ ./ConcEchoCliTCP Enter message to send ,Enter # to exit :

Hello! I am the Second Client. Received following message :

Hello! I am the Second Client.

Enter message to send ,Enter # to exit : #

student@student-Veriton-Series:~/Sowjanya$

# Implementation of concurrent echo server using connectionless socket system calls

**Server Program:**

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

int main(int argc,char \*argv[])

{

int sockfd,rval,pid;

char buff1[20],buff2[20];

struct sockaddr\_in server,client; int len;

sockfd=socket(AF\_INET,SOCK\_DGRAM,IPPROTO\_UDP); if(sockfd==-1)

{

perror("\n SOCK\_ERR\n"); exit(1);

}

server.sin\_family=AF\_INET; server.sin\_addr.s\_addr=inet\_addr("192.168.0.5"); server.sin\_port=htons(3221);

rval=bind(sockfd,(struct sockaddr \*)&server,sizeof(server)); if(rval!=-1)

{

pid=fork(); if(pid==0)

{

printf("\n child process executing\n"); printf("\n child process ID is:%d\n",getpid()); len=sizeof(client);

rval=recvfrom(sockfd,buff1,20,0,(struct sockaddr \*)&client,&len); if(rval==-1)

{

perror("\n RECV\_ERR\n"); exit(1);

}

else

{

printf("\n received message is:%s\n",buff1);

}

rval=sendto(sockfd,buff1,sizeof(buff1),0,(struct sockaddr \*)&client,sizeof(client)); if(rval!=-1)

{

printf("\n message sent successfully\n");

}

else

{ perror("\n SEND\_ERR\n"); exit(1);

}

}

else

printf("\n parent process \n");

printf("parent process ID is %d\n",getppid());

}

else

{ perror("\n BIND\_ERR\n"); exit(1);

}

}

Server Output:

student@student-Veriton-Series:~/Sowjanya$ vi ConcEchoSerUDP.c student@student-Veriton-Series:~/Sowjanya$ cc ConcEchoSerUDP.c -o

ConcEchoSerUDP

student@student-Veriton-Series:~/Sowjanya$ ./ConcEchoSerUDP

parent process

parent process ID is 7324 child process executing

child process ID is:11287

student@student-Veriton-Series:~/Sowjanya$ received message is:Hello!

message sent successfully parent process ID is 1

# Client Program:

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

int main(int argc,char \* argv[])

{

int sockfd,rval;

char buff1[20],buff2[20];

struct sockaddr\_in server,client; int len;

sockfd=socket(AF\_INET,SOCK\_DGRAM,IPPROTO\_UDP); if(sockfd==-1)

{

perror("\n SOCK\_ERR\n"); exit(1);

}

server.sin\_family=AF\_INET; server.sin\_addr.s\_addr=inet\_addr("192.168.0.5"); server.sin\_port=htons(3221);

rval=bind(sockfd,(struct sockaddr \*)&client,sizeof(client)); if(rval==-1)

{

perror("\n BIND\_ERR\n"); exit(1);

}

printf("\n enter a message\n"); scanf("%s",buff1);

rval=sendto(sockfd,buff1,sizeof(buff1),0,(struct sockaddr \*)&server,sizeof(server)); if(rval!=-1)

{

printf("\n message sent succesfully\n");

}

else

{

perror("\n SEND\_ERR\n"); exit(1);

}

len=sizeof(server);

rval=recvfrom(sockfd,buff1,20,0,(struct sockaddr \*)&server,&len); if(rval==-1)

{

perror("\n RECV\_ERR\n"); exit(1);

}

else

{

printf("\n received message is %s\n",buff1);

}

}

Client Output:

student@student-Veriton-Series:~/Sowjanya$ vi ConcEchoCliUDP.c student@student-Veriton-Series:~/Sowjanya$ vi ConcEchoCliUDP.c student@student-Veriton-Series:~/Sowjanya$ cc ConcEchoCliUDP.c -o

ConcEchoCliUDP

student@student-Veriton-Series:~/Sowjanya$ ./ConcEchoCliUDP

enter a message Hello!

message sent succesfully received message is Hello!

student@student-Veriton-Series:~/Sowjanya$

1. **SIMULATION OF NETWORK TOPOLOGIES**
   1. **AIM: Simulate the different network topologies using packet tracer THEORY:**

INTRODUCTION TO PACKET TRACER

This is a tool built by Cisco. This tool provides a network simulation to practice simple and complex networks.

To download: https://[www.netacad.com/courses/packet-tracer/introduction-packet-tracer.](http://www.netacad.com/courses/packet-tracer/introduction-packet-tracer) Workspace:

* + 1. Logical –

Logical workspace shows the logical network topology of the network the user has built. It represents the placing, connecting and clustering virtual network devices.

* + 1. Physical –

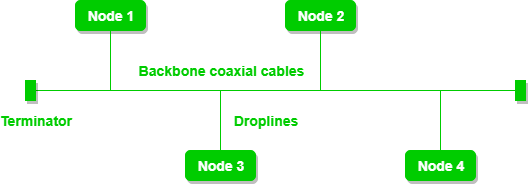
Physical workspace shows the graphical physical dimension of the logical network. It depicts the scale and placement in how network devices such as routers, switches and hosts would look in a real environment. It also provides geographical representation of networks, including multiple buildings, cities and wiring closets.

TOPOLOGY: The arrangement of a network that comprises nodes and connecting lines via sender and receiver is referred to as network topology.

## BUS TOPOLOGY

**AIM: Implement Bus topology using Packet tracer THEORY:**

Bus topology is a network type in which every computer and network device is connected to a single cable. It transmits the data from one end to another in a single direction. No bi-directional feature is in bus topology. It is a multi-point connection and a non-robust topology because if the backbone fails the topology crashes.



A bus topology with shared backbone cable. The nodes are connected to the channel via drop lines.

## Advantages of this topology:

If N devices are connected to each other in a bus topology, then the number of cables required to connect them is 1, which is known as backbone cable, and N drop lines are required.

The cost of the cable is less as compared to other topologies, but it is used to build small networks.

## Problems with this topology:

If the common cable fails, then the whole system will crash down.

If the network traffic is heavy, it increases collisions in the network. To avoid this, various protocols are used in the MAC layer known as Pure Aloha, Slotted Aloha, CSMA/CD, etc.

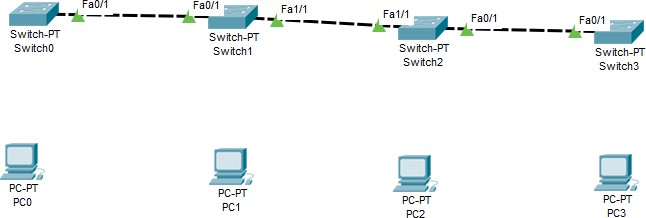
Security is very low.

## PROGRAM

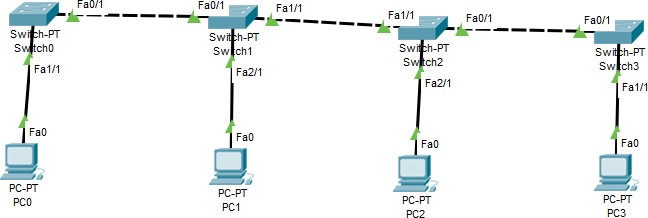
STEP 1: Take four devices (PC’s)

STEP 2: Take four switches (one for each device)

STEP 3: Click on Connections. Connect between two switches with fast Ethernet. Likewise connect all the switches (dotted black line – Copper Cross Over)



STEP 4: Connect the switch with the respective PC’s (solid black line- Copper Straight through)

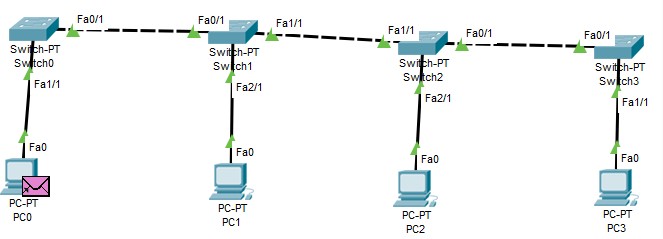
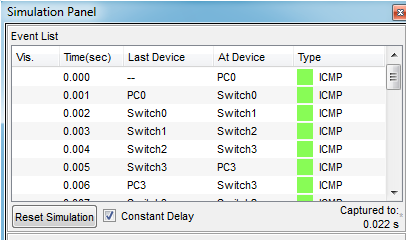


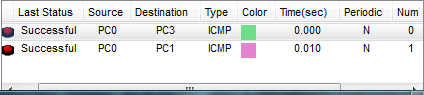
STEP 5: Assign IP address

Double click on the device. A pop up window will get open. Click on Desktop and IP Configuration PC0: IP Address – 192.168.0.1 Subnet mask –255.255.255.0

PC1: IP Address – 192.168.0.2 Subnet mask –255.255.255.0 PC2: IP Address – 192.168.0.3 Subnet mask –255.255.255.0 PC3: IP Address – 192.168.0.4 Subnet mask –255.255.255.0

STEP 6: Select a packet and check whether it is delivered from PC0 to PC3

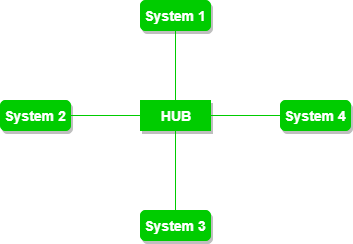
 



## STAR TOPOLOGY

**AIM: Implement star topology using packet tracer THEORY:**

In star topology, all the devices are connected to a single hub through a cable. This hub is the central node and all other nodes are connected to the central node. The hub can be passive in nature i.e., not intelligent hub such as broadcasting devices, at the same time the hub can be intelligent known as active hubs. Active hubs have repeaters in them.



A star topology having four systems connected to single point of connection i.e. hub.

## Advantages of this topology:

If N devices are connected to each other in a star topology, then the number of cables required to connect them is N. So, it is easy to set up.

Each device requires only 1 port i.e. to connect to the hub, therefore total number of ports required is N.

## Problems with this topology:

If the concentrator (hub) on which the whole topology relies fails, the whole system will crash down. The cost of installation is high.

Performance is based on the single concentrator i.e. hub.

## PROGRAM

STEP 1: Take four devices (PC’s) STEP 2: Take one switch

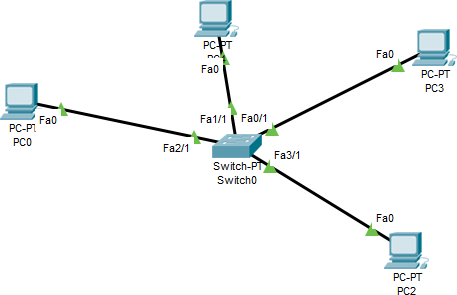
STEP 3: Click on Connections. Connect all the PC’s to the switch

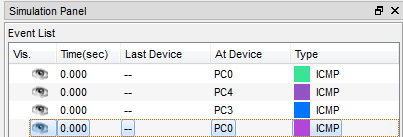
STEP 4: Assign IP address

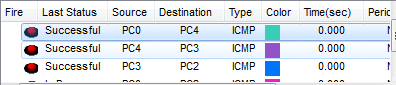
Double click on the device. A pop up window will get open. Click on Desktop and IP Configuration PC0: IP Address – 192.168.0.1 Subnet mask –255.255.255.0

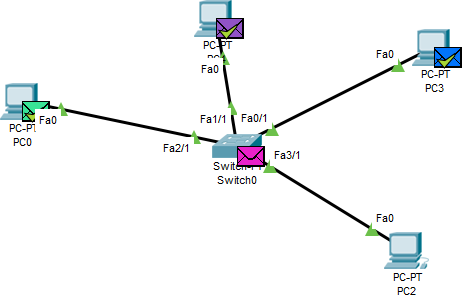
PC1: IP Address – 192.168.0.2 Subnet mask –255.255.255.0 PC2: IP Address – 192.168.0.3 Subnet mask –255.255.255.0 PC3: IP Address – 192.168.0.4 Subnet mask –255.255.255.0

STEP 5: Select a packet ping all PCs





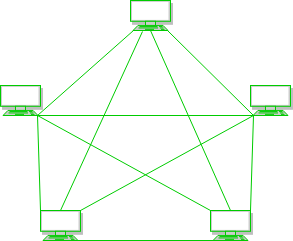




## MESH TOPOLOGY

**AIM: Implement mesh topology using packet tracer THEORY:**

In a mesh topology, every device is connected to another device via the particular channel.



Every device is connected with another via dedicated channels. These channels are known as links.

If suppose, N number of devices are connected with each other in a mesh topology, the total number of ports that are required by each device is N-1. In Figure 1, there are 5 devices connected to each other, hence the total number of ports required by each device is 4. Total number of ports required=N\*(N-1). If suppose, N number of devices are connected with each other in a mesh topology, then a total number of dedicated links required to connect them is NC2 i.e. N(N-1)/2. In Figure 1, there are 5 devices connected to each other, hence the total number of links required is 5\*4/2 = 10.

## Advantages of this topology :

It is robust.

The fault is diagnosed easily. Data is reliable because data is transferred among the devices through dedicated channels or links.

Provides security and privacy.

## Problems with this topology :

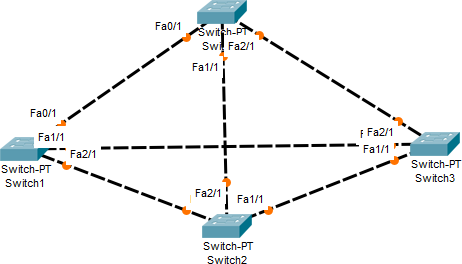
Installation and configuration are difficult.

The cost of cables is high as bulk wiring is required, hence suitable for less number of devices. The cost of maintenance is high.

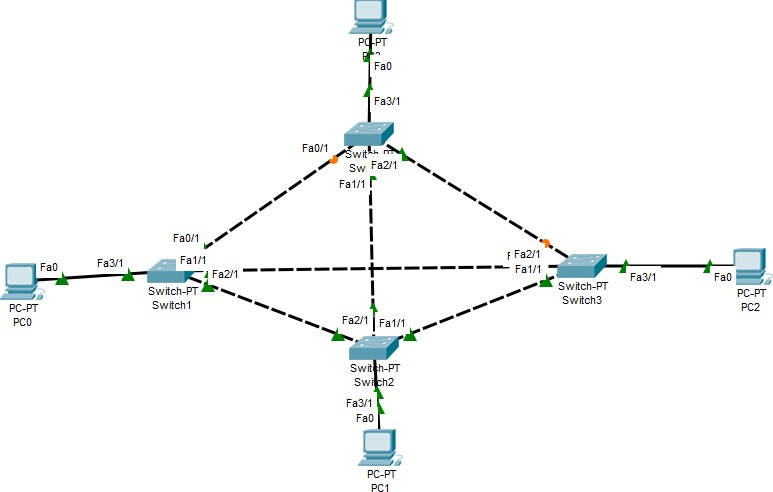
## PROGRAM

STEP 1: Take four switch

STEP 2: Connect all the switches to one another



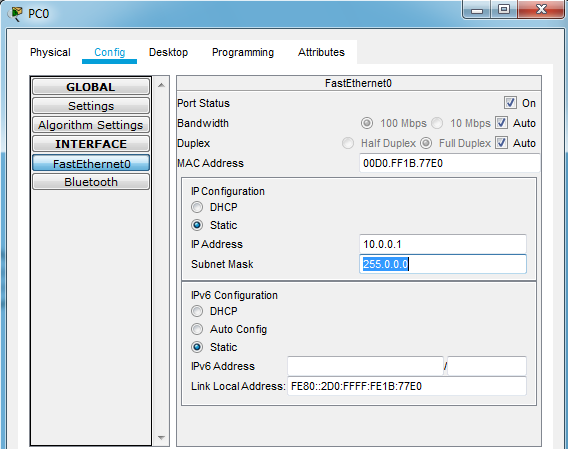
STEP 3: Take some PC’s and attach to switch



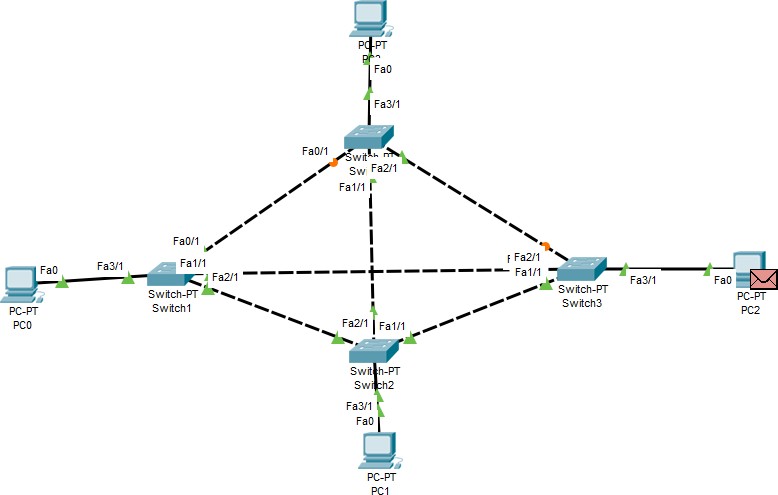
STEP 4: Assign IP address

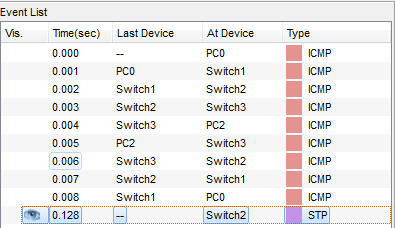
Double click on the device. A pop up window will get open. Click on Config then fast Ethernet 0 and assign IP addresses

PC0: IP Address – 10.0.0.1 Subnet mask –255.0.0.0 PC1: IP Address – 10.0.0.2 Subnet mask –255.0.0.0 PC2: IP Address – 10.0.0.3 Subnet mask –255.0.0.0 PC3: IP Address – 10.0.0.4 Subnet mask –255.0.0.0



STEP 5: Select a packet(PDU) ping from PC0 to PC2(for ex)





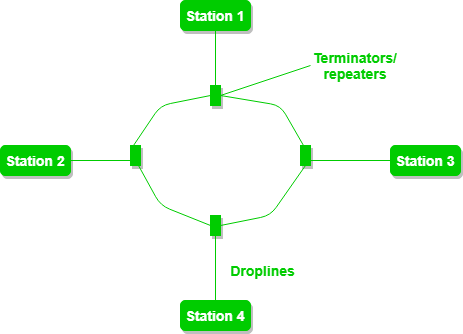
## RING TOPOLOGY

**AIM: Implement ring topology using packet tracer THEORY:**

In this topology, it forms a ring connecting devices with its exactly two neighboring devices.

A number of repeaters are used for Ring topology with a 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.

The transmission is unidirectional, but it can be made bidirectional by having 2 connections between each Network Node, it is called Dual Ring Topology.



A ring topology comprises of 4 stations connected with each forming a ring. The following operations take place in ring topology are :

One station is known as **monitor** station which takes all the responsibility to perform the operations. To transmit the data, the station has to hold the token. After the transmission is done, the token is to be released for other stations to use.

When no station is transmitting the data, then the token will circulate in the ring.

There are two types of token release techniques: **Early token release** releases the token just after transmitting the data and **Delay token release** releases the token after the acknowledgment is received from the receiver.

## Advantages of this topology:

The possibility of collision is minimum in this type of topology. Cheap to install and expand.

## Problems with this topology:

Troubleshooting is difficult in this topology.

The addition of stations in between or removal of stations can disturb the whole topology. Less secure.

PROGRAM

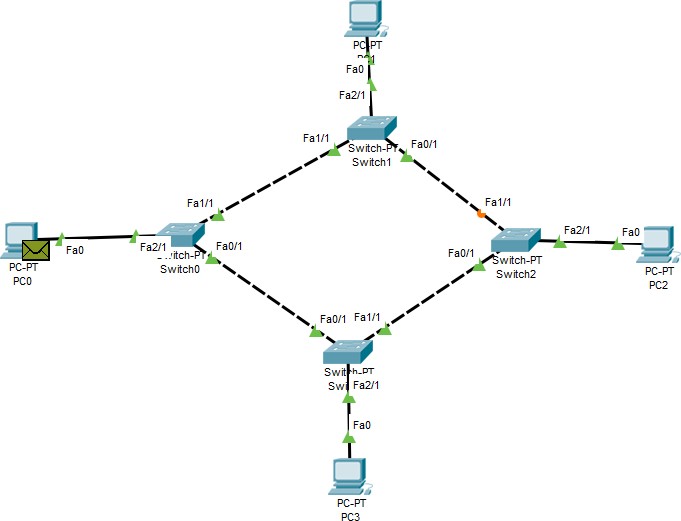
STEP 1: Take four devices (PCs or laptop) and four switch

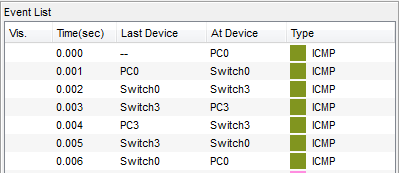
STEP 2: Connect all the switches to one another using cable (Copper Cross Over) STEP 3: Connect PCs to switch using cable (Copper Straight)

STEP 4: Assign IP address

Double click on the device. A pop up window will get open. Click on Config then fast Ethernet 0 and assign IP addresses

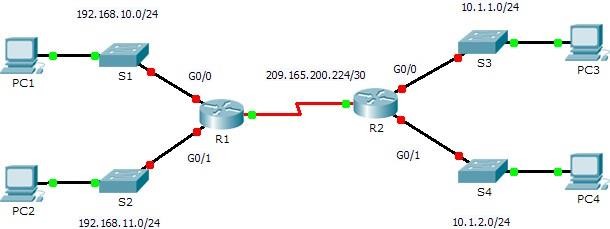
PC0: IP Address – 10.0.0.1 Subnet mask –255.0.0.0 PC1: IP Address – 10.0.0.2 Subnet mask –255.0.0.0 PC2: IP Address – 10.0.0.3 Subnet mask –255.0.0.0 PC3: IP Address – 10.0.0.4 Subnet mask –255.0.0.0 STEP 5: Send a PDU from PC0 to PC2(for ex)





## RESULT: Thus simulation of different network topologies like star, bus, ring, mesh is done successfully.

**Packet Tracer - Connect a Router to a LAN**



## Addressing Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Device** | **Interface** | **IP Address** | **Subnet Mask** | **Default Gateway** |
| R1 | G0/0 | 192.168.10.1 | 255.255.255.0 | N/A |
| G0/1 | 192.168.11.1 | 255.255.255.0 | N/A |
| S0/0/0 (DCE) | 209.165.200.225 | 255.255.255.252 | N/A |
| R2 | G0/0 | 10.1.1.1 | 255.255.255.0 | N/A |
| G0/1 | 10.1.2.1 | 255.255.255.0 | N/A |
| S0/0/0 | 209.165.200.226 | 255.255.255.252 | N/A |
| PC1 | NIC | 192.168.10.10 | 255.255.255.0 | 192.168.10.1 |
| PC2 | NIC | 192.168.11.10 | 255.255.255.0 | 192.168.11.1 |
| PC3 | NIC | 10.1.1.10 | 255.255.255.0 | 10.1.1.1 |
| PC4 | NIC | 10.1.2.10 | 255.255.255.0 | 10.1.2.1 |

Objectives

## Part 1: Display Router Information Part 2: Configure Router Interfaces Part

**3: Verify the Configuration**

Background

In this activity, you will use various **show** commands to display the current state of the router. You will then use the [Addressing Table](#_bookmark0) to configure router Ethernet interfaces. Finally, you will use commands to verify and test your configurations.

**Note:** The routers in this activity are partially configured. Some of the configurations are not covered in this course, but are provided to assist you in using verification commands.

**Note:** The serial interfaces are already configured and active. In addition, routing is configured using EIGRP. This is done so that this activity is (1) consistent with examples shown in the chapter,

(2) ready to provide complete output from **show** commands when the student configures and activates the Ethernet interfaces.

Part 1:Display Router Information

Step 1:Display interface information on R1.

**Note:** Click a device and then click the **CLI** tab to access the command line directly.

The console password is

**cisco**. The privileged EXEC password is **class**.

Which command displays the statistics for all interfaces configured on a router? show interfaces Which command displays the information about the Serial 0/0/0 interface only? show interface serial 0/0/0

Enter the command to display the statistics for the Serial 0/0/0 interface on R1 and answer the following questions:

What is the IP address configured on **R1**? 209.165.200.225/30 What is the bandwidth on the Serial 0/0/0 interface? 1544 kbits

Enter the command to display the statistics for the GigabitEthernet 0/0 interface and answer the following questions:

What is the IP address on **R1**? There is no IP address configured on the GigabitEthernet 0/0 interface.

What is the MAC address of the GigabitEthernet 0/0 interface? 000d.bd6c.7d01 What is the bandwidth on the GigabitEthernet 0/0 interface? 1000000 kbits

Step 2:Display a summary list of the interfaces on R1.

Which command displays a brief summary of the current interfaces, statuses, and IP addresses assigned to them? show ip interface brief

Enter the command on each router and answer the following questions:

How many serial interfaces are there on **R1** and **R2**? Each router has 2 serial interfaces.

How many Ethernet interfaces are there on **R1** and **R2**? R1 has 6 Ethernet interfaces and R2 has 2 Ethernet interfaces.

Are all the Ethernet interfaces on **R1** the same? If no, explain the difference(s). No they are not. There are two Gigabit Ethernet interfaces and 4 Fast Ethernet interfaces. Gigabit Ethernet interfaces support speeds of up to 1,000,000,000 bits and Fast Ethernet interfaces support speeds of up to 1,000,000 bits.

Step 3:Display the routing table on R1.

What command displays the content of the routing table? show ip route Enter the command on **R1** and answer the following questions:

How many connected routes are there (uses the C code)? 1 2) Which route is listed? 209.165.200.224/30

3) How does a router handle a packet destined for a network that is not listed in the routing table? A router will only send packets to a network listed in the routing table. If a network is not listed, the packet will be dropped.

Part 2:Configure Router Interfaces

Step 1:Configure the GigabitEthernet 0/0 interface on R1.

Enter the following commands to address and activate the GigabitEthernet 0/0 interface on **R1**: R1(config)# **interface gigabitethernet 0/0**

R1(config-if)# **ip address 192.168.10.1 255.255.255.0**

R1(config-if)# **no shutdown**

%LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/0, changed state to up

It is good practice to configure a description for each interface to help document the network information. Configure an interface description indicating to which device it is connected.

R1(config-if)# **description LAN connection to S1 R1** should now be able to ping PC1.

R1(config-if)# **end**

%SYS-5-CONFIG\_I: Configured from console by console R1# **ping 192.168.10.10**

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 192.168.10.10, timeout is 2 seconds:

.!!!!

Success rate is 80 percent (4/5), round-trip min/avg/max = 0/2/8 ms

Step 2:Configure the remaining Gigabit Ethernet Interfaces on R1 and R2.

Use the information in the [Addressing Table](#_bookmark0) to finish the interface configurations for **R1** and **R2**. For each interface, do the following:

Enter the IP address and activate the interface. Configure an appropriate description.

Verify interface configurations.

Step 3:Back up the configurations to NVRAM.

Save the configuration files on both routers to NVRAM. What command did you use? copy run start

Part 3:Verify the Configuration

Step 1:Use verification commands to check your interface configurations.

Use the **show ip interface brief** command on both **R1** and **R2** to quickly verify that the interfaces are configured with the correct IP address and active.

How many interfaces on **R1** and **R2** are configured with IP addresses and in the “up” and “up” state? 3 on each router

What part of the interface configuration is NOT displayed in the command output? The subnet mask

What commands can you use to verify this part of the configuration? show run, show interfaces, show ip protocols

Use the **show ip route** command on both **R1** and **R2** to view the current routing tables and answer the following questions:

How many connected routes (uses the **C** code) do you see on each router? 3 How many EIGRP routes (uses the **D** code) do you see on each router? 2

If the router knows all the routes in the network, then the number of connected routes and dynamically learned routes (EIGRP) should equal the total number of LANs and WANs. How many LANs and WANs are in the topology? 5

Does this number match the number of C and D routes shown in the routing table? yes

**Note:** If your answer is “no”, then you are missing a required configuration. Review the steps in Part 2.

Step 2:Test end-to-end connectivity across the network.

You should now be able to ping from any PC to any other PC on the network. In addition, you should be able to ping the active interfaces on the routers. For example, the following should tests should be successful:

From the command line on PC1, ping PC4. From the command line on R2, ping PC2.

**Note:** For simplicity in this activity, the switches are not configured; you will not be able to ping them.

## Configuration of a network using different routing protocols. AIM: Configuration of a network using different routing protocols. THEORY:

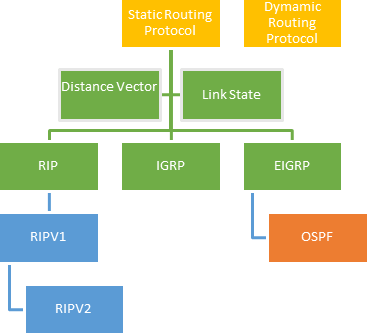
**Routing Protocols** are the set of defined rules used by the routers to communicate between source & destination. They do not move the information to the source to a destination, but only update the routing table that contains the information.

Network Router protocols helps you to specify way routers communicate with each other. It allows the network to select routes between any two nodes on a computer network.

Types of Routing Protocols

There are mainly two types of Network Routing Protocols

1. Static
2. Dynamic



*Static Routing Protocols*

Static routing protocols are used when an administrator manually assigns the path from source to the destination network. It offers more security to the network.

**Advantages**

* No overhead on router CPU.
* No unused bandwidth between links.
* Only the administrator is able to add routes

**Disadvantages**

* The administrator must know how each router is connected.
* Not an ideal option for large networks as it is time intensive.
* Whenever link fails all the network goes down which is not feasible in small networks.

*Dynamic Routing Protocols*

Dynamic routing protocols are another important type of routing protocol. It helps routers to add information to their routing tables from connected routers automatically. These types of protocols also send out topology updates whenever the network changes' topological structure.

**Advantage:**

* Easier to configure even on larger networks.
* It will be dynamically able to choose a different route in case if a link goes down.
* It helps you to do load balancing between multiple links.

**Disadvantage:**

* Updates are shared between routers, so it consumes bandwidth.
* Routing protocols put an additional load on router CPU or RAM.

## OPEN SHORTEST PATH FIRST (OSPF) ALGORITHM

**AIM: Implement Link state routing algorithm using packet tracer THEORY:**

*Open Shortest Path First (OSPF) is a link-state routing protocol that is used to find the best path between the source and the destination router using its own Shortest Path First). OSPF is developed by Internet Engineering Task Force (IETF) as one of the Interior Gateway Protocol (IGP), i.e, the protocol which aims at moving the packet within a large autonomous system or routing domain. It is a network layer protocol which works on the protocol number 89 and uses AD value 110. OSPF uses multicast address 224.0.0.5 for normal communication and 224.0.0.6 for update to designated router(DR)/Backup Designated Router (BDR).*

*OSPF terms –*

* + 1. *Router I’d – It is the highest active IP address present on the router. First, highest loopback ad- dress is considered. If no loopback is configured then the highest active IP address on the in-*

*terface of the router is considered. Router priority – It is a 8 bit value assigned to a router op- erating OSPF, used to elect DR and BDR in a broadcast network.*

* + 1. *Designated Router (DR) – It is elected to minimize the number of adjacency formed. DR dis- tributes the LSAs to all the other routers. DR is elected in a broadcast network to which all the other routers shares their DBD. In a broadcast network, router requests for an update to DR and DR will respond to that request with an update.*
    2. *Backup Designated Router (BDR) – BDR is backup to DR in a broadcast network. When DR goes down, BDR becomes DR and performs its functions.*

## PROGRAM

STEP1 **:** Take three generic routers(Router PT(generic router)) STEP 2: Take two generic computers

STEP 3: Establish the connections STEP 4: Assign the IP Address

Double click on the device. A pop up window will get open. Click on Config then fast Ethernet 0 and assign IP addresses

PC0: IP Address – 192.168.1.2 Subnet mask –255.255.255.0 Default gateway:192.168.1.1 PC1: IP Address – 192.168.2.2 Subnet mask –255.255.255.0 Default gateway:192.168.2.1 STEP 5: Configure the routers and serial ports

Double click on the router. Go to config then to FastEthernet 0, and then IP Congiguration Set it ON

* Router1: 192.168.1.1 255.255.255.0

Select Serial2/0 (connection between the routers) Click on ON

IP address: 10.0.0.2 255.0.0.0

Set the clock to 64000

Select Serial3/0 (connection between the routers) Click on ON

IP address: 12.12.0.2 255.0.0.0

Set the clock to 64000

* Router2: No fast Ethernet connection

Select Serial2/0 (connection between the routers)

Click on ON (Port Status)

IP address: 10.10.0.3 255.0.0.0

Set the clock to 64000

Select Serial3/0 (connection between the routers) Click on ON (Port Status)

IP address: 11.11.0.2 255.0.0.0

Set the clock to 64000

* Router3: 192.168.2.1 255.255.255.0

Select Serial2/0 (connection between the routers) Click on ON (Port Status)

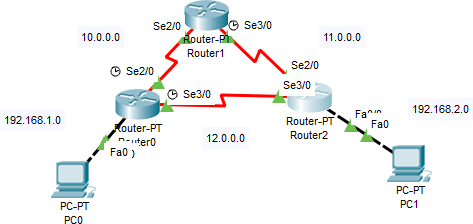
IP address: 11.11.0.3 255.0.0.0

Set the clock to 64000

Select Serial3/0 (connection between the routers) Click on ON (Port Status)

IP address: 12.12.0.3 255.0.0.0

Set the clock to 64000



STEP 6: Configure the network with OSPF

1. Select Router0 , select CLI(command line interface) exit

Router(config)#router ospf 1 // goes into ospf configuration with process id 1

// For connection with PC and other routers (Adding the network)

//Command is network <ip address> <wild card mask which parts of ip address are

// available (complement of subnet mask)

Router(config-router)#network 192.168.1.0 0.0.0.255 area 0 //PC to router 0

Router(config-router)#network 10.0.0.0 0.255.255.255 area 0 //Router 0 to router 1

Router(config-router)#network 12.0.0.0 0.255.255.255 area 0 // Router 0 to router 2 Router(config-router)# exit

Go to Config  Settings  SaveClose

1. Select Router1 , select CLI(command line interface) Router(config-if)#exit

Router(config)#router ospf 1

Router(config-router)#network 10.0.0.0 0.255.255.255 area 0

Router(config-router)#network 11.0.0.0 0.255.255.255 area 0 Router(config-router)#exit

Go to Config  Settings  SaveClose

1. Select Router2 , select CLI(command line interface) Router(config-if)#exit

Router(config)#router ospf 1

Router(config-router)#network 192.168.2.0 0.0.0.255 area 0

Router(config-router)#network 11.0.0.0 0.255.255.255 area 0

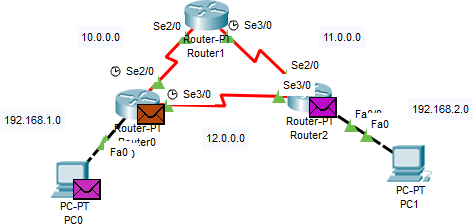
Router(config-router)#network 12.0.0.0 0.255.255.255 area 0 Router(config-router)#exit

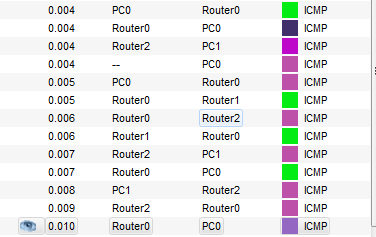
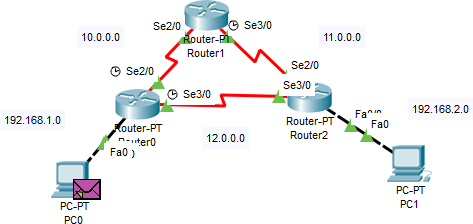
Go to Config  Settings  SaveClose

STEP 7: Drop the packets from PC to router and check whether they are successful or not



STEP 8: Check whether the packet is sent from PC0 to PC1





## RIP (ROUTING INFORMATION PROTOCOL)

**AIM: Implement RIP Routing Protocol using packet tracer THEORY:**

Routing Information Protocol (RIP) is **a distance-vector routing protocol**. Routers running the distance-vector protocol send all or a portion of their routing tables in routing-update messages to their neighbors. You can use RIP to configure the hosts as part of a RIP network.

## Hop Count:

Hop count is the number of routers occurring in between the source and destination network. The path with the lowest hop count is considered as the best route to reach a network and therefore placed in the routing table. RIP prevents routing loops by limiting the number of hopes allowed in a path from source and destination. The maximum hop count allowed for RIP is 15 and hop count of 16 is considered as network unreachable.

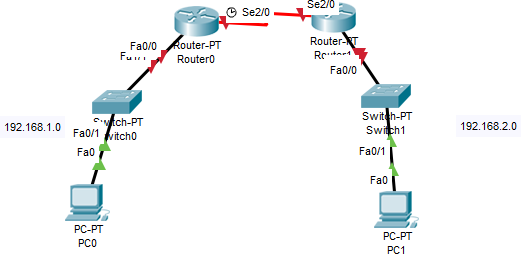
## Features of RIP:

1. Updates of the network are exchanged periodically.
2. Updates (routing information) are always broadcast.
3. Full routing tables are sent in updates.
4. Routers always trust on routing information received from neighbor routers. This is also known as Routing on rumours.

PROGRAM

STEP 1: Take two PC’s , two routers and two switches

STEP 2: Connect PC to switch and router. Connect the router



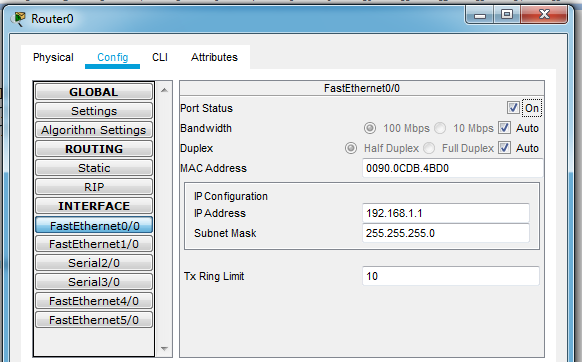
STEP 3: Configure the network

* 1. Assign the IP Address

Double click on the device. A pop up window will get open. Click on Config then fast Ethernet 0 and assign IP addresses

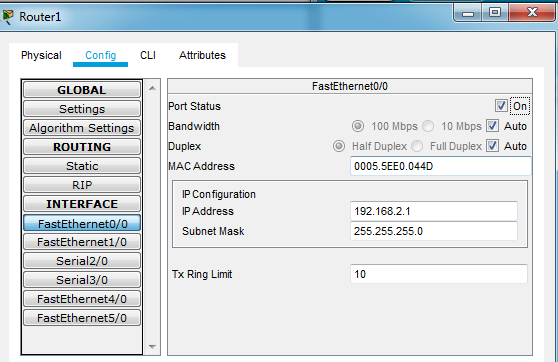
PC0: IP Address – 192.168.1.2 Subnet mask –255.255.255.0 Default gateway:192.168.1.1 PC1: IP Address – 192.168.2.2 Subnet mask –255.255.255.0 Default gateway:192.168.2.1

* 1. Configure router Router 0

Go to Config, Fast Ethernet 0 , Assign the IP Address 192.168.1.1 255.255.255.0

Click on ON Router 1

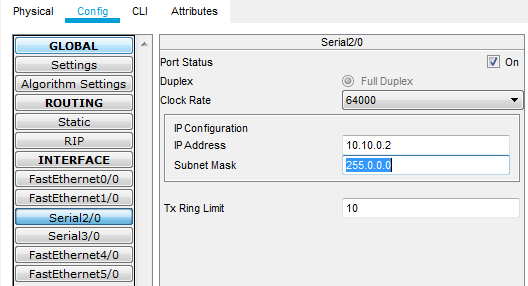
Go to Config, Fast Ethernet 0 , Assign the IP Address 192.168.2.1 255.255.255.0



Click on ON

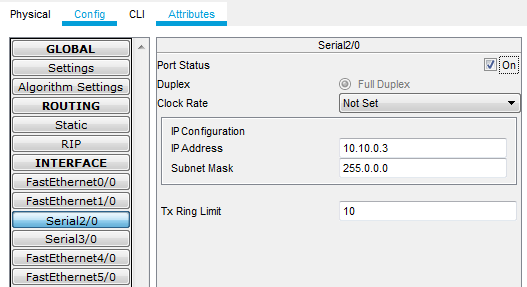
Configure the network between two Routers(10.0.0.0) Click on Router 0

Select Config, Serial 2/0 and assign the IP Address 10.10.0.2 255.0.0.0 Clock rate 64000 Port Status  ON



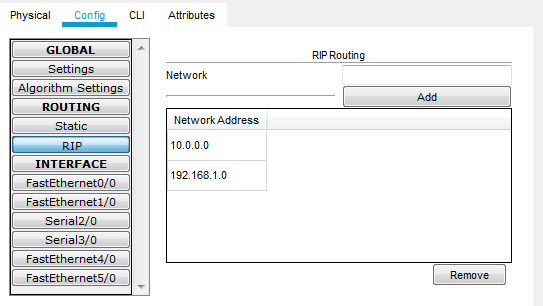
Click on Router 1

Select Config, Serial 2/0 and assign the IP Address 10.10.0.3 255.0.0.0 Clock rate  Not Set Port Status  ON



STEP 4: Configure the routers to use RIP and to know other network Click on Router 0  Config  RIP

In network tab add the networks id which the router knows



Click on Setting and Save option

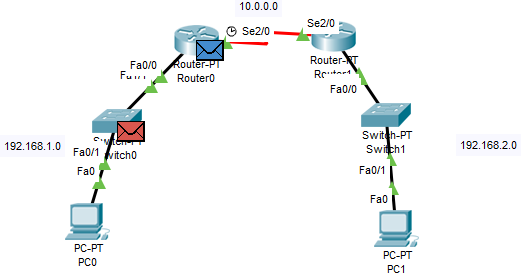
Click on Router 1  CLI exit Router(config)#router rip

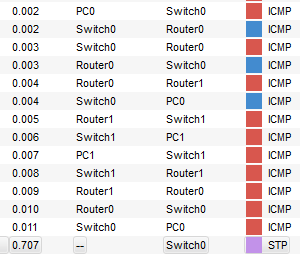
Router(config-router)#network 192.168.2.0

Router(config-router)#network 10.0.0.0 Router(config-router)#exit Router(config)#

Click on Setting and Save option

STEP 5: Check whether the packet is routed successfully from one machine to another

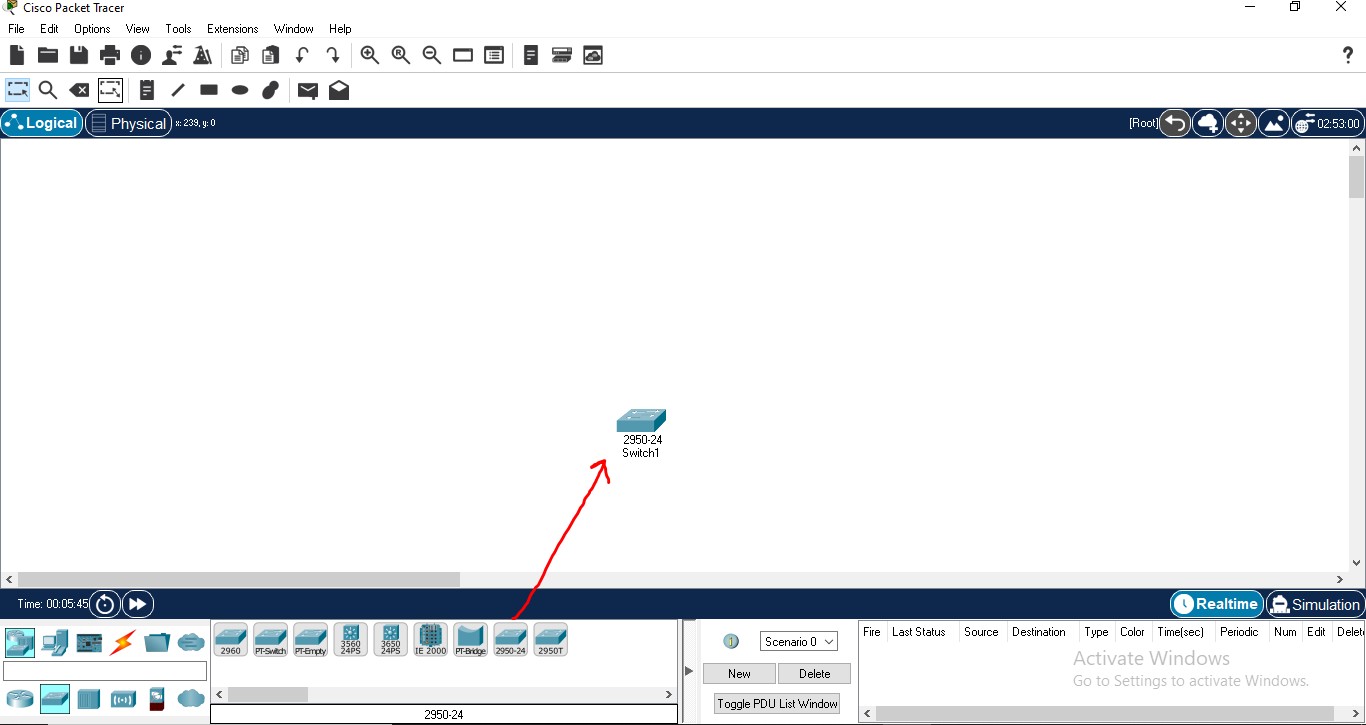


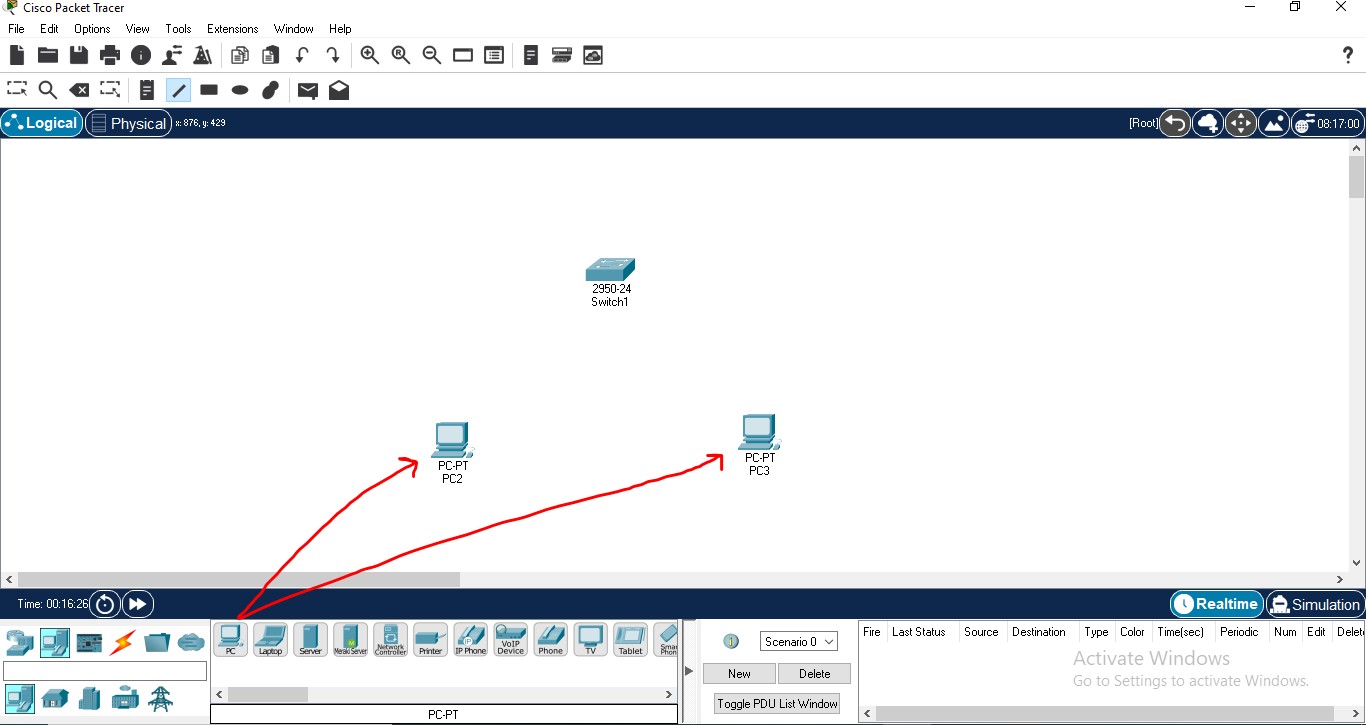
**RESULT: Thus the configuration of a network using different routing protocols is done success- fully.**

* + 1. Basic Switch configuration using CISCO packet tracer

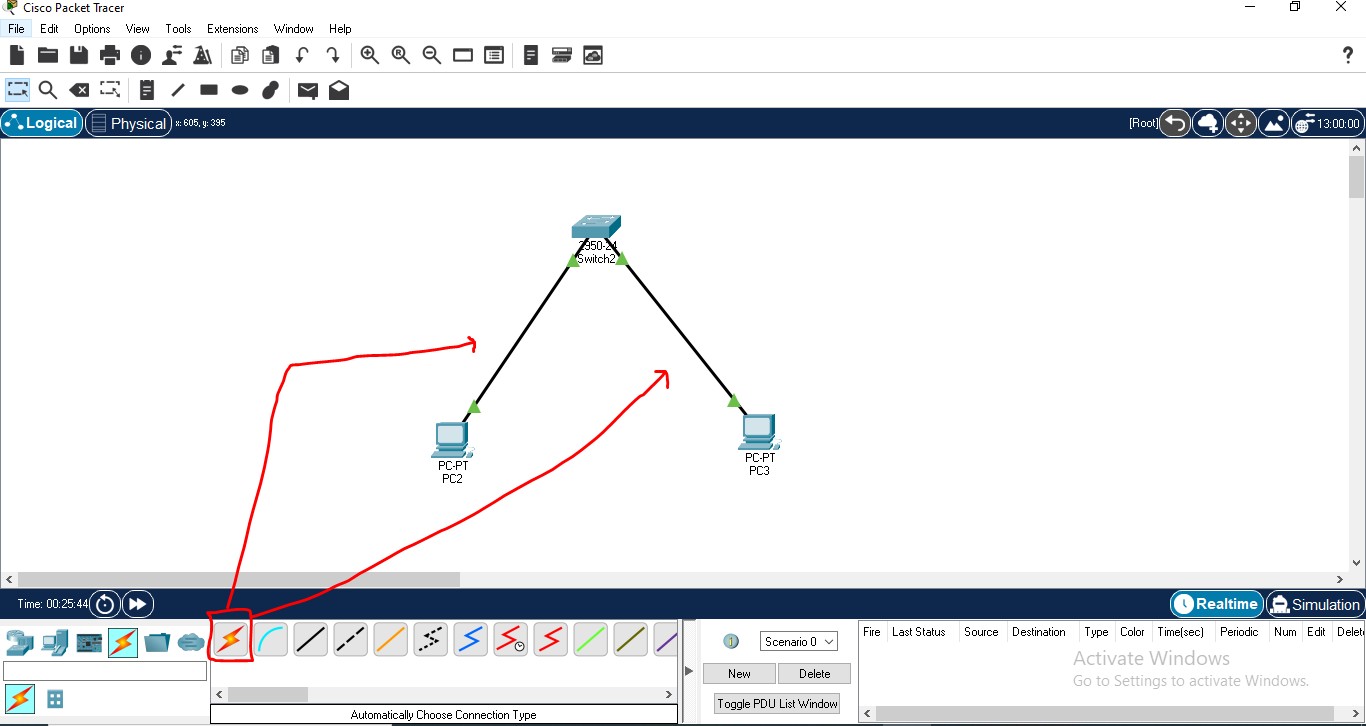
Using Packet Tracer 8.0 : In this section, We will design a simple network topology and show you how Packet Tracer works. First, start Packet Tracer 8.0. Now click on Network Devices icon and then click on Switches icon as marked in the screenshot below.



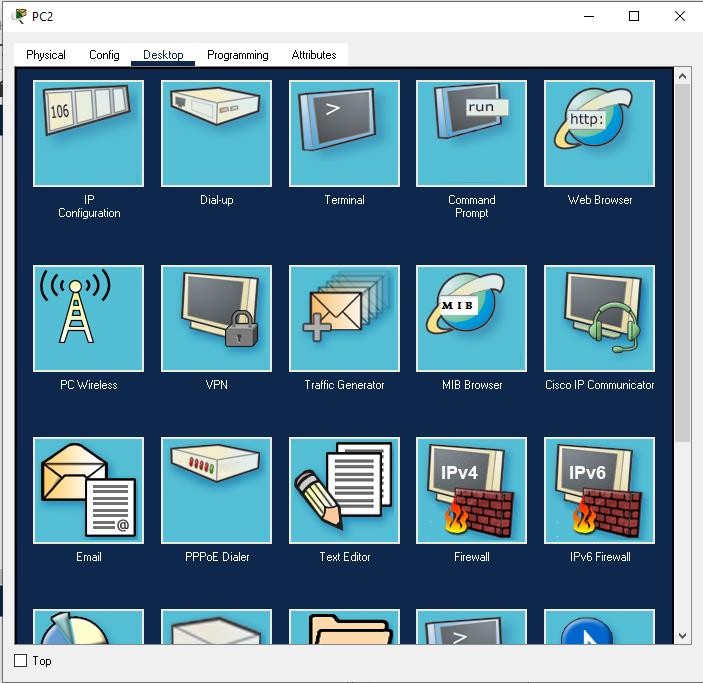
* Then select end devices icon and select two pc’s as shown in the below pic



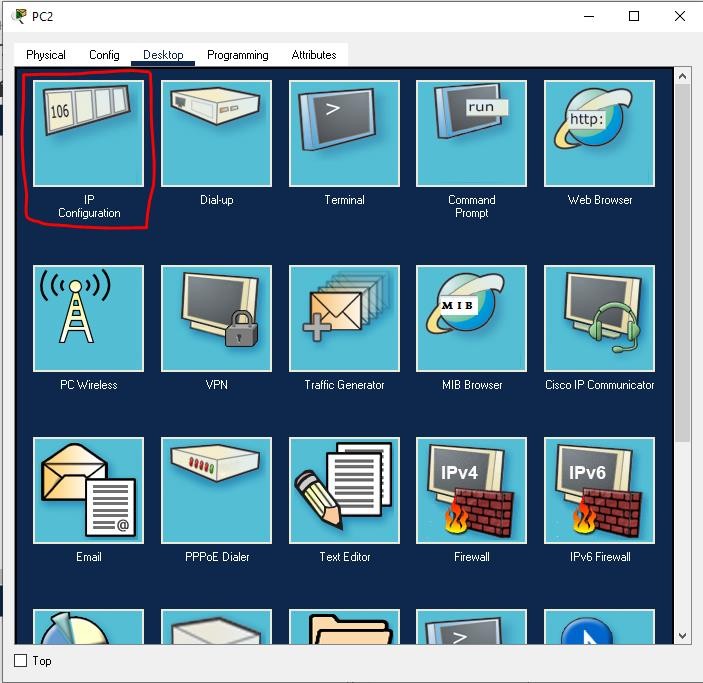
* Select connection icon as shown in below pic and connect switch and end systems with appropriate interface.



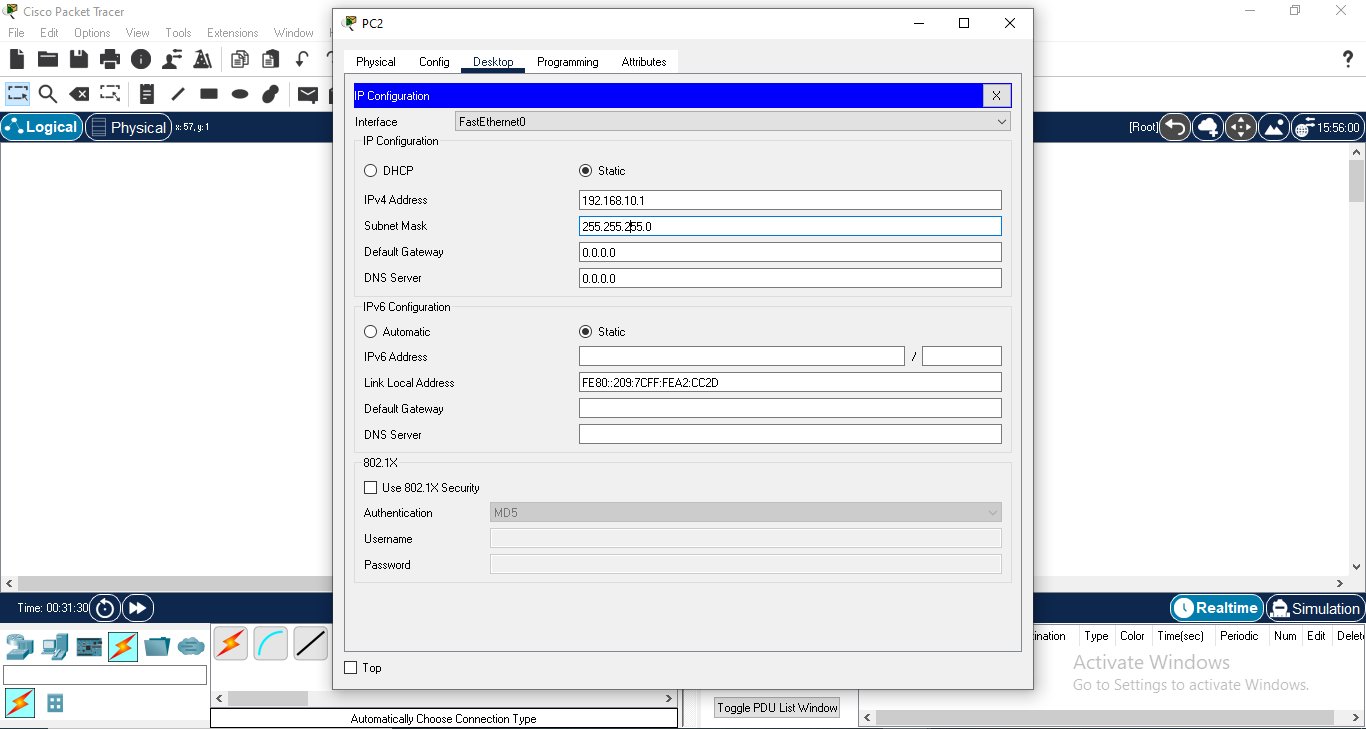
* Now configure both the end systems.
* Now double click on any of the PC and you should see the following window. Go to the Desktop tab.



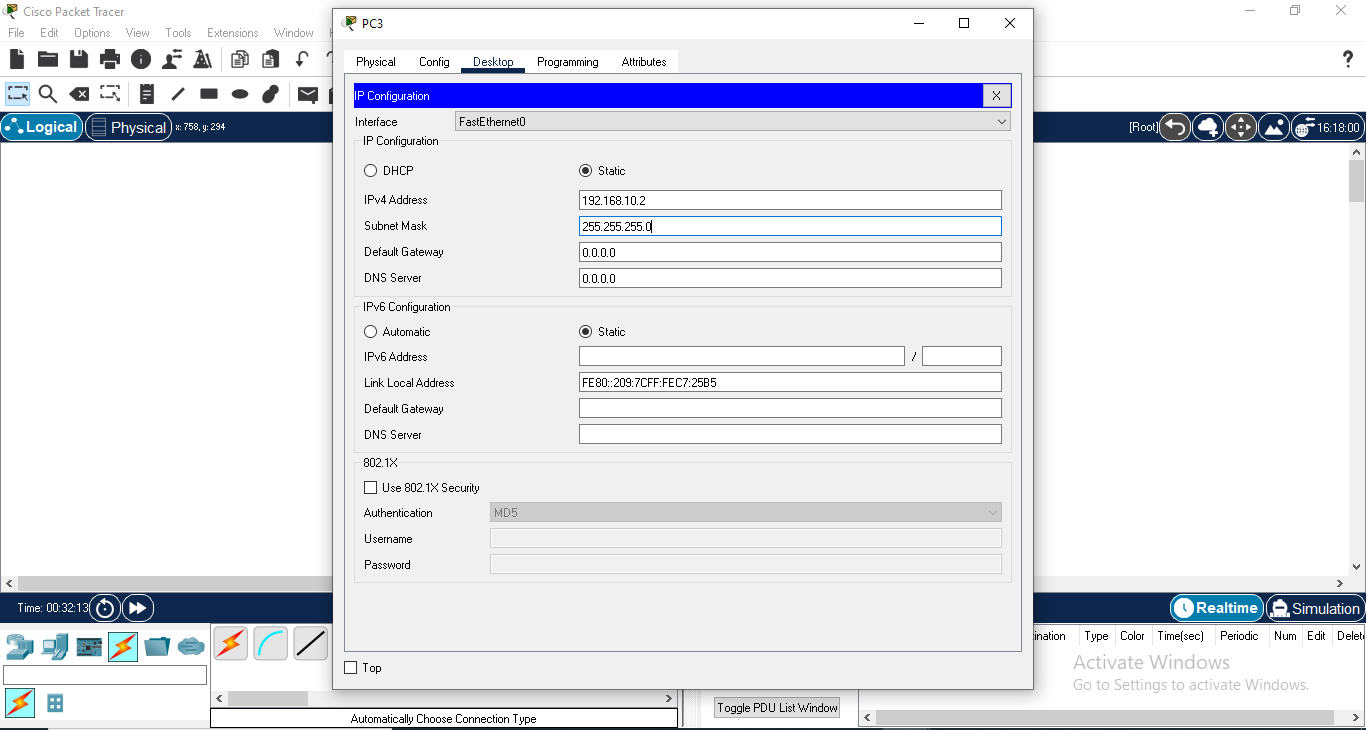
Now click on Configuration



* Now, fill in the IPv4 details as follows in one of the PCs and click on the X button once you’re done.

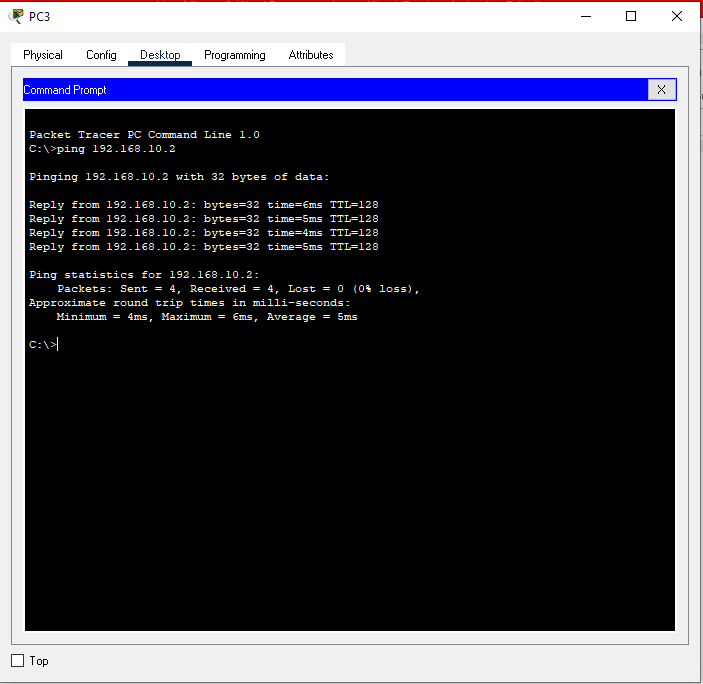


Same with second pc

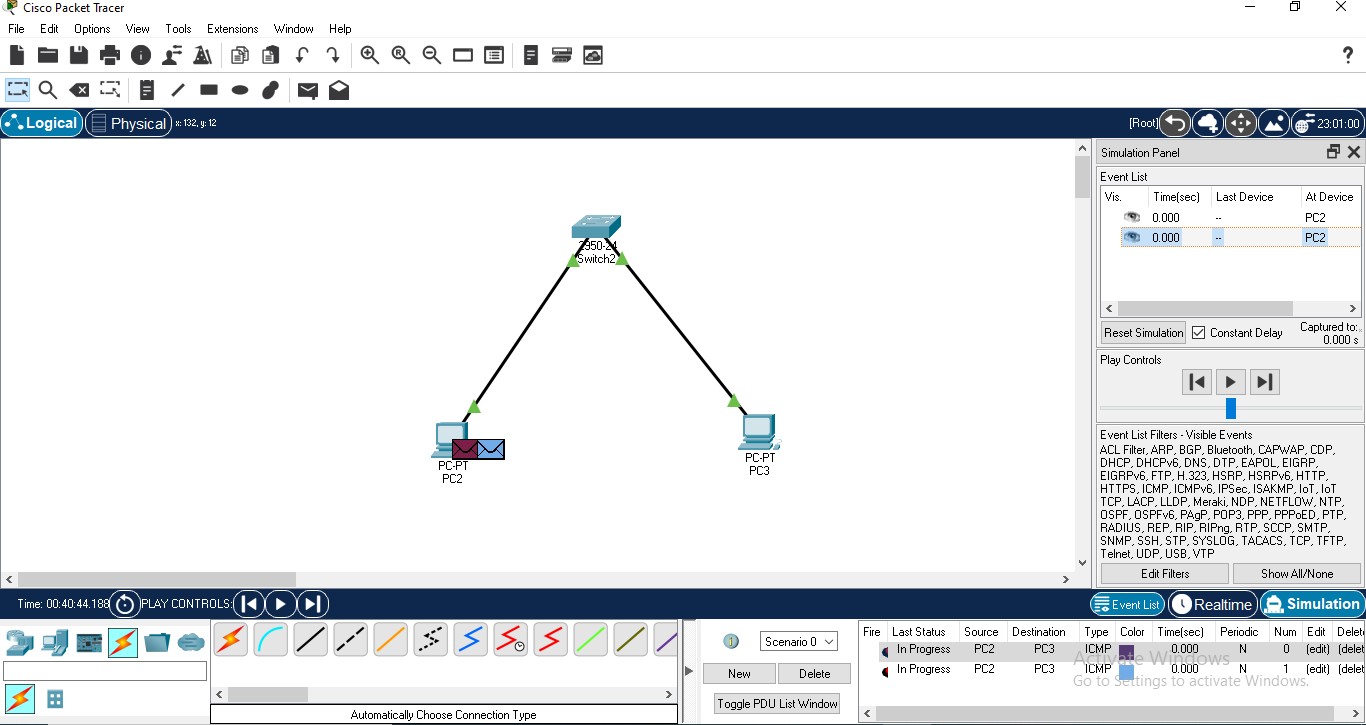


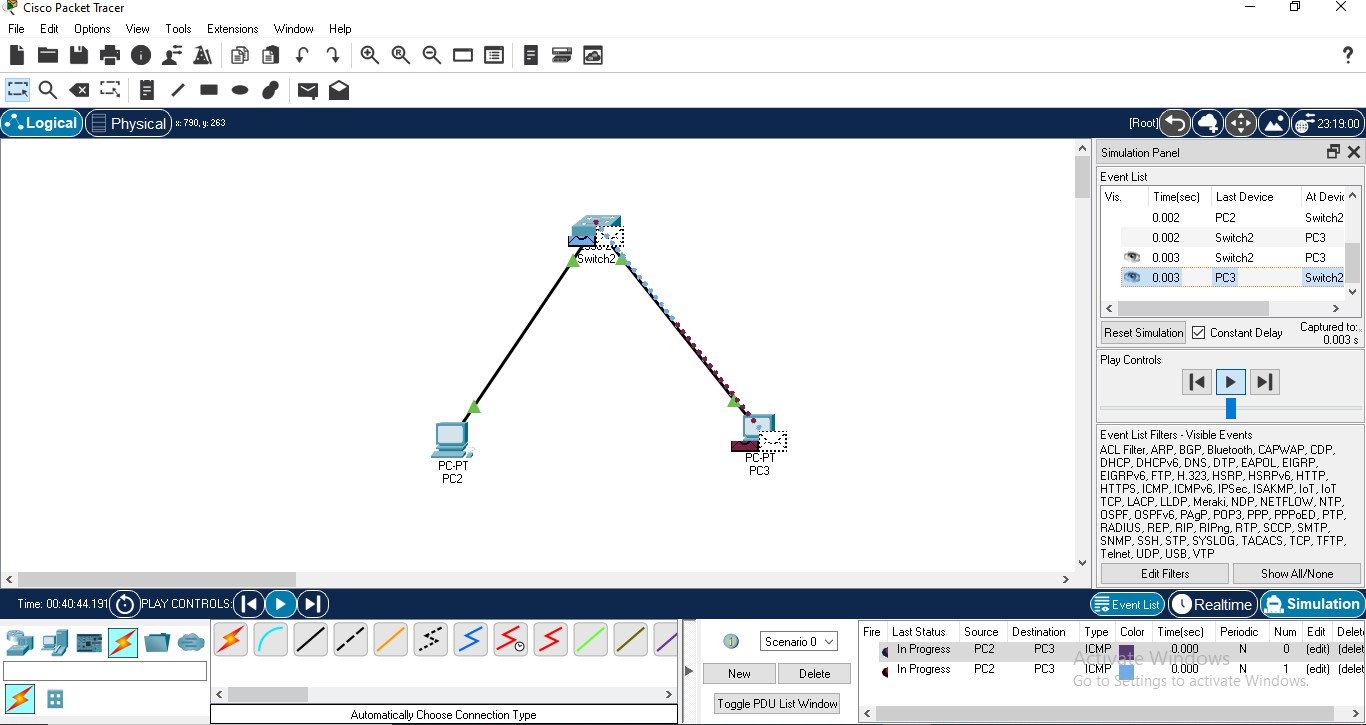
Now if we want to check if both the system are connected together we can select **desktop tab ->then command prompt**

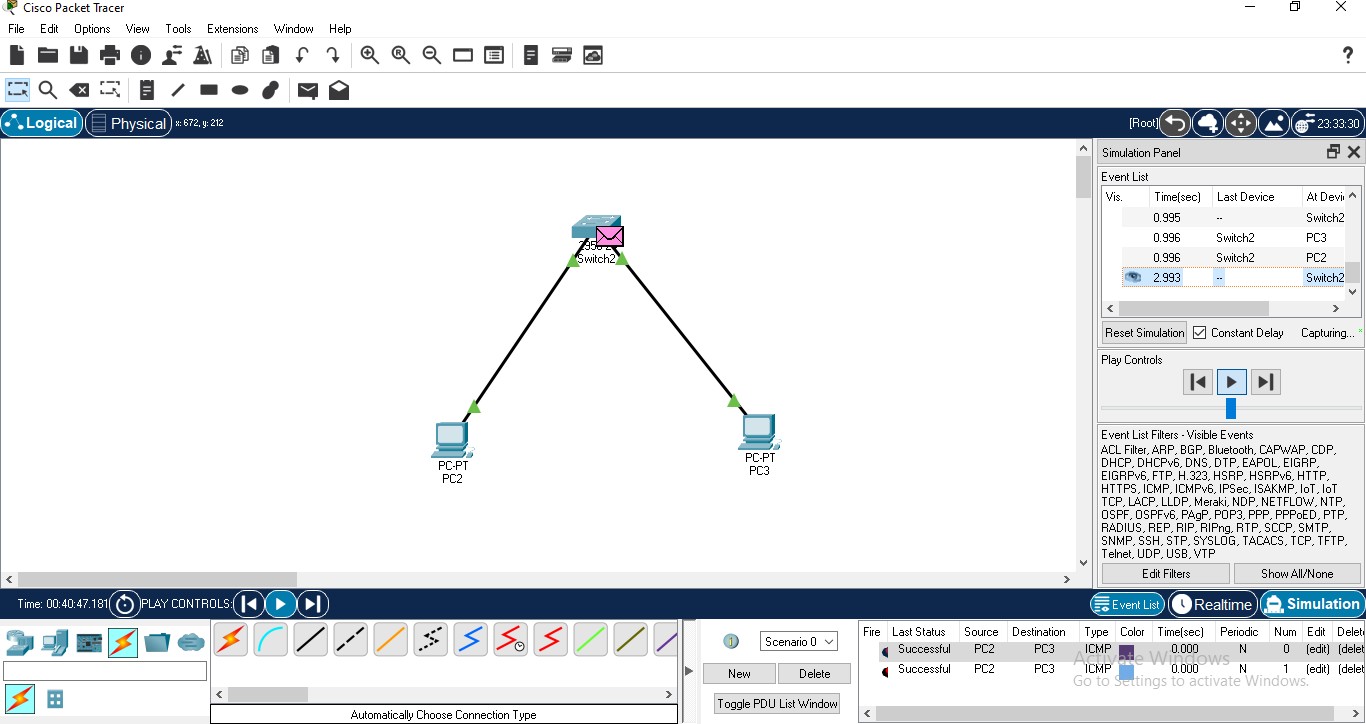
And type ping 192.168.10.2



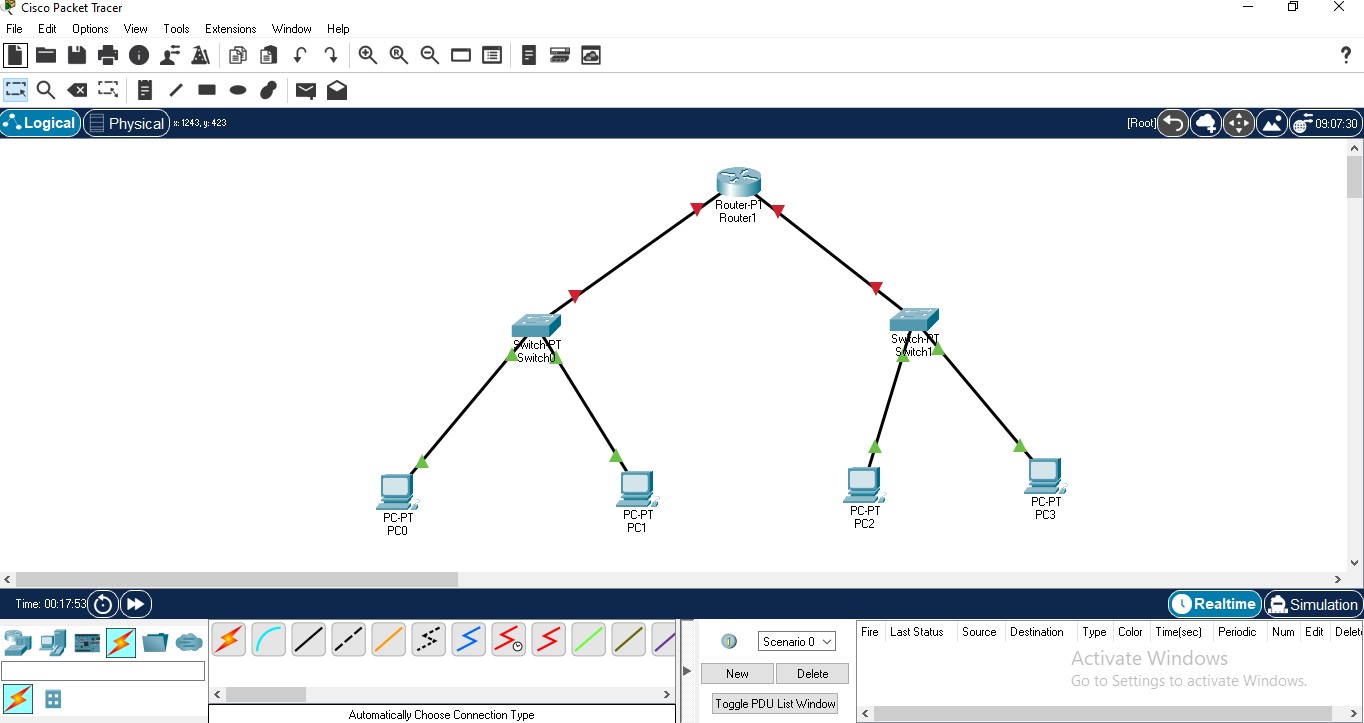
Select a packet and check whether it is delivered from PC0 to PC1







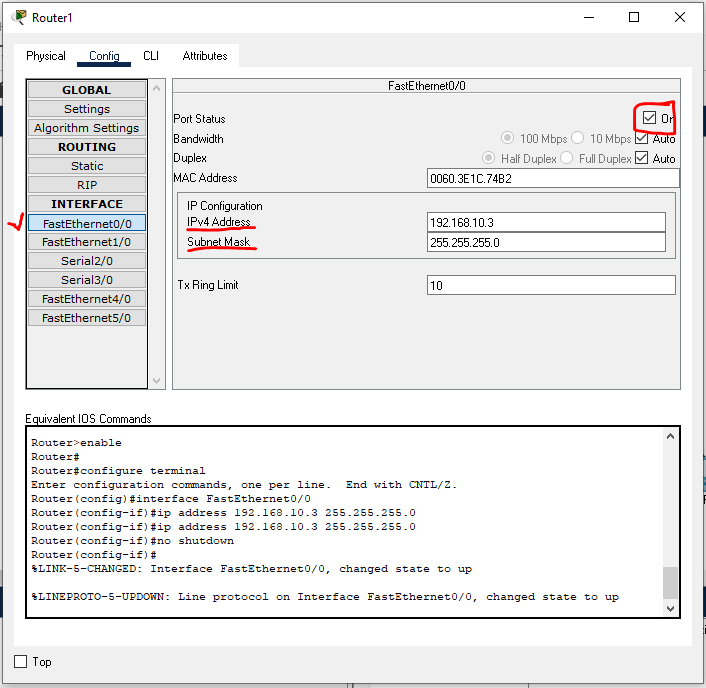
1. Basic Router configuration using CISCO packet tracer

Using Packet Tracer 8.0 : In this section, We will design a simple network topology and show you how Packet Tracer works. First, start Packet Tracer 8.0. Now click on Network Devices icon and then click on Switches icon and routers as per the screenshot below.

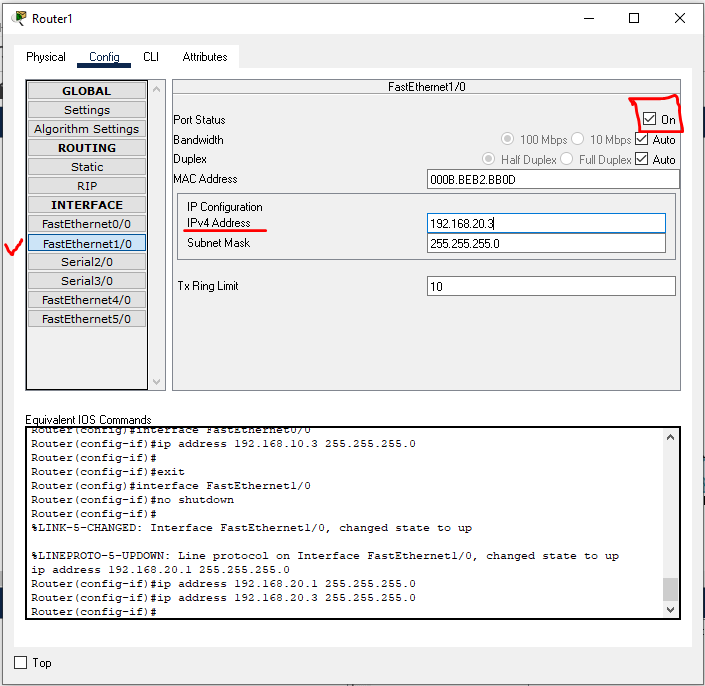
* Then select end devices icon and select two pc’s for each switch and connect all the end devices, switches and routers with appropriate interfaces.
* Now configure the topology
* For the router i.e Router 1 for the pc0 and pc1.

|  |  |  |  |
| --- | --- | --- | --- |
| o PC0   | Go to configuration window | and | set IP address and default |
|  | gateway address |  |  |
|  | IP address->192.168.10.1 |  |  |
|  | Subnet mask->255.255.255.0 |  |  |
|  | Gateway->192.168.10.3 |  |  |
| o PC1   | Go to configuration window | and | set IP address and default |
|  | gateway address |  |  |
|  | IP address->192.168.10.2 |  |  |
|  | Subnet mask->255.255.255.0 |  |  |
|  | Gateway->192.168.10.3 |  |  |

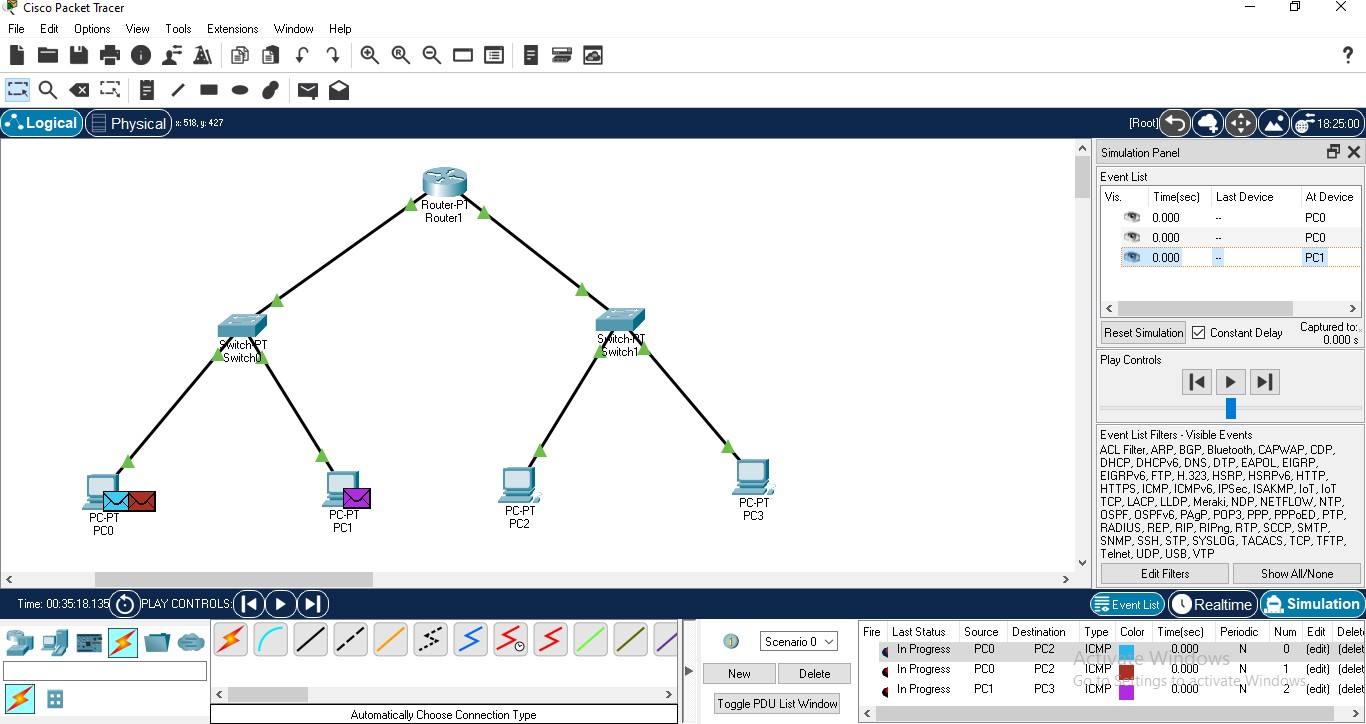
* + Configure Router 1
    - Click on router 1 go to config tab as shown in the screenshot below

 

* + - Set IP address of router for interface Fast Ethernet 0/0 i.e gateway given to both PC0 & PC1 ->192.168.10.3
    - And switch on the port as shown in the above screen shot i.e check the checkbox ”ON” at top rightmost .
* Similarly for Switch to PC2 and PC3
  + PC2
    - Go to configuration window and set IP address and default gateway address
    - IP address->192.168.20.1
    - Subnet mask->255.255.255.0
    - Gateway->192.168.20.3
  + PC3
    - Go to configuration window and set IP address and default gateway address
    - IP address->192.168.20.2
    - Subnet mask->255.255.255.0
    - Gateway->192.168.20.3
  + Configure Router 1
    - Click on router 1 go to config tab as shown in the screenshot below

o

Select a packet and check whether it is delivered from PC0 to PC2 and PC1 to PC3



* Simulate

