MANIPAL ACADEMY
OF HIGHER EDUCATION

(Deemed-to-be-University under Section 3 of the UGC Act, 1956)

Network Programming and Simulation Lab

(ICT 3165)

V Semester : B.Tech Information Technology (IT)

Year : August 2020

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

Contents

Course Objectives, Outcomes and Evaluation Plan

Course Objectives

- To learn client server socket programming.
- To learn network concepts using packet tracer.

Course Outcomes

- Implement client server socket programming.
- Simulate the network concepts using packet tracer.

Evaluation Plan

- Split up of 60 marks for Regular Lab Evaluation
  - Execution: 2 Marks
  - Record: 4 Marks
  - Evaluation: 4 Marks
- End Semester Lab evaluation: 40 marks (Duration 2 hrs)
  - Program write-up: 20 Marks
  - Program execution: 20 Marks

Instructions to Students

Pre-Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

1. Follow the instructions on the allotted exercises
2. Show the program and results to the instructors on completion of experiments
3. Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
  - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
  - Comments should be used to give the statement of the problem.
  - The statements within the program should be properly indented

- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- In case a student misses a lab, he/ she must ensure that the experiment is completed before the next evaluation with the permission of the faculty concerned.
- Students missing out the lab for genuine reasons like conference, sports or activities assigned by the Department or Institute will have to take prior permission from the HOD to attend additional lab (with another batch) and complete it before the student goes on leave. The student could be awarded marks for the write up for that day provided he submits it during the immediate next lab.
- Students who feel sick should get permission from the HOD for evaluating the lab records. However, attendance will not be given for that lab.
- Students will be evaluated only by the faculty with whom they are registered even though they carry out additional experiments in another batch.
- The presence of the student during the lab end semester exams is mandatory even if the student assumes he has scored enough to pass the examination
- Minimum attendance of 75
- If the student loses his book, he/she will have to rewrite all the lab details in the lab record.
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

The students should NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

Lab No. 1: Introduction to Socket Programming

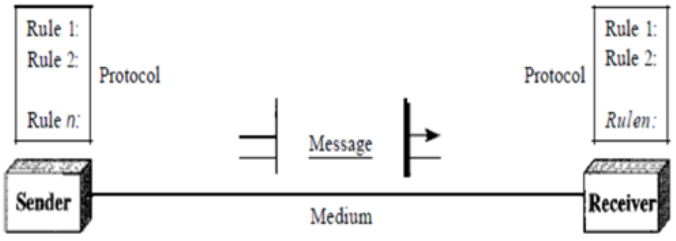
Objectives

- To illustrate the significance of socket programming
- To recognize basic socket function calls by performing simple client server operations

Introduction

Computer network is a communication network in which a collection of computers are connected together to facilitate data exchange .The connection between the computers can be wired or wireless. A computer network basically comprises of 5 components as shown in Figure.

- Sender
- Receiver
- Message
- Transmission medium
- Protocols



Components of data communication

Client Server Architecture

Communication in computer networks follows a client server model as illustrated in Figure. Here, a machine (referred as client) makes a request to connect to another machine (called as server) for providing some service. The services running on the server run on known ports (application identifiers) and the client needs to know the address of the server machine and this port in order to connect to the server. On the other hand, the server does not need to know about the address or the port of the client at the time of connection initiation. The first packet which the client sends as a request to the server contains these information about the client which are further used by the server to send any information. Client (Active Open) acts as the active device which makes the first move to establish the connection whereas the server (Passive Open) passively waits for such requests from some client.

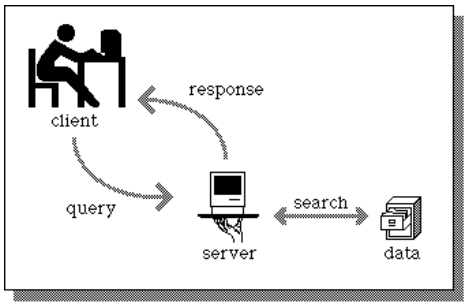


Illustration of Client Server Model

Basics of Socket Programming

In UNIX, whenever there is a need for inter process communication within the same machine, we use mechanism like signals or pipes. Similarly, when we desire a communication between two applications possibly running on different machines, we need sockets. A network socket is an endpoint for sending or receiving data at a single node in a computer network that supports full duplex transmission.

Sockets are created and used with a set of programming requests or ”function calls”

sometimes called the sockets Application Programming Interface (API). The most common sockets API is the Berkeley UNIX C interface for sockets which are the interfaces between applications layer and the transport layer that acts as a virtual connection between two processes. Each socket is identified by an address so that processes can connect to them.

Socket = IP Address + Port Number

Types of Communication Services

The data transfer between client and server application initiated by socket APIs can be achieved either by using connection oriented or connectionless services.

- Connection Oriented Communication :In connection oriented service a connection has to be established between two devices before starting the communication to allocate the resources needed to aid the data transfer. Then the message transfer takes place until the connection is released. This type of communication is characterized by a high level of reliability in terms of the number and the sequence of bytes. It is analogous to the telephone network.

- Connectionless Communication: In connectionless the data is transferred in one direction from source to destination without checking that destination is still there or not or if it prepared to accept the message. Authentication is not needed in this. It is analogous to the postal service where Packets (letters) are sent at a time to a particular destination.

Based on the two types of communication described above, two kinds of sockets are used:

- Stream sockets: used for connection-oriented communication, when reliability in connection is desired. Protocol used is TCP (Transmission Control Protocol) for data transmission.

- Datagram sockets: used for connectionless communication, when reliability is not as much as an issue compared to the cost of providing that reliability. For e.g.

Streaming audio/video is always sent over such sockets so as to diminish network traffic. Protocol used is UDP (User Datagram Protocol) for data transmission.

Socket System Calls for Connection-Oriented Protocol

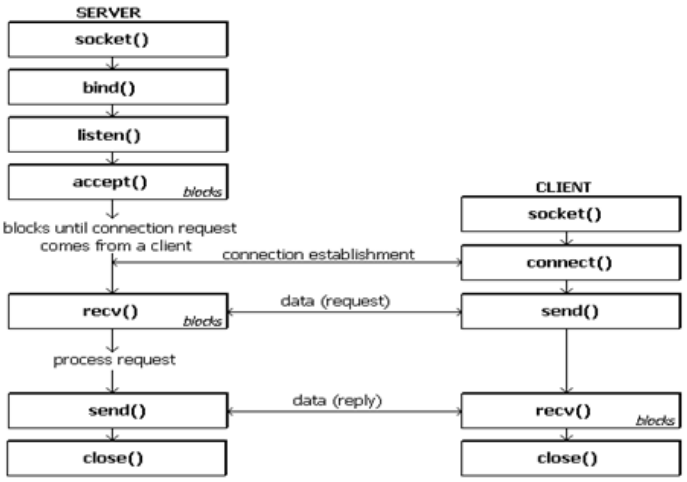
Steps followed by client to establish the connection:

1. Create a socket
2. Connect the socket to the address of the server
3. Send/Receive data
4. Close the socket connection

Steps followed by server to establish the connection:

1. Create a socket
2. Bind the socket to the port number known to all clients
3. Listen for the connection request
4. Accept connection request. This call typically blocks until a client connects with the server.
5. Send/Receive data
6. Close the socket connection

The sequence of socket function calls to be invoked for a connection oriented client server communication is depicted in Figure.



Connection-oriented Socket Structure

The significance of these function calls is summarized.

- Socket: Create a new communication request
- Bind: Attach a local address to a socket
- Listen: Announce willingness to accept connections
- Accept: Block caller until a connection request arrives
- Connect: Actively attempt to establish a connection
- Send: Send some data over the connection
- : Receive: Receive some data over the connection
- : Close: Release the connection

Socket System Calls for Connectionless Protocol

Steps of establishing a UDP socket communication on the client side are as follows:

1. Create a socket using the socket() function

2. Send and receive data by means of the recvfrom() and sendto() functions.

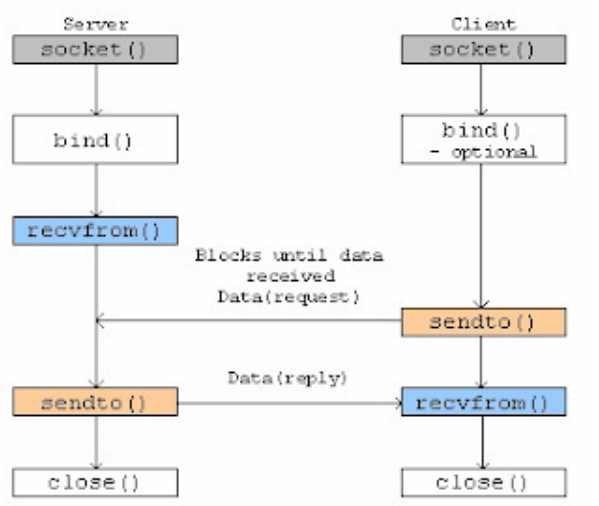
Steps of establishing a UDP socket communication on the server side are as follows:

1. Create a socket with the socket() function;

2. Bind the socket to an address using the bind() function;

3. Send and receive data by means of recvfrom() and sendto().

Figure shows the interaction between a UDP client and server. The client sends a datagram to the server using the sendto() function which requires the address of the destination as a parameter. Similarly, the server does not accept a connection from a client. Instead, the server calls the recvfrom() function, which waits until data arrives from some client. recvfrom() returns the IP address of the client, along with the datagram, so the server can send a response to the client.



Connectionless Socket Structure

Socket System Calls in detail

- Headers

1. sys/types.h: Defines the data type of socket address structure in unsigned long.

2. sys/socket.h: The socket functions can be defined as taking pointers to the generic socket address structure called sockaddr.
3. netinet/in.h: Defines the IPv4 socket address structure commonly called Internet socket address structure called sockaddr_in.

- Structures: Various structures are used in Unix Socket Programming to hold information about the address and port, and other information. Most socket functions require a pointer to a socket address structure as an argument.

1. Sockaddr: This is a generic socket address structure, which will be passed in most of the socket function calls. It holds the socket information. The Table provides a description of the member fields.

```
struct sockaddr {
unsigned short sa_family;
char sa_data[14];        };
```

Generic socket address structure fields

Attribute	Values	Description
sa_family	**AF_INET** **AF_UNIX** **AF_NS** **AF_IMPLINK**	**It represents an address family. In most of the Internet-based applications, we use AF_INET.**
sa_data	**Protocol-specific Address**	**The content of the 14 bytes of protocol specific address are interpreted according to the type of address. For the Internet family, we use port number IP address, which is represented by sockaddr_in structure.**

2. sockaddr_in: This helps to reference to the socket's elements and Table provides a description of the member fields.

```
struct sockaddr_in {
short int sin_family;
unsigned short int sin_port;
structin_addrsin_addr;
unsigned char sin_zero[8];       };
```

Structure fields of sockaddr_in

Attribute	**Values**	**Description**
sa_family	**AF_INET** **AF_UNIX** **AF_NS** **AF_IMPLINK**	**It represents an address family. In most of the Internet-based applications, we use AF_INET.**
sin_port	**Service Port**	**A 16-bit port number in Network Byte Order.**
sin_addr	**IP Address**	**A 32-bit IP address in Network Byte Order.**
sin_zero	**Not Used**	**This value could be set to NULL as this is not being used.**

3. in_addr: This structure is used only in the above structure as a structure field and holds 32 bit netid/hostid. The Table provides a description of the member fields.

```
struct in_addr {
unsigned long s_addr; };
```

Structure fields of in_addr

Attribute	Values	Description
s_addr	service port	A 32-bit IP address in Network Byte Order.

4. hostent: This structure is used to keep information related to host. The Table provides a description of the member fields.

```
struct hostent {
        char *h_name;
        char **h_aliases;
        int h_addrtype;
        int h_length;
        char **h_addr_list
        #define h_addrh_addr_list[0]   };
```

5. servent: This particular structure is used to keep information related to service and associated ports. The Table provides a description of the member fields.

Structure fields of hostent

Attribute	Values	Description
h_name	ti.com etc.	It is the official name of the host. For example, tutorialspoint.com, google.com, etc.
h_aliases	TI	It holds a list of host name aliases.
h_addrtype	AF_INET	It contains the address family and in case of Internet based application, it will always be AF_INET.
h_length	4	It holds the length of the IP address, which is 4 for Internet Address.
h_addr_list	in_addr	For Internet addresses, the array of pointers h_addr_list[0], h_addr_list[1], and so on, are points to structure in_addr.

NOTE: h_addr is defined as h_addr_list[0] to keep backward compatibility.

```
struct servent {
char *s_name;
char **s_aliases;
int s_port;
char *s_proto;
};
```

Structure fields of servent

Attribute	**Values**	**Description**
s_name	**http**	**This is the official name of the service. For example, SMTP, FTP POP3, etc.**
s_aliases	**ALIAS**	**It holds the list of service aliases. Most of the time this will be set to NULL.**
s_port	**80**	**It will have associated port number. For example, for HTTP, this will be 80.**
s_proto	**TCP** **UDP**	**It is set to the protocol used. Internet services are provided using either TCP or UDP.**

- Ports and Services:

  To resolve the problem of identifying a particular server process running on a host, both TCP and UDP have defined a group of ports. A port is always associated with an IP address of a host and the protocol type of the communication, and thus completes the destination or origination address of a communication session. A port is identified

for each address and protocol by a 16-bit number, commonly known as the port number. Specific port numbers are often used to identify specific services as listed below.

1. Ports 0-1023 (well-known ports): reserved for privileged services like for ftp: 21 and for telnet: 23.
2. Ports 1024-49151 (registered ports): vendors use for some applications.
3. Ports above 49151 (dynamic/ephemeral/private ports): short-lived transport protocol port for Internet Protocol (IP) communications allocated automatically from a predefined range by the IP stack software. After communication is terminated, the port becomes available for use in another session. However, it is usually reused only after the entire port range is used up.

- Constructing Messages - Byte Ordering: In computers, addresses and port numbers are stored as integers of type:
  - u_short sin_port; (16 bit)
  - in_addr sin_addr; (32 bit)

  Unfortunately, not all computers store the bytes that comprise a multibyte value in the same order. There are 2 types of byte ordering.
  - Little Endian (Host byte order)- in this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next higher address (A + 1).
  - Big Endian (Network byte order)- in this scheme, high-order byte is stored on the starting address (A) and low-order byte is stored on the next higher address (A + 1).

  To allow machines with different byte order conventions communicate with each other, the internet protocols specify a canonical byte order convention for data

Byte order conversion function description

Function	Description
htons()	**Host to Network Short (16 bit)**
htonl()	**Host to Network Long (32 bit)**
ntohl()	**Network to Host Long (32 bit)**
ntohs()	**Network to Host Short (16 bit)**

transmitted over the network. This is known as Network Byte Order. Table describes some of the byte order functions available in UNIX programming.

- IP Address Functions: These functions convert Internet addresses between ASCII strings (what humans prefer to use) and network byte ordered binary values (values that are stored in socket address structures). Some of the commonly used IP address functions are described below.

  1. int inet_aton(const char *strptr, struct in_addr *addrptr): This function call converts the specified string in the Internet standard dot notation to a network address, and stores the address in the structure provided. The converted address will be in Network Byte Order (bytes ordered from left to right). It returns 1 if the string was valid and 0 on error.

     Following is the usage example:

     ```
     #include <arpa/inet.h>
     (...)
     int retval;
     struct in_addr addrptr
     memset(&addrptr, '\0', sizeof(addrptr));
     retval = inet_aton("68.178.157.132", &addrptr);
     (...)
     ```

  2. in_addr_tinet_addr(const char *strptr): This function call converts the specified string in the Internet standard dot notation to an integer value suitable for use

as an Internet address. The converted address will be in Network Byte Order (bytes ordered from left to right). It returns a 32-bit binary network byte ordered IPv4 address and INADDR_NONE on error.

Following is the usage example:

```
#include <arpa/inet.h>
(...)
struct sockaddr_ indest;
memset(&dest, '\0', sizeof(dest));
dest.sin_addr.s_addr = inet_addr("68.178.157.132");
(...)
```

3. char *inet_ntoa(struct in_addr inaddr): This function call converts the specified Internet host address to a string in the Internet standard dot notation.

   Following is the usage example:

```
#include <arpa/inet.h>
char *ip;
ip = inet_ntoa(dest.sin_addr);
printf("IP Address is: %s\n",ip);
```

Programming Functions

1. Socket function: int socket (int family, int type, int protocol); This call returns a socket descriptor that you can use in later system calls or -1 on error.

   Parameters:

   - family: specifies the protocol family and is one of the constants as given n Table.
   - type: It specifies the kind of socket you want. The possible socket types are listed in Table.
   - protocol: The argument should be set to the specific protocol type given below,

Different socket address family

Family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing Sockets
AF_KEY	Ket socket

Socket Types

Type	Description
SOCK_STREAM	Stream socket
SOCK_DGRAM	Datagram socket
SOCK_SEQPACKET	Sequenced packet socket
SOCK_RAW	Raw socket

or 0 to select the system's default for the given combination of family and type .The default protocol for SOCK_STREAM with AF_INET family is TCP. Other protocol description is given in Table.

Different protocols

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

- Example:

```
if ( (sd = socket( AF_INET, SOCK_STREAM, 0 )) < 0 ) {
cout <<"Socket creation error";
exit(-1);
}
```

2. Connect: The connect function is used by a TCP client to establish a connection with a TCP server. This call normally blocks until either the connection is established or is rejected.

   int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

   This call returns 0 if it successfully connects to the server, otherwise it returns -1 on

error.

Parameters:

- sockfd: it is a socket descriptor returned by the socket function.
- serv_addr: it is a pointer to struct sockaddr that contains destination (Server) IP address and port.
- addrlen: the length of the address structure pointed to by servaddr. Set it to sizeof(structsockaddr).

3. Bind: The bind function assigns a local protocol address to a socket. With the Internet protocols, the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number. This function is called by TCP server only. bind() allows to specify the IP address, the port, both or neither. The Table summarizes the combinations for IPv4.
   int bind(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
   This call returns 0 if it successfully binds to the address, otherwise it returns -1 on error.
   Parameters:
   - sockfd: it is a socket descriptor returned by the socket function.
   - servaddr: it is a pointer to struct sockaddr that contains the local IP address and port.
   - addrlen: Set it to sizeof(structsockaddr).
     The ¡sys/socket.h¿ header shall define the type socklen_t, which is an integer type of width of at least 32 bits.

IP address and port number combinations

IP Address	IP Port	Result
INADDR_ANY	0	Kernel chooses IP address and port
INADDR_ANY	non zero	Kernel chooses IP address, process specifies port
Local IP address	0	Process specifies IP address, kernel chooses port
Local IP address	non zero	Process specifies IP address and port

4. Listen: The listen function is called only by a TCP server. It converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
   int listen(int sockfd,int backlog);
   This call returns 0 on success, otherwise it returns -1 on error.
   Parameters:
   - sockfd: it is a socket descriptor returned by the socket function.
   - backlog: it is the maximum number of connections the kernel should queue for this socket.

5. Accept: The accept function is called by a TCP server to return the next completed connection from the front of the completed connection queue.
   int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
   This call returns a non-negative descriptor on success, otherwise it returns -1 on error. The returned descriptor is assumed to be a client socket descriptor and all read-write operations will be done on this descriptor to communicate with the client.
   Parameters:
   - sockfd: It is a socket descriptor returned by the socket function.
   - cliaddr: It is a pointer to structsockaddr that contains client IP address and port.
   - addrlen: Set it to sizeof(structsockaddr).

6. Send: The send function is used to send data over stream sockets or CONNECTED datagram sockets. If you want to send data over UNCONNECTED datagram sockets, you must use sendto() function.
   int send(int sockfd, const void *msg, int len, int flags);
   This call returns the number of bytes sent out, otherwise it will return -1 on error.
   Parameters:
   - sockfd: it is a socket descriptor returned by the socket function.
   - msg: it is a pointer to the data you want to send.

- len: it is the length of the data you want to send (in bytes).
- flags: it is set to 0.The additional argument flags is used to specify how we want the data to be transmitted.

7. Recv: The recv function is used to receive data over stream sockets or CONNECTED datagram sockets. If you want to receive data over UNCONNECTED datagram sockets you must use recvfrom().
   int recv(int sockfd, void *buf, int len, unsigned int flags);
   This call returns the number of bytes read into the buffer, otherwise it will return -1 on error.
   Parameters:
   - sockfd: it is a socket descriptor returned by the socket function.
   - buf: it is the buffer to read the information into.
   - len: it is the maximum length of the buffer.
   - flags: it is set to 0. The additional argument flags is used to specify how we want the data to be transmitted.

8. Sendto: The sendto function is used to send data over UNCONNECTED datagram sockets. Upon successful completion, sendto() shall return the number of bytes sent. Otherwise, -1 shall be returned and errno set to indicate the error.
   int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);
   Parameters:
   - sockfd: it is a socket descriptor returned by the socket function.
   - msg: it is a pointer to the data you want to send.
   - len: it is the length of the data you want to send (in bytes).
   - flags: it is set to 0.
   - to: it is a pointer to struct sockaddr for the host where data has to be sent.

- tolen: it is set it to sizeof(struct sockaddr).

9. Recvfrom: The recvfrom function is used to receive data from UNCONNECTED datagram sockets. Upon successful completion, recvfrom() shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recvfrom() shall return 0. Otherwise, the function shall return -1 and set errno to indicate the error.

   size_t recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

   Parameters:

   - sockfd: it is a socket descriptor returned by the socket function.
   - buf: it is the buffer to read the information into.
   - len: it is the maximum length of the buffer.
   - flags: it is set to 0.
   - from: it is a pointer to structsockaddr for the host where data has to be read.
   - fromlen: it is set it to sizeof(structsockaddr).
   - ssize_t: this data type is used to represent the sizes of blocks that can be read or written in a single operation. It is similar to size_t, but must be a signed type.

10. Close: The close function is used to close the communication between the client and the server.

    int close(int sockfd);

    This call returns 0 on success, otherwise it returns -1 on error.

    Parameters:

    - sockfd: it is a socket descriptor returned by the socket function.

11. Shutdown: The shutdown function is used to gracefully close the communication between the client and the server. This function gives more control in comparison to the close function.

int shutdown(int sockfd, int how);

This call returns 0 on success, otherwise it returns -1 on error.

Parameters:

- sockfd: it is a socket descriptor returned by the socket function.
- how: Put one of the numbers:
  - 0 : indicates that receiving is not allowed,
  - 1 : indicates that sending is not allowed, and
  - 2 : indicates that both sending and receiving are not allowed. When 'how' is set to 2, it's the same thing as close().

Sample exercise

Algorithm to implement an UDP Echo Client/Server Communication

Server:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
3. Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to SERVER_PORT.
4. Bind the local host address to socket using the bind function.
5. Within an infinite loop, receive message from the client using recvfrom function, print it on the console and send (echo) the message back to the client using sendto function.

Client:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.

3. Initialize the socket parameters. Assign the sin_family to AF_INET, sin_addr to "127.0.0.1", sin_port to dynamically assigned port number.

4. Within an infinite loop, read message from the console and send the message to the server using the sendto function.

5. Receive the echo message using the recvfrom function and print it on the console.

A simple UPD client-server program where a client connects to the server. Server sends a message to the client which is displayed at the client side.

Server Code

```
#include<string.h>
#include<unistd.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<stdio.h>
main()
{
int s,r,recb,sntb,x;
int ca;
printf("INPUT port number: ");
scanf("%d", &x);
socklen_t len;
struct sockaddr_in server,client;
char buff[50];
s=socket(AF_INET,SOCK_DGRAM,0);
if(s==-1)
{
printf("\nSocket creation error.");
```

```
exit(0);
}
printf("\nSocket created.");
server.sin_family=AF_INET;
server.sin_port=htons(x);
server.sin_addr.s_addr=htonl(INADDR_ANY);
len=sizeof(client);
ca=sizeof(client);
r=bind(s,(struct sockaddr*)&server,sizeof(server));
if(r==-1)
{
printf("\nBinding error.");
exit(0);
}
printf("\nSocket binded.");
while(1){
recb=recvfrom(s,buff,sizeof(buff),0,(struct sockaddr*)&client,&ca);
if(recb==-1)
{
printf("\nMessage Recieving Failed");
close(s);
exit(0);
}
printf("\nMessage Recieved: ");
printf("%s", buff);
if(!strcmp(buff,"stop"))
break;
printf("\n\n");
printf("Type Message: ");
```

```
scanf("%s", buff);
sntb=sendto(s,buff,sizeof(buff),0,(struct sockaddr*)&client,len);
if(sntb==-1)
{
printf("\nMessage Sending Failed");
close(s);
exit(0);
}
if(!strcmp(buff,"stop"))
break;
}
close(s);
}
```

Client Code

```
#include<string.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<fcntl.h>
#include<sys/stat.h>
main()
{
int s,r,recb,sntb,x;
int sa;
socklen_t len;
```

```
printf("INPUT port number: ");
scanf("%d", &x);
struct sockaddr_in server,client;
char buff[50];
s=socket(AF_INET,SOCK_DGRAM,0);
if(s==-1)
{
printf("\nSocket creation error.");
exit(0);
}
printf("\nSocket created.");
server.sin_family=AF_INET;
server.sin_port=htons(x);
server.sin_addr.s_addr=inet_addr("127.0.0.1");
sa=sizeof(server);
len=sizeof(server);
while(1){
printf("\n\n");
printf("Type Message: ");
scanf("%s", buff);
sntb=sendto(s,buff,sizeof(buff),0,(struct sockaddr *)&server, len);
if(sntb==-1)
{
close(s);
printf("\nMessage sending Failed");
exit(0);
}
if(!strcmp(buff,"stop"))
break;
```

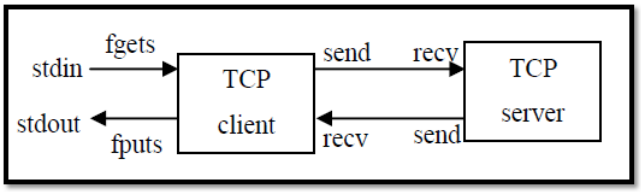
```
recb=recvfrom(s,buff,sizeof(buff),0,(struct sockaddr *)&server,&sa);
if(recb==-1)
{
printf("\nMessage Recieving Failed");
close(s);
exit(0);
}
printf("\nMessage Recieved: ");
printf("%s", buff);
if(!strcmp(buff,"stop"))
break;
}
close(s);
}
```

TCP Echo Client/Server Communication:

An echo client/server program performs the following:

1. The client reads a line of text from the standard input and writes the line to the server using send() function.
2. The server reads the line from the network input using recv() and echoes the line back to the client using send() function.
3. The client reads the echoed line using recv() and prints it on its standard output.

Figure depicts the echo client/server along with the functions used for input and output.



Echo client/server

A simple TCP client-server program where a client connects to the server. Server sends a message to the client which is displayed at the client side.

Client Code

```
#include<string.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<fcntl.h>
#include<sys/stat.h>
main()
{
int s,r,recb,sntb,x;
printf("INPUT port number: ");
scanf("%d", &x);
struct sockaddr_in server;
char buff[50];
s=socket(AF_INET,SOCK_STREAM,0);
if(s==-1)
{
printf("\nSocket creation error.");
exit(0);
}
printf("\nSocket created.");
server.sin_family=AF_INET;
server.sin_port=htons(x);
```

```
server.sin_addr.s_addr=inet_addr("127.0.0.1");
r=connect(s,(struct sockaddr*)&server,sizeof(server));
if(r==-1)
{
printf("\nConnection error.");
exit(0);
}
printf("\nSocket connected.");
printf("\n\n");
printf("Type Message: ");
scanf("%s", buff);
sntb=send(s,buff,sizeof(buff),0);
if(sntb==-1)
{
close(s);
printf("\nMessage Sending Failed");
exit(0);
}
recb=recv(s,buff,sizeof(buff),0);
if(recb==-1)
{
printf("\nMessage Recieving Failed");
close(s);
exit(0);
}
printf("\nMessage Recieved: ");
printf("%s", buff);
printf("\n\n");
close(s);
```

```
}
```

Server Code

```
#include<string.h>
#include<unistd.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<stdio.h>
main()
{
int s,r,recb,sntb,x,ns,a=0;
printf("INPUT port number: ");
scanf("%d", &x);
socklen_t len;
struct sockaddr_in server,client;
char buff[50];
s=socket(AF_INET,SOCK_STREAM,0);
if(s==-1)
{
printf("\nSocket creation error.");
exit(0);
}
printf("\nSocket created.");
server.sin_family=AF_INET;
server.sin_port=htons(x);
server.sin_addr.s_addr=htonl(INADDR_ANY);
r=bind(s,(struct sockaddr*)&server,sizeof(server));
if(r==-1)
```

```
{
printf("\nBinding error.");
exit(0);
}
printf("\nSocket binded.");
r=listen(s,1);
if(r==-1)
{
close(s);
exit(0);
}
printf("\nSocket listening.");
len=sizeof(client);
ns=accept(s,(struct sockaddr*)&client, &len);
if(ns==-1)
{
close(s);
exit(0);
}
printf("\nSocket accepting.");
recb=recv(ns,buff,sizeof(buff),0);
if(recb==-1)
{
printf("\nMessage Recieving Failed");
close(s);
close(ns);
exit(0);
}
printf("\nMessage Recieved: ");
```

```
printf("%s", buff);
printf("\n\n");
scanf("%s", buff);
sntb=send(ns,buff,sizeof(buff),0);
if(sntb==-1)
{
printf("\nMessage Sending Failed");
close(s);
close(ns);
exit(0);
}
close(ns);
close(s);
}
```

Steps to execute the program:

1. Open two terminal windows and open a text file from each terminal with .c extension using command: gedit filename.c

2. Type the client and server program in separate text files and save it before exiting the text window.

3. First compile and run the server using commands mentioned below
   a. gcc -o filename
   b. ./a.out or ./filename

4. Compile and run the client using the same instructions as listed in 3a and 3b

Note: The ephemeral port number has to be changed every time the program is executed.

Lab Exercises:

1. Write two separate C programs (one for server and other for client) using socket APIs for TCP, to implement the client-server model such that the client should send a set of integers along with a choice to search for a number or sort the given set in ascending/descending order or split the given set to odd and even to the server. The server performs the relevant operation according to the choice. Client should continue to send the messages until the user enters selects the choice "exit".

2. Write two separate C programs (one for server and other for client) using UNIX socket APIs for UDP, in which the client accepts a string from the user and sends it to the server. The server will check if the string is palindrome or not and send the result with the length of the string and the number of occurrences of each vowel in the string to the client. The client displays the received data on the client screen. The process repeats until user enter the string "Halt". Then both the processes terminate. (The program should make use of TCP and UDP separately).

Additional Exercise:

1. Write two separate C programs (one for the Server and the other for Client) using UNIX socket APIs using both connection oriented and connectionless services, in which the server displays the client's socket address, IP address and port number on the server screen.

Lab No. 2: File Operations using Socket Programming

Objectives

- To illustrate file operations like open, read and write.
- To exemplify the FTP application in a client server communication.

Introduction

The File Transfer Protocol (FTP) is a standard network protocol used to transfer computer files between a client and server on a computer network. In this lab, we learn to implement FTP application in a client server architecture, where the Client on establishing a connection with the Server, sends the name of the file it wishes to access remotely. The Server then sends the contents of the file to the Client. An algorithm to implement this scenario is given below.

Algorithm (TCP)

Server:

1. Include necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize the socket and set its attributes. Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
4. Bind the server to socket using bind function.

5. Listen to incoming client requests and wait for the client request. On connection request from the client establish a connection using accept() function.
6. Read the files name sent by the client.
7. Open the file, read the file contents to a buffer and send the buffer to the Client.
8. After reading till the end of the file, close the file.
9. Close the connection.

Client

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize the socket and set its attribute set. Assign the sin_family to AF_INET, sin_addr to "127.0.0.1", sin_port to dynamically assigned port number.
4. Connect to server using connect () function to initiate the request.
5. Send the file names to server.
6. Receive the buffer from the server and print its contents at the console.
7. Close the connection.

Algorithm (UDP)

Server:

1. Include necessary header files
2. Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.

3. Initialize the socket and set its attributes.Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.

4. Read the files name sent by the client using recvfrom() function.

5. Open the file, read the file contents to a buffer and send the buffer to the Client using sendto() function.

6. Close the connection.

Client:

1. Include the necessary header files.

2. Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.

3. Initialize the socket and set its attribute set. Assign the sin_family to AF_INET, sin_addr to "127.0.0.1", sin_port to dynamically assigned port number.

4. Send the file names to server using sendto function.

5. Receive the buffer from the server using recvfrom function and print its contents at the console.

6. Close the connection.

Lab Exercises

1. Write two separate C programs (one for server and other for client) using UNIX socket APIs for both TCP and UDP to implement the following: The user at the client side sends name of the file to the server. If the file is not present, the server sends "File not present" to the client and terminates. Otherwise the following menu is displayed at the client side.
   1. Search 2. Replace 3. Reorder 4. Exit

- If the user at the client side wants to search a string in file, the users sends to the server option '1' along with the string to be searched. The server searches for the string in the file. If present, it sends the number of times the string has occurred to the client, else the server sends 'String not found' message to the client.
- If the user wants to replace a string, along with option 2, the two strings 'str1' and 'str2' are sent to the server. The Server searches for str1 in the file and replaces it with 'str2'. After replacing the string, 'String replaced' message is sent to the client. If it is not found 'String not found' command is sent to the client.
- Option 3 rearranges the entire text in the file in increasing order order of their ASCII value.
- To terminate the application option '4' is selected

Additional Exercise:

1. Write two separate C programs (one for server and other for client) using socket APIs for TCP and UDP, to implement the File Server. The client program will send the name of the text file to the server. If the file is present at the server side, the server should send the contents of the file to the client along with the file size, number of alphabets number of lines, number of spaces, number of digits, and number of other characters present in the text file to the client. If the file is not present, then the server should send the proper message to the client. Note that the results are always displayed at the client side. Client should continue to send the filenames until the user enters the string 'stop'.

Lab No. 3: Chat Server using Socket Programming

Objectives

1. To contrast different modes of communication by implementing half duplex and full duplex chat server in connectionless and connection oriented environment.
2. To illustrate the use of fork() system call in socket programming.

Introduction

Writing a chat application with popular web applications stacks has traditionally been very hard. It involves polling the server for changes, keeping track of timestamps, and it's a lot slower than it should be. Sockets have traditionally been the solution around which most real-time chat systems are architected, providing a bi-directional communication channel between a client and a server.

A chat server is a computer dedicated to providing the processing power to handle and maintain chatting and it users. This means that the server can push messages to clients and also support multiple clients to initiate the conversation. To develop a chat server model, it is important to understand the different types of server and the mode of communication supported.

Basic Modes of Communication

Data communication can be either simplex, half duplex or full duplex which are described below.

1. Simplex Operation: In simplex operation, a network cable or communications channel can only send information in one direction; it's a "one-way street". Simplex operation is used in special types of technologies, especially ones that are

asymmetric. For example, one type of satellite Internet access sends data over the satellite only for downloads, while a regular dial-up modem is used for upload to the service provider. In this case, both the satellite link and the dial-up connection are operating in a simplex mode.

2. Half-Duplex Operation: Technologies that employ half-duplex operation are capable of sending information in both directions between two nodes, but only one direction or the other can be utilized at a time. This is a fairly common mode of operation when there is only a single network medium (cable, radio frequency and so forth) between devices. For example, in conventional Ethernet networks, any device can transmit, but only one may do so at a time. For this reason, regular Ethernet networks are often said to be "half-duplex".

3. Full-Duplex Operation: In full-duplex operation, a connection between two devices is capable of sending data in both directions simultaneously. Full-duplex channels can be constructed either as a pair of simplex links or using one channel designed to permit bidirectional simultaneous transmissions. A full-duplex link can only connect two devices, so many such links are required if multiple devices are to be connected together.

Types of Server

There are two types of servers.

- Iterative Server - this is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting. In other words, an iterative server iterates through each client, handling it one at a time.

- Concurrent Servers - this type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under UNIX

is to fork a child process to handle each client separately. An alternative technique is to use threads instead (i.e., light-weight processes).

Some System Calls

- fork() causes a process to duplicate. The child process is an almost-exact duplicate of the original parent process; it inherits a copy of its parent's code, data, stack, open file descriptors, and signal table. However, the parent and child have different process id numbers and parent process id numbers.
- If fork() succeeds, it returns the PID of the child to the parent process, and returns 0 to the child process. If it fails, it returns -1 to the parent process and no child is created.

System Calls:

int getpid()

int getppid()

getpid() and getppid() return a process's id and parent process's id numbers, respectively. They always succeed. The parent process id number of PID 1 is 1.

Simple fork example to display PID and PPID

```
#include <stdio.h>
main()
{
int pid;
printf("I'm the original process with PID %d and PPID %d.\n",
getpid(),getppid());
pid=fork();
/* Duplicate. Child and parent continue from here.*/
if (pid!=0)  /* pid is non-zero, so I must be the parent  */
{
```

```
printf("I'm the parent process with PID %d and PPID %d.\n",
getpid(),getppid());
printf("My child's PID is %d.\n", pid);
}
else  /* pid is zero, so I must be the child. */
{
printf("I'm the child process with PID %d and PPID %d.\n",
getpid(),getppid());
}
printf("PID %d terminates.\n",pid);
/* Both processes execute this */
}
```

Half Duplex TCP Chat Program

Aim: To implement a half-duplex application, where the Client establishes a connection with the Server. The client and server can send and receive messages one at a time.

Algorithm (TCP):

Server:

1. Include necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize the socket and set its attributes.Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
4. Bind the server to socket using bind function.
5. Listen to incoming client requests and wait for the client request. On connection request from the client establish a connection using accept() function.

6. Receive and display the message sent by the client.
7. Read the message from the console and send it to the client.
8. If the received message is “BYE” terminate the connection (kill the process).
9. Close the socket.

Client:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize the socket and set its attribute set. Assign the sin_family to AF_INET, sin_addr to “127.0.0.1”, sin_port to dynamically assigned port number.
4. Connect to server using connect () function to initiate the request.
5. Read the message from the console and send it to the server using send call.
6. Receive the message from the server and display it.
7. If the received message is “BYE” terminate the connection (kill the process).
8. Close the socket.

Full Duplex TCP Chat Program

Aim: To implement a full duplex application, where the Client establishes a connection with the Server. The Client and Server can send as well as receive messages at the same time. Both the Client and Server exchange messages.

Algorithm (TCP):

Server:

1. Include necessary header files.

2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.

3. Initialize the socket and set its attributes.Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.

4. Bind the server to socket using bind function.

5. Listen to incoming client requests and wait for the client request. On connection request from the client establish a connection using accept () function.

6. Fork the process to create child process which is an exact copy of the calling process (parent process).

7. If the process is child receive the message from the client and display it. Else if the process is parent read the message from the console and send it to the client.

8. If the received message is "BYE" terminate the connection (kill the process).

9. Close the socket.

Client:

1. Include the necessary header files.

2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.

3. Initialize the socket and set its attribute set. Assign the sin_family to AF_INET, sin_addr to "127.0.0.1", sin_port to dynamically assigned port number.

4. Connect to server using connect () function to initiate the request.

5. Fork the process to create child process which is an exact copy of the calling process (parent process).

6. If the process is child read the message from the console and send it to the server. If the process is parent receive the message from the server and display it.

7. If the received message is “BYE” terminate the connection (kill the process).

8. Close the socket.

Note : Operations for UDP follows the same process but make necessary changes in the system calls.

Lab Exercises

1. Write two separate C programs using UNIX socket APIs illustrate full duplex mode chat application between a single client and server using connection oriented service. Display PID and PPID of both parent and child processes.

2. Write two separate C programs using UNIX socket APIs illustrate half duplex mode chat application between a single client and server connection less service in which the server estimates and prints all permutations of a string sent by the client.

3. Write two separate C programs (one for server and other for client) using socket APIs, to implement the following connection-oriented client-server model.

   (a) The user at the client side sends an alphanumeric string to the server.

   (b) The child process at the server sorts the numbers of the alphanumeric string in ascending order. The parent process at the server sorts the characters of the alphanumeric string in descending order.

   (c) Both the processes send the results to the client along with its corresponding process ID.

Sample Output: At the client side:

Input string: hello451bye7324

At the server side:

Output at the child process of the server: 1234457

Output at the parent process of the server: yollheeb

Additional Exercises

1. Write a C program to simulate a menu driven calculator using client server architecture that performs the following. The client prompts the user with the options as listed below 1. Add/Subtract two integers

   2. Find the value of 'x' in a linear equation

   3. Multiply two matrices

   4. Exit

   Based on the user input the client prompts the user to enter required data. The client sends the option chosen and the relevant data to the server. The server performs the required operation and sends the result to the client. Note that if option 1 is selected, the server provides result of both addition and subtraction of the two integers.

Lab No. 4: Database Operations and Domain Name Servers (DNS) using Socket Programming

Objectives

1. To implement client server communication that involves database operations such as insert, delete, edit, search or replace the data.
2. To illustrate the significance of Domain Name Server using socket programming.

Introduction

As we are moving towards an era of big data and digitization, handling applications that involve huge database and providing controlled user access is a daunting task. Many applications like banking/e-commerce use complex authentication protocols to provide access to its registered customers. A simple authentication algorithm would be to validate the login credentials with the existing database at the server. An instance of such application can be implemented in a client server architecture using socket programming. To build such an application structures are used. Structure is user defined data type available in C that allows to combine data items of different kinds. They are used to represent a record. Suppose it is necessary to keep track of all the books in a library, then the following attributes might define a book.

- Title
- Author
- Subject
- Book ID

Accessing Structure Information:

/******* Define a structure with name Books*****/

```
struct Books {
char  title[50];
char  author[50];
char  subject[100];
int book_id;
};
int main( ) {
struct Books Book1;        /* Declare Book1 of type Book */
struct Books Book2;        /* Declare Book2 of type Book */
/* book 1 specification */
strcpy( Book1.title , "C Programming");
strcpy( Book1.author , "Nuha Ali");
strcpy( Book1.subject , "C Programming Tutorial");
Book1.book_id = 6495407;
/* print Book1 info */
printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author);
printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);
}
```

Algorithm to implement Library Database Management System (TCP):

Server:

1. Include necessary header files. Create a structure containing all the fields required in the database along with their types.

2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize the socket and set its attributes.Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
4. Bind the server to socket using bind function.
5. Listen to incoming client requests and wait for the client request. On connection request from the client establish a connection using accept() call.
6. Based on client's choice perform Suitable database operations.
7. Communicate the results by sending appropriate messages back to the client informing the success or failure of the requested operation using send()- recv() calls
8. If the client chooses "Exit "option Close the connection.

Client:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize the socket and set its attribute set. Assign the sin_family to AF_INET, sin_addr to "127.0.0.1", sin_port to dynamically assigned port number.
4. Connect to server using connect () function to initiate the request.
5. Select the required option and sends it along with the necessary information to the server.
6. Receive the buffer from the server and print its contents (results) at the console.
7. Close the connection.

Note: For connectionless services repeat the above steps by making suitable changes in the socket calls.

Domain Name Servers (DNS)

Computers and other network devices on the Internet use an IP address to route the client request to required website. It's impossible for us to remember all the IP addresses of the servers we access every day. Hence we assign a domain name for every server and use a protocol called DNS to turn a user-friendly domain name like "howstuffworks.com" into an Internet Protocol (IP) address like 70.42.251.42 that computers use to identify each other on the network. In other words, DNS is used to map a host name in the application layer to an IP address in the network layer. DNS is a client/server application in which a domain name server, also called a DNS server or name server, manages a massive database that maps domain names to IP addresses. Client requests for address resolution which is defined as mapping a name to an address or an address to a name. It can be done in a recursive fashion or in an iterative fashion.

Algorithm to simulate DNS environment (iterative method)

Server:

1. Include header files.
2. Create the socket for the server.
3. Bind the socket to the port.
4. Listen for the incoming client connection.
5. Receive the IP address from the client to be resolved.
6. Get the domain name from the client.
7. Check the existence of the domain in the server database which is a text file.
8. If domain matches then send the corresponding address to the client. Otherwise send a negative response.
9. Stop the program execution.

Client:

1. Include header files.
2. Create the socket for the client.
3. Connect the socket to the server.
4. Prompt user to enter the hostname and send the hostname to the server.
5. Display response received.
6. Terminate the program.

Lab Exercises

1. Write two separate C programs (one for server and other for client) using socket APIs for TCP and UDP to perform the following. The user at the client side has an option to enter:
   1. Registration Number
   2. Name of the Student
   3. Subject Code.

   The Client sends the selected option along with the requisite details to the server. Based on the options received the parent process in the server assigns the task to respective child process.

   (a) If registration number is sent then the first child process sends Name and Residential Address of the student along with the PID of the child process.

   (b) If Name of the Student is received then the second child process sends student enrollment details (Dept., Semester, Section and Courses Registered) along with the PID of the child process.

   (c) If Subject Code is entered then the third child process sends the corresponding marks obtained in that subject along with its PID.

   (d) The details sent by the server have to be displayed at the client.

2. Write two separate C programs (one for server and other for client) using UNIX socket APIs using connection oriented services to implement DNS Server. Accept suitable input messages from the user. Assume the server has access to database.txt (can be a structure too). Response is always displayed at the client side.

Additional Exercise

1. Create a Book database at the server side and store the following information: title, author, accession number, total pages, and the publisher. Write C programs to implement the following client-server model:

   a. Insert new book information

   b. Delete a book

   c. Display all book information

   d. Search a book (Based on Title or author)

   e. Exit

   At the client side, the user selects the required option and sends it along with the necessary information to the server and server will perform the requested operation. Server should send appropriate messages back to the client informing the success or failure of the requested operation. Client should continue to request the operation until user selects the option ”Exit”. To search the book by author name, the client program should send the name of the author to the server. The list of all book details for that author should be sent to the client. If the author name is not found, then server should send appropriate message to the client.

Lab No. 5 : Multiple Clients Communication using Socket Programming

Objectives

1. To implement multiple client communication with concurrent server.
2. To implement the services performed by an iterative DNS server to multiple clients using socket programming.

Introduction

There are two main classes of servers, iterative and concurrent. An iterative server iterates through each client, handling it one at a time. A concurrent server handles multiple clients at the same time. The simplest technique for a concurrent server is to call the fork function, creating one child process for each client. An alternative technique is to use threads instead (i.e., light-weight processes).

Process-based using Fork:

- Spawn one server process to handle each client connection
- Kernel automatically interleaves multiple server processes
- Each server process has its own private address space

A typical concurrent server using fork has the following structure:

```
pid_t pid;
int listenfd, connfd;
listenfd = socket(...);
/*** fill the socket address with server's well known port ***/
bind(listenfd, ...);
listen(listenfd, ...);
```

```
for ( ; ; ) {
connfd = accept(listenfd, ...); /* blocking call */
if ( (pid = fork()) == 0 ) {
close(listenfd); /* child closes listening socket */
/***process the request doing something using connfd ***/
/* ................. */
close(connfd);
exit(0);  /* child terminates
}
close(connfd);  /*parent closes connected socket*/
}
}
```

When a connection is established, accept returns, the server calls fork, and the child process services the client (on the connected socket connfd). The parent process waits for another connection (on the listening socket listenfd). The parent closes the connected socket since the child handles the new client. The interactions among client and server are presented in Figure.

TCP Iterative Server Example

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/
#define LISTENQ 8 /*maximum number of client connections */
int main (intargc, char **argv)
{
int listenfd, connfd, n;
socklen_t clilen;
char buf[MAXLINE];
struct sockaddr_incliaddr, servaddr;
//creation of the socket
listenfd = socket (AF_INET, SOCK_STREAM, 0);
//preparation of the socket address
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
bind (listenfd, (structsockaddr *) &servaddr, sizeof(servaddr));
listen (listenfd, LISTENQ);
printf("%s\n","Server running...waiting for connections.");
for ( ; ; ) {
clilen = sizeof(cliaddr);
connfd = accept (listenfd, (structsockaddr *) &cliaddr, &clilen);
```

```
printf("%s\n","Received request...");
while ( (n = recv(connfd, buf, MAXLINE,0)) > 0)  {
printf("%s","String received from and resent to the client:");
puts(buf);
send(connfd, buf, n, 0);
}
if (n < 0) {
perror("Read error");
exit(1);
}
close(connfd);
}
//close listening socket
close (listenfd);
}
```

TCP Concurrent server Using Fork

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <unistd.h>
#define MAXLINE 4096 /*max text line length*/
#define SERV_PORT 3000 /*port*/
#define LISTENQ 8 /*maximum number of client connections*/

int main (intargc, char **argv)
{
```

```
Int listenfd, connfd, n;
pid_t childpid;
socklen_t clilen;
char buf[MAXLINE];
struct sockaddr_in cliaddr, servaddr;
//Create a socket
//If sockfd<0 there was an error in the creation of the socket
if ((listenfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
perror("Problem in creating the socket");
exit(2);
}
//preparation of the socket address
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
//bind the socket
bind (listenfd, (structsockaddr *) &servaddr, sizeof(servaddr));
//listen to the socket by creating a connection queue, then wait for
listen (listenfd, LISTENQ);
printf("%s\n","Server running...waiting for connections.");
for ( ; ; ) {
clilen = sizeof(cliaddr);
//accept a connection
connfd = accept (listenfd, (structsockaddr *) &cliaddr, &clilen);
printf("%s\n","Received request...");
if ( (childpid=fork ())==0 ) {//if it's 0, it's child process
printf ("%s\n","Child created for dealing with client requests");
//close listening socket
close (listenfd);
```

```
while ( (n = recv(connfd, buf, MAXLINE,0)) > 0)  {
printf("%s","String received from and resent to the client:");
puts(buf);
send(connfd, buf, n, 0);
}
if (n < 0)
printf("%s\n", "Read error");
exit(0);
}
//close socket of the server
close(connfd);
}
}
```

Lab Exercises

1. Write a single server and multiple client program to illustrate multiple clients communicating with a concurrent server. The client1 on establishing successful connection sends "Institute Of" string to the server along with its socket address. The client2 on establishing successful connection sends "Technology" string to the server along with its socket address. The server opens a text file having the keyword "Manipal", append the keywords "Institute of" and "Technology" and displays "Manipal Institute of Technology" along with the socket addresses of the clients . If the number of clients connected exceeds 2, the server sends "terminate session" to all clients and the program terminates.

2. Write a single server multiple client program to illustrate multiple clients communicating with a single iterative server. The client on establishing successful connection prompts the user to enter 2 strings which is sent to the server along with client socket address. The server checks whether the strings are anagrams or not and

sends an appropriate message to the client. The result obtained is then displayed on the client side. The server displays the date and time along with client socket address that it is connected to it at any instant.

Additional Exercise

1. Write C program to simulate travel ticket reservation system. Where the server displays the number of seats available and the number of seats booked of two different source and destination locations. Multiple clients try to connect to server and sends the number of seats to be booked as entered by the user. The server database has to be updated and the client should terminate its session after successful seat reservation. Note that if the requested number of seats are unavailable the server sends appropriate message to the client and the client program terminates. Price of ticket need not be taken into consideration.

Lab No. 6: Token Bus and Token Ring Implementation using Socket Programming

Objectives

- Design and Implementation of Token Bus Topology using Socket Programming
- Design and Implementation of Token Ring Topology using Socket Prorgamming

Token Bus Protocol

In this, one long cable acts as a backbone to link all the devices in a network. If the backbone is broken, the entire segment fails. When a station has finished sending its data, it releases the token and inserts the address of its successor in the token. Only the station with address matching the destination address of the token gets the token to access the shared media. In the intermediate system fails, the token directly passes to the next available system. But it's not applicable in token ring protocol.

Algorithm

client 1:

1. Start the program
2. Open socket with input address and port
3. Establish a connection between client 1, client 2 and client 3
4. Pass the token from client1 to client 2
5. If the client 2 breaks, pass token to client3

6. Stop the program

client 2:

1. Start the program
2. Initialize server socket
3. Wait to connect with client 2
4. Initialize the socket and accept the client 1 message
5. Display connected with client 1
6. Receive the token sent by client 1
7. Establish a connection between client 2 and client 3
8. Open socket with input address and port
9. Pass the token to client 3
10. If the client 3 is disconnected, terminate the program
11. Stop

client 3:

1. Start the program
2. Initialize server socket
3. Wait to connect with client 2
4. Initialize the socket and accept the client 2 message
5. Receive the token which has sent by client 2
6. If the client 2 is disconnected, receive the token by client 1.
7. Stop

How to Execute the Programs

1. Type the Client 1, Client 2 and Client 3 program in three different computers.

2. To verify Case 1: first execute Client 3 then Client 2 then Client 1, so Client 3 waiting for token from Client 2, Client 2 waiting for Token from Client 1, Client 1 starts the token sending, it sends token to Client 2, now client 2 holds token for 40 seconds then it sends token to Client 3.

3. To verify Case 2: first execute Client 3 then Client 1, don't execute program in Client 2 (It is assumed like Client 2 is disconnected from the LAN), Client 1 starts the token sending, it sends token to Client 2, but Client 2 is disconnected from the LAN, so the token sent to Client 3.

4. To verify Case 3: Don't execute Client 3 (It is assumed like Client 3 is disconnected from LAN), execute program in Client 2 then Client 1, Client 1 sends token to Client 2, now Client 2 trying to send token to Client 3, but Client 3 is disconnected from the LAN, so token not sent to Client 3.

5. To verify Case 4: Don't execute Client 3 and Client 2 (It is assumed like Client 3 and Client 2 are disconnected from the LAN)

Token Ring Protocol

In the token passing method, the stations in a network are organized in a logical ring. In a physical ring topology, when a station sends the token to its successor, the token cannot be seen by other stations. In this, each device has a dedicated point-to-point connection with only the two devices on either side of it. In this method, a special packet called token circulates throughout the ring. When a station has some data to send, it waits until it receives the token from its predecessor. It then holds the token and sends its data. When the station has no more data to send, it releases the token, passing it to the next logical station in the ring.

Algorithm

client 1:

1. Start the program
2. Open socket with input address and port
3. Establish a connection between client 1 and client2
4. Pass the token to client 2
5. Stop the program

client 2:

1. Start the program
2. Initialize server server socket
3. Wait to connect with client 2
4. Initialize the socket and accept the client message
5. Display connected with client1
6. Receive the token sent by client 1
7. Establish a connection between client 2 and client 3
8. Open socket with input address and port
9. Pass the token to client 3
10. Stop

client3:

1. Start the program
2. Initialize server socket

3. Wait to connect with client 2
4. Initialize the socket and accept the client 2 message
5. Receive the token which has sent by client 2
6. Stop

How to Execute:

1. Type the Client 1, Client 2 and Client 3 program in three different computers.
2. To verify Case 1: first execute Client 3 then Client 2 then Client, so Client 3waiting for token from Client 2, Client 2 waiting for Token from Client 1, Client 1 starts the token sending, it sends token to Client 2, now client 2 holds token for 40 seconds then it sends token to Client 3.
3. To verify Case 2: first execute Client 3 then Client 1, don't execute program in Client 2 (It is assumed like Client 2 is disconnected from the LAN), Client 1 starts the token sending, it sends token to Client 2, but Client 2 is disconnected from the LAN, so that, the token ring breaks.
4. To verify Case 3: Don't execute Client 3 (It is assumed like Client 3 is disconnected from LAN), execute program in Client 2 then Client 1, Client 1 sends token to Client 2, now Client 2 trying to send token to Client 3, but Client 3 is disconnected from the LAN, so that, token ring breaks.
5. To verify Case 4 : Don't execute Client 3 and Client 2 (It is assumed like Client 3 and Client 2 are disconnected from the LAN). So the token ring process is not initiated at all

Lab Exercises

1. Implement the Token Bus Protocol with variations in the topology using Socket Programming

2. Implement the Token Ring Protocol with variations in the topology using Socket Programming

Lab No. 7 : Application Development using Socket Programming

Objectives

- To apply the socket programming concepts in developing the real world applications.
- Develop an application using C programming or Python programming.

Socket Programming Using Python

Summary of the key functions from Socket - Low-level networking interface:

1. socket.socket(): Create a new socket using the given address family, socket type and protocol number.
2. socket.bind(address): Bind the socket to address.
3. socket.listen(backlog): Listen for connections made to the socket. The backlog argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.
4. socket.accept(): The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.

   At accept(), a new socket is created that is distinct from the named socket. This new socket is used solely for communication with this particular client.

   For TCP servers, the socket object used to receive connections is not the same socket used to perform subsequent communication with the client. In particular, the accept() system call returns a new socket object that's actually used for the connection. This allows a server to manage connections from a large number of clients simultaneously.

5. socket.send(bytes[, flags]): Send data to the socket. The socket must be connected to a remote socket. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.

6. socket.colse(): Mark the socket closed. all future operations on the socket object will fail. The remote end will receive no more data (after queued data is flushed). Sockets are automatically closed when they are garbage-collected, but it is recommended to close() them explicitly.

In Python 3, all strings are Unicode. So, if any kind of text string is to be sent across the network, it needs to be encoded.This is why the server is using the encode('ascii') method on the data it transmits. Likewise, when a client receives network data, that data is first received as raw unencoded bytes. If you print it out or try to process it as text, we're unlikely to get what we expected. Instead, we need to decode it first.This is why the client code is using decode('ascii') on the result.

Server

```
# server.py
import socket
import time
# create a socket object
serversocket = socket.socket(
socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# bind to the port
serversocket.bind((host, port))
# queue up to 5 requests
```

```
serversocket.listen(5)
while True:
# establish a connection
clientsocket,addr = serversocket.accept()
print("Got a connection from %s" % str(addr))
currentTime = time.ctime(time.time()) + "\r\n"
clientsocket.send(currentTime.encode('ascii'))
clientsocket.close()
```

Client

```
# client.py
import socket
# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# connection to hostname on the port.
s.connect((host, port))
# Receive no more than 1024 bytes
tm = s.recv(1024)
s.close()
print("The time got from the server is %s" % tm.decode('ascii'))
```

Output

```
$ python server.py &
Got a connection from ('127.0.0.1', 54597)
$ python client.py
The time got from the server is Wed Jul 14 19:14:15 2020
```

Lab Exercises

1. Write two separate C programs or python programs (one for server and other for client) using socket APIs, to implement the following connection-oriented client-server model for "BANKING APPLICATION". To login, the user at client side sends username and password (can be alphanumeric) to the server. The server maintains a file (database) that has a list of username, its corresponding password in encrypted form (Use Caser Cipher for encryption where each letter is replaced by a letter which is a shift of 3 of the original letter. Eg. 'a' is replaced by 'd') and the current account balance. On receiving the credentials from the client, the server first encrypts the password and validates it against the file contents).

   (a) If the username is incorrect, the server displays 'Incorrect Username 'and sends this message to the client.

   (b) Otherwise, the server displays 'Incorrect Password' and sends this message to the client.

   On Successful login the server sends a menu with following options.

   a. Debit b. Credit c. View Balance d. EXIT Based on the Choice of the client transactions are done (must be reflected in the database) and the application Quits on choosing option d.

2. Demonstrate a TCP Chat System between two PCS using C or Python.

3. Write two separate C programs or python programs (one for server and other for client) using socket APIs, to implement the following client-server model. The user at the client side sends a filename containing a text to the server. The server checks the existence of the file. If present it performs the following operations:

   - The child process at the server converts the text to uppercase.
   - The parent process at the server replaces each letter in the text with its equivalent digit. Map a to 1, b to 2 and so on.

- Both the processes should write the results onto the file with their process IDs.
- Client reads the results from the file.

Additional Exercises

1. Write two separate C programs or python programs (one for server and the other for client) using socket APIs, to implement the following connection-oriented client -server model for Movie ticket booking application. The server maintains a database (file/structure) consisting of 'Movie Names', 'Movie timings' and 'Seats Available'. At the client side the following menu is displayed.

1. Book Tickets

2. Exit.

The option selected by the user is sent to the server. If option '1' is selected, a list of 'Movie Names' is sent to the client by the server.

i. The user at the client side selects a movie name which is displayed at the server side.

ii. The server sends 'Movie timings' and 'Seats available' to the user at the client.

iii. The user sends movie timings and the required number of seats to the server. If seats are available the server decrements the number of seats available and sends the message 'Seats booked' to the user at client. If Seats are unavailable the Server sends the message 'Seats Unavailable' to the client. The updated database must be displayed at the server side. This process should continue until user selects option 2 (Exit).

Lab No. 8: Prototyping the Network Model using Packet Tracer

Objectives

1. Develop an understanding of the basic functions of Packet Tracer.
2. To build a network using packet tracer
3. To demonstrate use of different network components such as routers, switches etc
4. Observe traffic behavior on the network.

Introduction

Cisco Packet Tracer is a powerful network simulation program that allows students to experiment with network behavior and ask "what if" questions. As an integral part of the Networking Academy comprehensive learning experience, Packet Tracer provides simulation, visualization, authoring, assessment, and collaboration capabilities and facilitates the teaching and learning of complex technology concepts.

Packet Tracer supplements physical equipment in the classroom by allowing students to create a network with an almost unlimited number of devices, encouraging practice, discovery, and troubleshooting. The simulation-based learning environment helps students develop 21st century skills such as decision making, creative and critical thinking, and problem solving. Packet Tracer complements the Networking Academy curricula, allowing instructors to easily teach and demonstrate complex technical concepts and networking systems design.

Packet Tracer complements the Networking Academy curricula, allowing instructors to easily teach and demonstrate complex technical concepts and networking systems design. Instructors can customize individual or multiuser activities, providing hands-on lessons for students that offer value and relevance in their classrooms. Students can build,

configure, and troubleshoot networks using virtual equipment and simulated connections, alone or in collaboration with other students. Packet Tracer offers an effective, interactive environment for learning networking concepts and protocols. Most importantly, Packet Tracer helps students and instructors create their own virtual "network worlds" for exploration, experimentation, and explanation of networking concepts and technologies.

Key Features

Packet Tracer Workspaces: Cisco Packet Tracer has two workspaces—logical and physical. The logical workspace allows users to build logical network topologies by placing, connecting, and clustering virtual network devices. The physical workspace provides a graphical physical dimension of the logical network, giving a sense of scale and placement in how network devices such as routers, switches, and hosts would look in a real environment. The physical view also provides geographic representations of networks, including multiple cities, buildings, and wiring closets.

Packet Tracer Modes

Cisco Packet Tracer provides two operating modes to visualize the behavior of a network—real-time mode and simulation mode. In real-time mode the network behaves as real devices do, with immediate real-time response for all network activities. The real-time mode gives students a viable alternative to real equipment and allows them to gain configuration practice before working with real equipment. In simulation mode the user can see and control time intervals, the inner workings of data transfer, and the propagation of data across a network. This helps students understand the fundamental concepts behind network operations. A solid understanding of network fundamentals can help accelerate learning about related concepts.

Protocols: Cisco Packet Tracer supports the following protocols

Layer	Cisco Packet Tracer Supported Protocols
Application	• FTP , SMTP, POP3, HTTP, TFTP, Telnet, SSH, DNS, DHCP, NTP, SNMP, AAA, ISR VOIP, SCCP config and calls ISR command support, Call Manager Express
Transport	• TCP and UDP, TCP Nagle Algorithm & IP Fragmentation, RTP
Network	• BGP, IPv4, ICMP, ARP, IPv6, ICMPv6, IPSec, RIPv1/ v2/ng, Multi-Area OSPF, EIGRP, Static Routing, Route Redistribution, Multilayer Switching, L3 QoS, NAT, CBAL , Zone-based policy firewall and Intrusion Protection System on the ISR, GRE VPN, IPSec VPN
Network Access/ Interface	• Ethernet (802.3), 802.11, HDLC, Frame Relay, PPP, PPPoE, STP, RSTP, VTP, DTP, CDP, 802.1q, PAgP, L2 QoS, SLARP, Simple WEP, WPA, EAP

A client has requested that you set up a simple network with two PCs connected to a switch. Verify that the hardware, along with the given configurations, meet the requirements of the client. Following are the steps to setup the above given scenario:

Step 1: Set up the network topology

a) Add two PCs and a Cisco 2950T switch.

b) Using straight-through cables, connect PC0 to interface Fa0/1 on Switch0 and PC1 to interface Fa0/2 on Switch0.

c) Configure PC0 using the Config tab in the PC0 configuration window:

1. IP address: 192.168.10.10

2. Subnet Mask 255.255.255.0

d) Configure PC1 using the Config tab in the PC1 configuration window:

1. IP address: 192.168.10.11

2. Subnet Mask 255.255.255.

Step 2: Test connectivity from PC0 to PC1

a) Use the ping command to test connectivity.

1. Click PC0.

2. Choose the Desktop tab.

3. Choose Command Prompt.

4. Type: ping 192.168.10.11 and press enter.

b) A successful ping indicates the network was configured correctly and the prototype

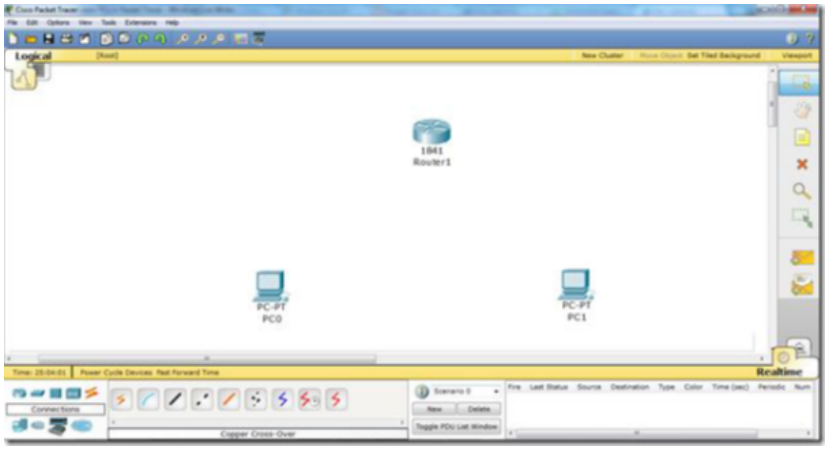
validates the hardware and software configurations. A successful ping should resemble the below output:

```
Pinging 192.168.10.11 with 32 bytes of data:

Reply from 192.168.10.11: bytes=32 time=170ms TTL=128
Reply from 192.168.10.11: bytes=32 time=71ms TTL=128
Reply from 192.168.10.11: bytes=32 time=70ms TTL=128
Reply from 192.168.10.11: bytes=32 time=68ms TTL=128

Ping statistics for 192.168.10.11:
   Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
   Minimum = 68ms, Maximum = 170ms, Average = 94ms
```

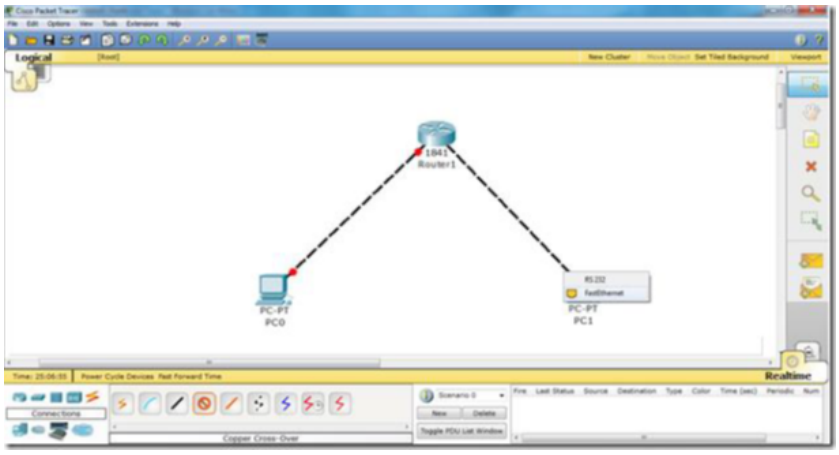
c) Close the configuration window.

Communication between 2 PC's via the router

Here, we will see communication enabled between PCs via Router in Packet Tracer. So, for this we need two PCs, a router, and two cross over cables to connect them. Important point is that we use cross over cable to connect PC to a router because they both use the same pins for transmission and receiving of data.



Now, we will connect them by selecting fast ethernet interfaces on both ends.



Now, we have connect the devices. Further, we will go to the router CLI mode and enter the following commands. Step by step, we will have to do the following things.

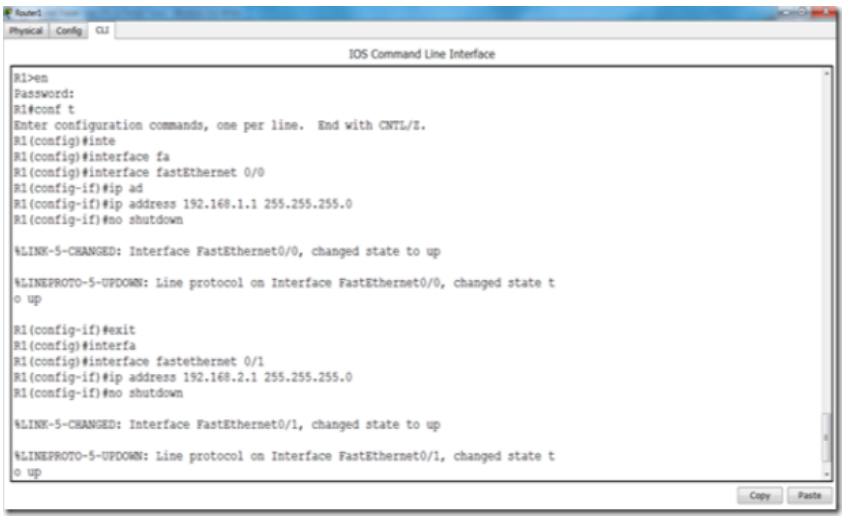
i. Access the interfaces one by one

ii. Assign IP addresses to interfaces

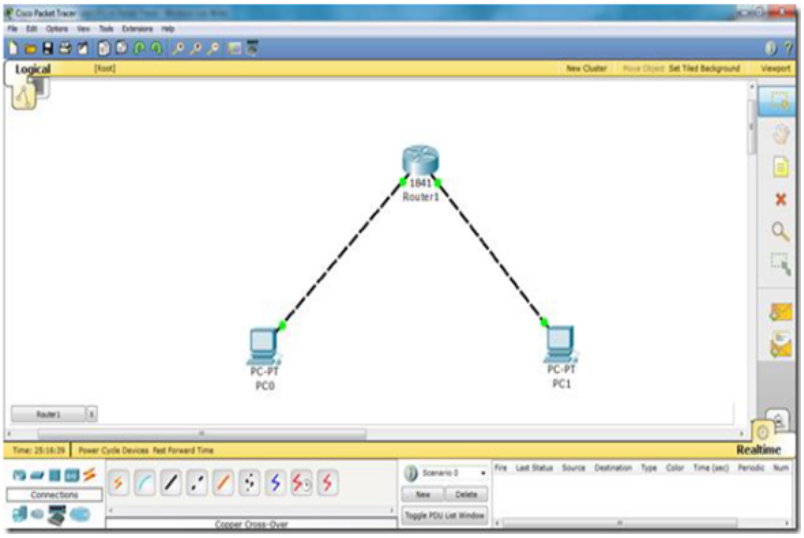
iii. Change the status of the interfaces i.e. from Down to Up

iv. Assign IP addresses to PCs

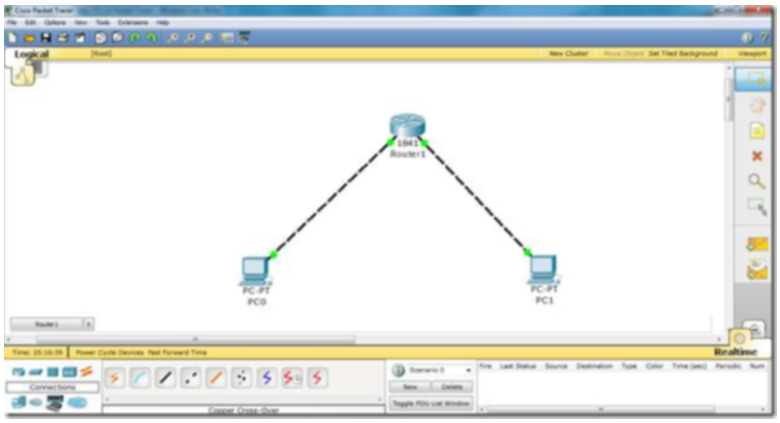
v. Assign Default Gateway to PCs. FYI fast Ethernet ip address is the gateway address to the PC



Now, we have accessed both interfaces one by one and we have assigned IP addresses respectively.

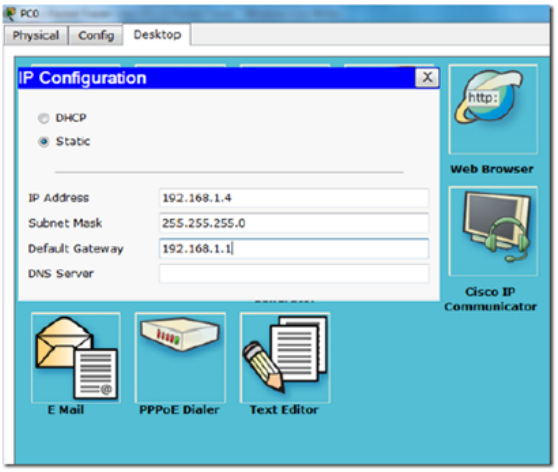


Now, we have accessed both interfaces one by one and we have assigned IP addresses respectively.

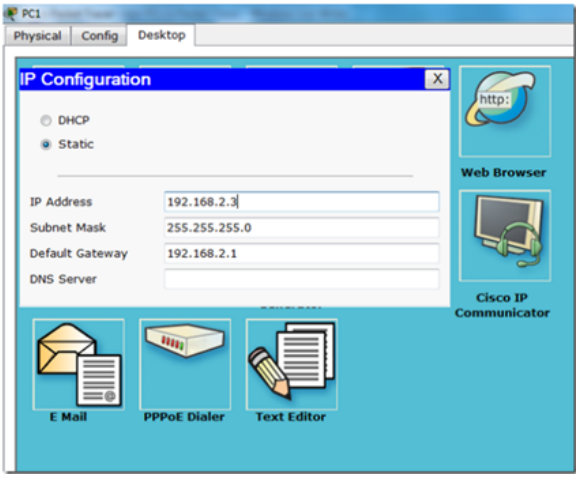


See the difference the lights have changed the color from Red to Green. Now, lets assign IP addresses to the PCs. Click on PC0, go to Desktop, then click IP Configuration.

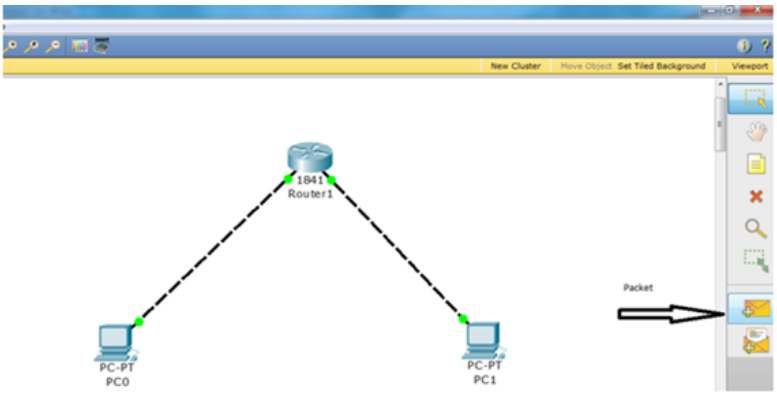
PC0

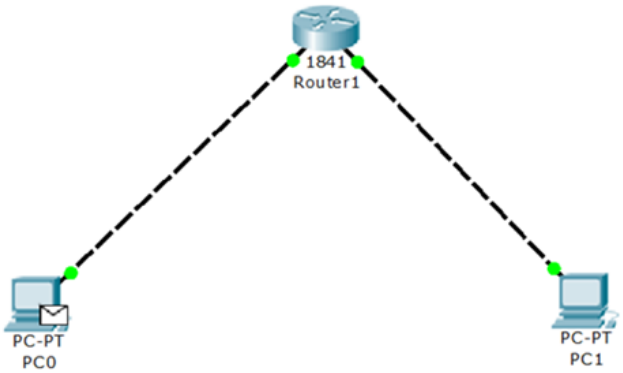


PC1



Now, our communication is enabled and we are able to communicate from PC0 to PC1 via Router. Click on the packet in the right panel on the packet tracer, then click on PC0 and then click on PC1. You will see the successful packet tracer (status is shown in the bottom right corner).





Lab Exercises:

1. Set up a network with 4 switches and 8 PC's and verify the connectivity between all the PC's.

2. Design and configure a network using 4 PC's and a router. Configuration should be done using Command Line Interface (CLI).

Lab No. 9: Implementation Basic Topologies using Packet Tracer

Objectives

1. Implementation of Basic Topologies such as BUS, RING, MESH, STAR using packet tracer.

What is Network Topology

Network topology refers to how various nodes, devices, and connections on your network are physically or logically arranged in relation to each other. Think of your network as a city, and the topology as the road map. Just as there are many ways to arrange and maintain a city—such as making sure the avenues and boulevards can facilitate passage between the parts of town getting the most traffic—there are several ways to arrange a network. Each has advantages and disadvantages and depending on the needs of your company, certain arrangements can give you a greater degree of connectivity and security.

There are two approaches to network topology: physical and logical. Physical network topology, as the name suggests, refers to the physical connections and interconnections between nodes and the network—the wires, cables, and so forth. Logical network topology is a little more abstract and strategic, referring to the conceptual understanding of how and why the network is arranged the way it is, and how data moves through it.

Why Is Network Topology Important?

The layout of your network is important for several reasons. Above all, it plays an essential role in how and how well your network functions. Choosing the right topology for your company's operational model can increase performance while making it easier to

locate faults, troubleshoot errors, and more effectively allocate resources across the network to ensure optimal network health. A streamlined and properly managed network topology can increase energy and data efficiency, which can in turn help to reduce operational and maintenance costs.

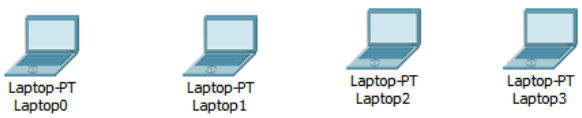
The design and structure of a network are usually shown and manipulated in a software-created network topology diagram. These diagrams are essential for a few reasons, but especially for how they can provide visual representations of both physical and logical layouts, allowing administrators to see the connections between devices when troubleshooting.

Bus Topology

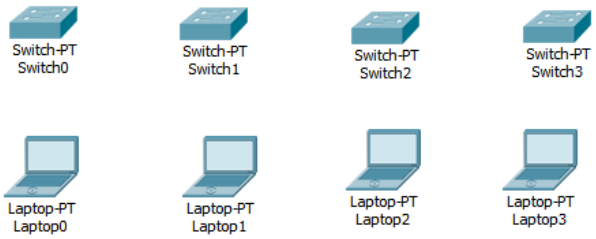
In local area network, it is a single network cable runs in the building or campus and all nodes are connected along with this communication line with two endpoints called the bus or backbone. In other words, it is a multipoint data communication circuit that is easily control data flow between the computers because this configuration allows all stations to receive every transmission over the network.

To build the topology and to test follow the given steps:

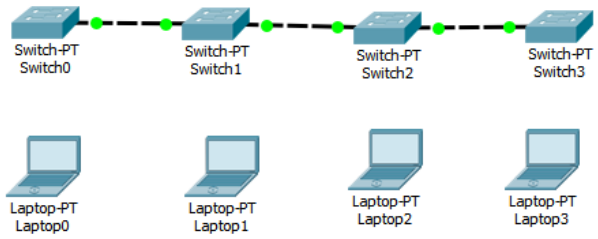
1. Select the end devices (Generic Laptop-PT) from the available devices.



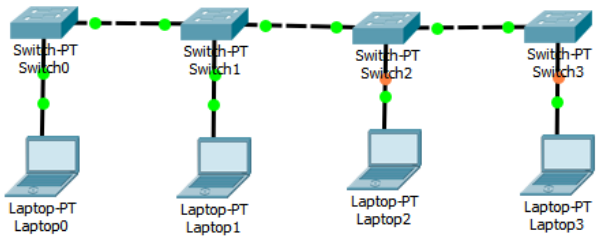
2. Choose switches (Generic)



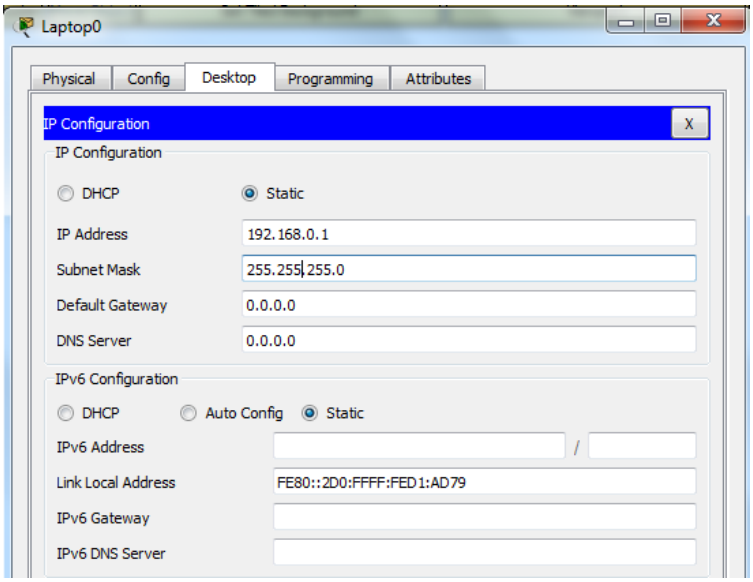
3. Choose connections between switches (Copper Cross-over)



4. Choose connections between end devices (Copper Straight-Through)

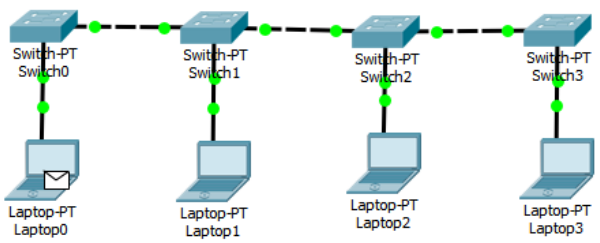


5. Configure end devices with class-c IP address.



End device	IP	Subnet mask
Laptop-PT Laptop0	192.168.0.1	255.255.255.0
Laptop-PT Laptop1	192.168.0.2	255.255.255.0
Laptop-PT Laptop2	192.168.0.3	255.255.255.0
Laptop-PT Laptop3	192.168.0.4	255.255.255.0

6. Send packet from source to destination.

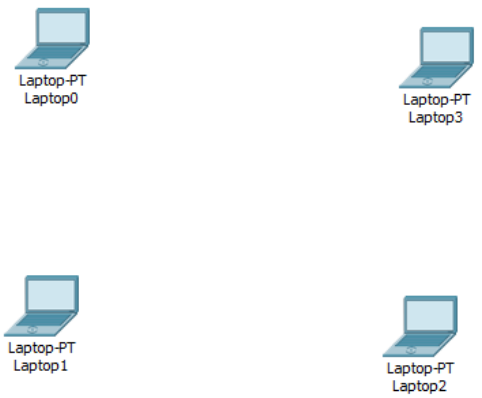


Ring Topology

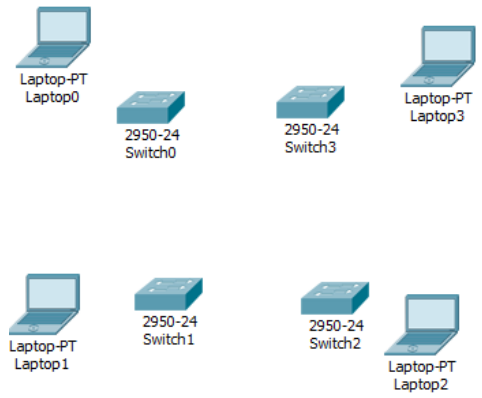
In ring topology each device is connected with the two devices on either side of it. There are two dedicated point to point links a device has with the devices on the either side of it. This structure forms a ring thus it is known as ring topology. If a device wants to send data to another device then it sends the data in one direction, each device in ring topology has a repeater, if the received data is intended for other device then repeater forwards this data until the intended device receives it.

To build the topology and to test follow the given steps:

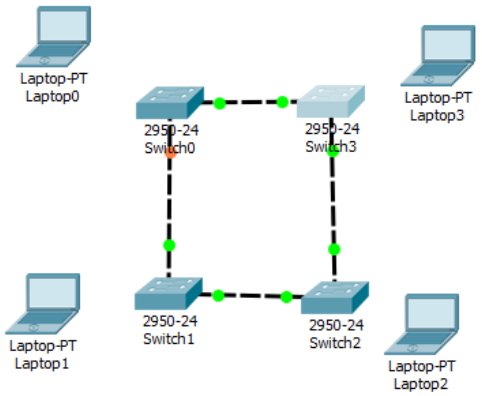
1. Select the end devices (Generic Laptop-PT) from the available devices.



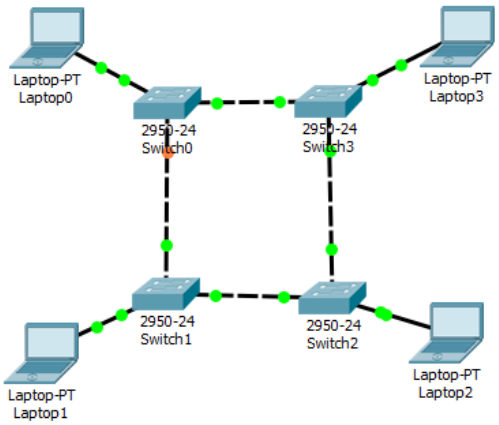
2. Choose switches (2950-24)



3. Choose connections between switches (Copper Cross-over)

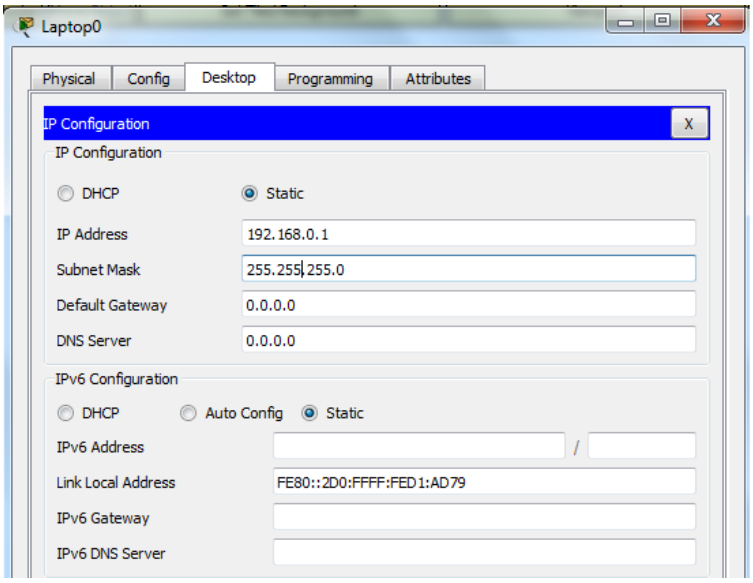


4. Choose connections between end devices (Copper Straight-Through)

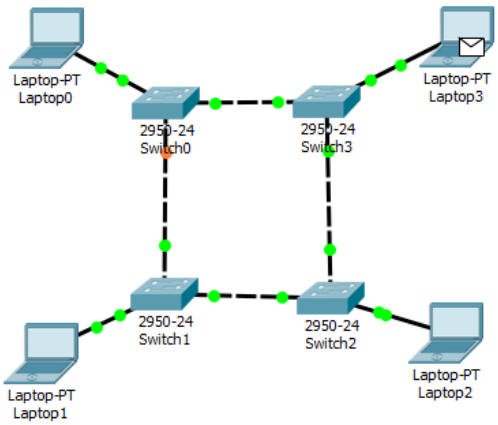


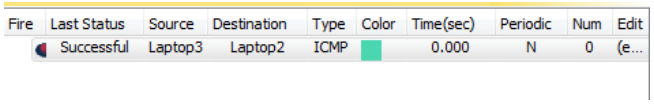
5. Assign IP configuration for the devices

End devices	IP	Subnet mask
Laptop-PT Laptop0	192.168.0.1	255.255.255.0
Laptop-PT Laptop1	192.168.0.2	255.255.255.0
Laptop-PT Laptop2	192.168.0.3	255.255.255.0
Laptop-PT Laptop3	192.168.0.4	255.255.255.0



6. Send packet from source to destination.





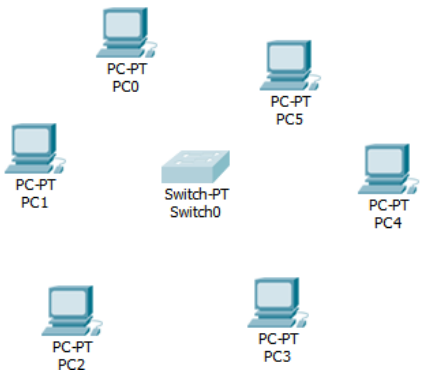
Star Topology

In star topology, all the cables run from the computers to a central location where they are all connected by a device called a hub. It is a concentrated network, where the endpoints are directly reachable from a central location.

To build the topology and to test follow the given steps:

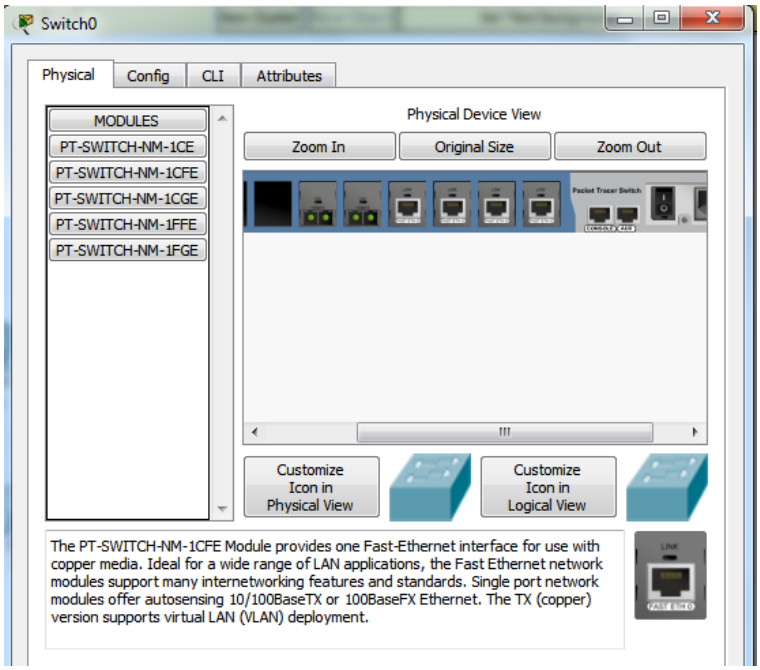
1. Select Switch

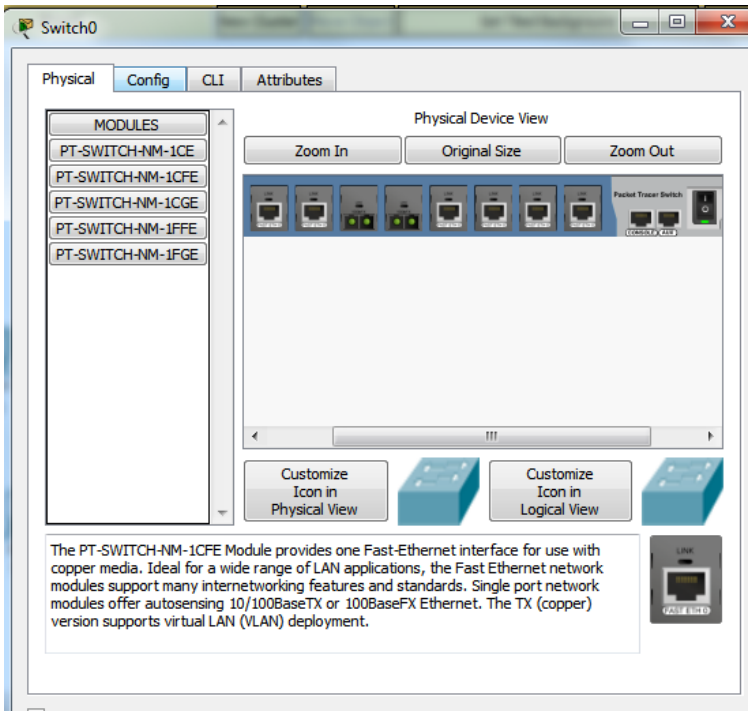
2. Select End Devices



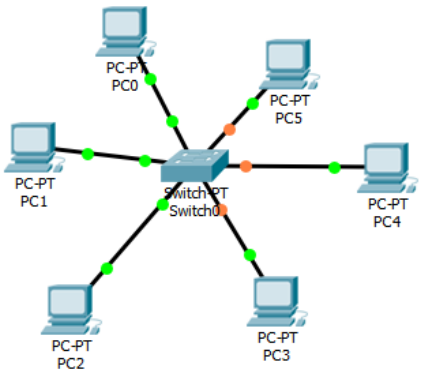
3. Since switch can be connected to four end devices, and here we have six end devices, so add two more modules.

   Click on switch physical $->$ packet tracer switch off $->$ select PT-SWITCH-NM-1CFE $->$ add two modules of it $->$ packet tracer switch on

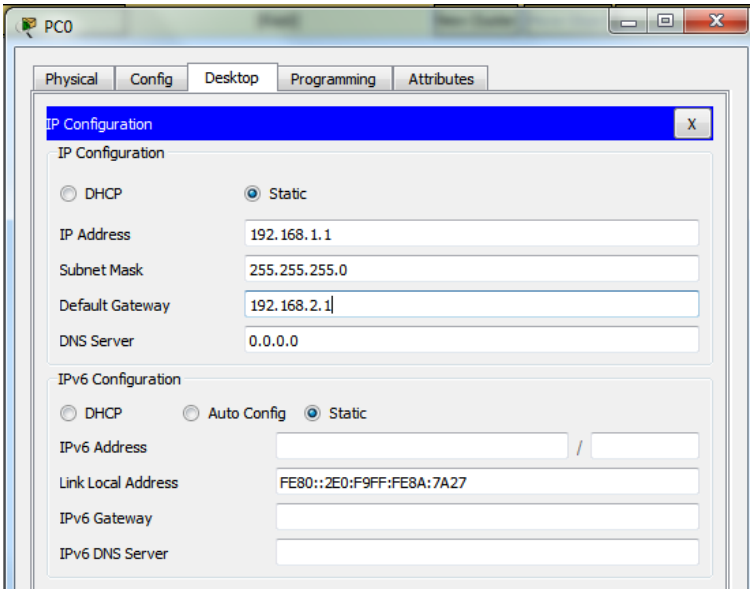




4. Choose connections between PCs and switch (Automatically choose Connection type).

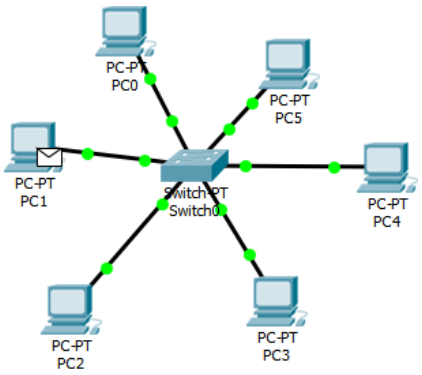


5. Configure the PC with IP address, subnet mask and default gateway



End device	IP	Subnet mask	Default Gateway
PT –PT PC0	192.168.1.1	255.255.255.0	192.168.2.1
PT –PT PC1	192.168.1.2	255.255.255.0	192.168.2.1
PT –PT PC2	192.168.1.3	255.255.255.0	192.168.2.1
PT –PT PC3	192.168.1.4	255.255.255.0	192.168.2.1
PT –PT PC4	192.168.1.5	255.255.255.0	192.168.2.1
PT –PT PC5	192.168.1.6	255.255.255.0	192.168.2.1

6. Send packet from source to destination.

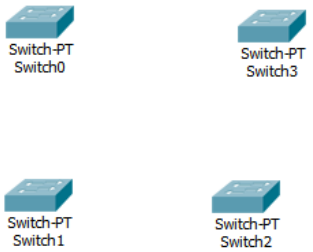


Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit
	Successful	PC0	PC5	ICMP		0.000	N	0	(e...

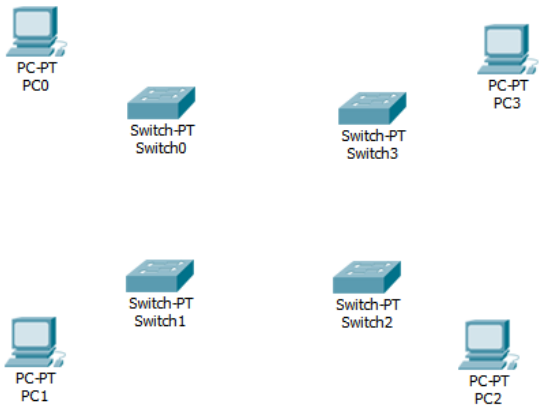
Mesh Topology

In mesh topology every device has a dedicated point to point link to every other device. The term dedicated stand for link carries traffic only between two devices it connects. It is a well-connected topology; in this every node has a connection to every other node in the network. The cable requirements are high and it can include multiple topologies. Failure in one of the computers does not cause the network to break down, as they have alternative paths to other computers star topology, all the cables run from the computers to a central location. To build the topology and to test follow the given steps:

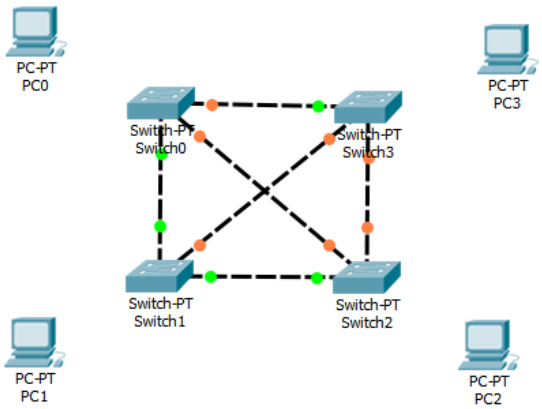
1. Select the switches



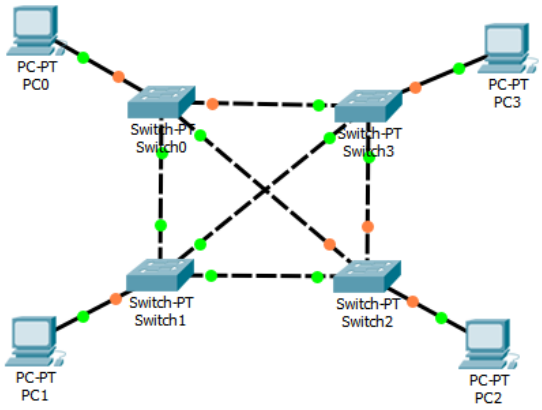
2. Select the end devices



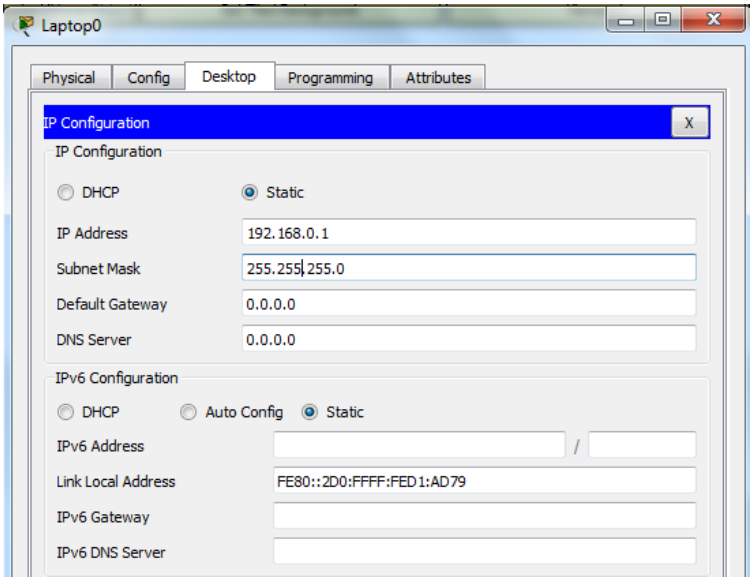
3. Connect the switches (Copper Cross Over)



4. Connect end devices and switches (Copper Straight Through)

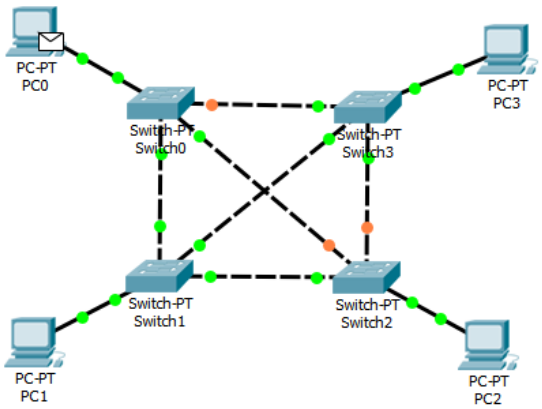


5. Configure end devices with IP address



End device	IP	Subnet mask
PC-PT PC0	192.168.0.1	255.255.255.0
PC-PT PC1	192.168.0.2	255.255.255.0
PC-PT PC2	192.168.0.3	255.255.255.0
PC-PT PC3	192.168.0.4	255.255.255.0

6. Send packets from source to destination



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit
	Successful	PC1	PC3	ICMP		0.000	N	0	(e...
	Successful	PC0	PC2	ICMP		0.000	N	1	(e...

7.

Lab Exercises

1. Design and implement Bus Topology with variations using Packet Tracer.
2. Design and implement Ring Topology with variations using Packet Tracer.
3. Design and implement Star Topology with variations using Packet Tracer.
4. Design and implement Mesh Topology with variations using Packet Tracer.

Additional Exercise

1. Design and implement Hybrid Topology with variations using Packet Tracer.
2. Implement the Token Bus and Token Ring Protocols using Packet Tracer.

Lab No. 10: Configuring Routing Protocols using Packet Tracer

Objectives

1. To understand working principle of various routing protocols using packet tracer.
2. To make use of various routing protocols to route packets between the systems.

RIP: Routing Information Protocol

Routing Information Protocol (RIP) is a standards-based, distance-vector, interior gateway protocol (IGP) used by routers to exchange routing information. RIP uses hop count to determine the best path between two locations. Hop count is the number of routers the packet must go through till it reaches the destination network. The maximum allowable number of hops a packet can traverse in an IP network implementing RIP is 15 hops. It has a maximum allowable hop count of 15 by default, meaning that 16 is deemed unreachable. RIP works well in small networks, but it's inefficient on large networks with slow WAN links or on networks with a large number of routers installed. In a RIP network, each router broadcasts its entire RIP table to its neighboring routers every 30 seconds. When a router receives a neighbor's RIP table, it uses the information provided to update its own routing table and then sends the updated table to its neighbors.

Differences between RIPv1 or RIPv2

RIPv1

- A classful protocol, broadcasts updates every 30 seconds, hold-down period 180 seconds. Hop count is metric (Maximum 15).
- RIP supports up to six equal-cost paths to a single destination, where all six paths

can be placed in the routing table and the router can load-balance across them. The default is actually four paths, but this can be increased up to a maximum of six. Remember that an equal-cost path is where the hop count value is the same. RIP will not load-balance across unequal-cost paths.

RIPv2

- RIPv2 uses multicasts, version 1 use broadcasts, RIPv2 supports triggered updates—when a change occurs, a RIPv2 router will immediately propagate its routing information to its connected neighbors.
- RIPv2 is a classless protocol. RIPv2 supports variable-length subnet masking (VLSM) RIPv2 supports authentication. You can restrict what routers you want to participate in RIPv2. This is accomplished using a hashed password value.

RIP Timers

RIP uses four different kinds of timers to regulate its performance:

1. Route update timer: Sets the interval (typically 30 seconds) between periodic routing updates in which the router sends a complete copy of its routing table out to all neighbors.
2. Route invalid timer: Determines the length of time that must elapse (180 seconds) before a router determines that a route has become invalid. It will come to this conclusion if it hasn't heard any updates about a particular route for that period. When that happens, the router will send out updates to all its neighbors letting them know that the route is invalid.
3. Hold down timer: This sets the amount of time during which routing information is suppressed. Routes will enter into the hold down state when an update packet is received that indicated the route is unreachable. This continues either until an update

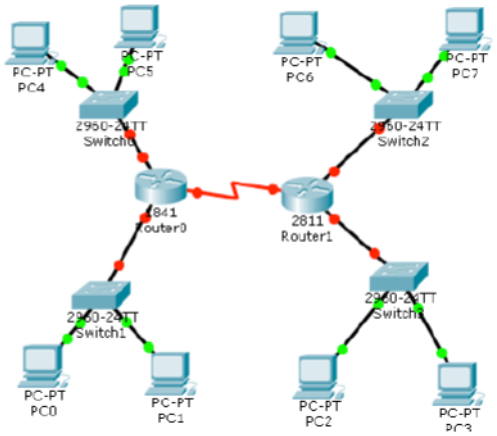
packet is received with a better metric or until the hold down timer expires. The default is 180 seconds.

4. Route flush timer: Sets the time between a route becoming invalid and its removal from the routing table (240 seconds). Before it's removed from the table, the router notifies its neighbors of that route's impending failure. The value of the route invalid timer must be less than that of the route flush timer. This gives the router enough time to tell its neighbors about the invalid route before the local routing table is updated.

RIP Routing Configurations

Configure RIP Routing command cheat sheet

Commands	Descriptions
Router(config)#router rip	Enables RIP as a routing protocol
Router(config-router)#network w.x.y.z	w.x.y.z is the network number of the directly connected network you want to advertise.
Router(config)#no router rip	Turns off the RIP routing process
Router(config-router)#no network w.x.y.z	Removes network w.x.y.z from the RIP routing process.
Router(config-router)#version 2	RIP will now send and receive RIPv2 packets globally.
Router(config-router)#version 1	RIP will now send and receive RIPv1 packets only
Router(config-router)#no auto-summary	RIPv2 summarizes networks at the classful boundary. This command turns auto summarization off.
Router(config-router)#passive-interface s0/0/0	RIP updates will not be sent out this interface.
Router(config-router)#no ip split-horizon	Turns off split horizon (on by default).
Router(config-router)#ip split-horizon	Re-enables split horizon
Router(config-router)#timers basic 30 90 180 270 360	Changes timers in RIP: 30 = Update timer (in seconds) 90 = Invalid timer (in seconds) 180 = Hold-down timer (in seconds) 270 = Flush timer (in seconds) 360 = Sleep time (in milliseconds)
Router#debug ip rip	Displays all RIP activity in real time
Router#show ip rip database	Displays contents of the RIP database



Assign ip address to PC. Select pc and double click on it. Select ip configurations from desktop tab and set ip address given as in table. To configure router double click on it and select CLI. To configure this topology use this step by step guide.

(1841Router0) Hostname R1:

To configure and enable rip routing on R1 follow these commands exactly.

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.
End with CNTL/Z.
Router(config)#hostname R1
R1(config)#interface fastethernet 0/0
R1(config-if)#ip address 10.0.0.1 255.0.0.0
R1(config-if)#no shutdown
%LINK-5-CHANGED: Interface FastEthernet0/0,
changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/0, changed state to up
R1(config-if)#exit
R1(config)#interface fastethernet 0/1
R1(config-if)#ip address 20.0.0.1 255.0.0.0
R1(config-if)#no shutdown
%LINK-5-CHANGED: Interface FastEthernet0/1,
changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/1, changed state to up
R1(config-if)#exit
R1(config)#interface serial 0/0/0
R1(config-if)#ip address 50.0.0.1 255.0.0.0
R1(config-if)#clock rate 64000
```

```
R1(config-if)#bandwidth 64
R1(config-if)#no shutdown
%LINK-5-CHANGED: Interface Serial0/0/0, changed
state to down
R1(config-if)#exit
R1(config)#router rip
R1(config-router)#network 10.0.0.0
R1(config-router)#network 20.0.0.0
R1(config-router)#network 50.0.0.0
R1(config-router)#exit
```

(2811Router1) Hostname R2:

To configure and enable rip routing on R2 follow these commands exactly.

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.
End with CNTL/Z.
Router(config)#hostname R2
R2(config)#interface fastethernet 0/0
R2(config-if)#ip address 30.0.0.1 255.0.0.0
R2(config-if)#no shutdown
%LINK-5-CHANGED: Interface FastEthernet0/0,
changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/0,
changed state to up
R2(config-if)#exit
R2(config)#interface fastethernet 0/1
R2(config-if)#ip address 40.0.0.1 255.0.0.0
R2(config-if)#no shutdown
```

```
%LINK-5-CHANGED: Interface FastEthernet0/1,
changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/1,
changed state to up
R2(config-if)#exit
R2(config)#interface serial 0/0/0
R2(config-if)#ip address 50.0.0.2 255.0.0.0
R2(config-if)#no shutdown
%LINK-5-CHANGED: Interface Serial0/0/0, changed
state to up
R2(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface
Serial0/0/0,
changed state to up
R2(config-if)#exit
R2(config)#router rip
R2(config-router)#network 30.0.0.0
R2(config-router)#network 40.0.0.0
R2(config-router)#network 50.0.0.0
R2(config-router)#exit
```

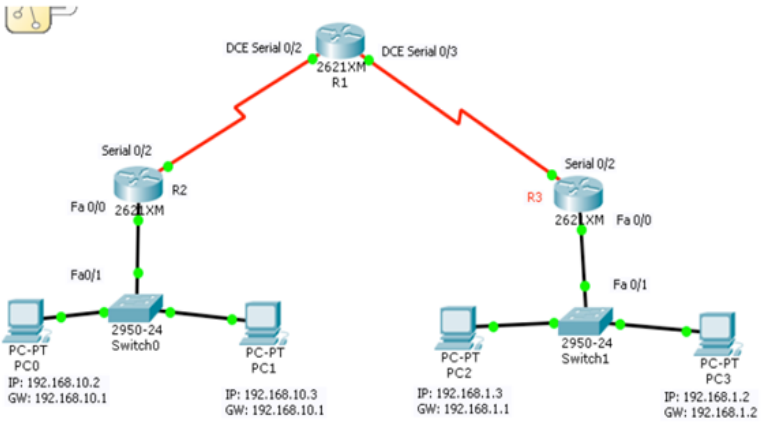
To test rip routing do ping from pc0 to all pc and vice versa. If you get replay then you have successfully configured rip routing.

Configure OSPF Routing Protocol

Configuration of OSPF routing protocol is easy as RIP Routing. The Open Shortest Path First (OSPF) is a routing protocol for wide area networks and enterprise network. OSPF is perhaps the most widely used interior gateway protocol (IGP) in large enterprise

networks. The most widely used exterior gateway protocol is the Border Gateway Protocol (BGP), the principal routing protocol between autonomous systems on the Internet.

Solved Example: Configure the following network using OSPF



Router 1

```
R1>enable
R1#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#hostname R1
R1(config)#interface serial 0/3
R1(config-if)#ip address 10.10.10.1 255.255.255.252
R1(config-if)#clock rate 64000
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#
R1(config)#interface serial 0/2
R1(config-if)#ip address 10.10.10.5 255.255.255.252
R1(config-if)#clock rate 64000
R1(config-if)#no shutdown
```

Router 2

```
Router#enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#hostname R2
R2(config)#interface serial 0/2
R2(config-if)#ip address 10.10.10.6 255.255.255.252
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface fastEthernet 0/0
R2(config-if)#ip address 192.168.10.1 255.255.255.0
R2(config-if)#no shutdown
```

Router 3

```
Router#enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#hostname R3
R3(config)#interface serial 0/2
R3(config-if)#ip address 10.10.10.2 255.255.255.252
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface fastEthernet 0/0
R3(config-if)#ip address 192.168.1.1 255.255.255.0
R3(config-if)#no shutdown
```

Configure OSPF Routing Protocol

```
R1>enable
R1#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R1(config)#router ospf 1
```

```
R1(config-router)#network 10.10.10.0 0.0.0.3 area 0
R1(config-router)#network 10.10.10.4 0.0.0.3 area 0
R1(config-router)#
```

The router OSPF command is enable OSPF routing on the router, and the 1 before OSFP is the process ID of the OSFP Protocol. You can set different process id from "1-65535" for each router. The network command with network ID "network 20.10.10.0" is the network identifier, and the "0.0.0.3" is the wildcard mask of 20.10.10.0 network. Wildcard mask determine which interfaces to advertise, because OSPF advertise interfaces, not networks.

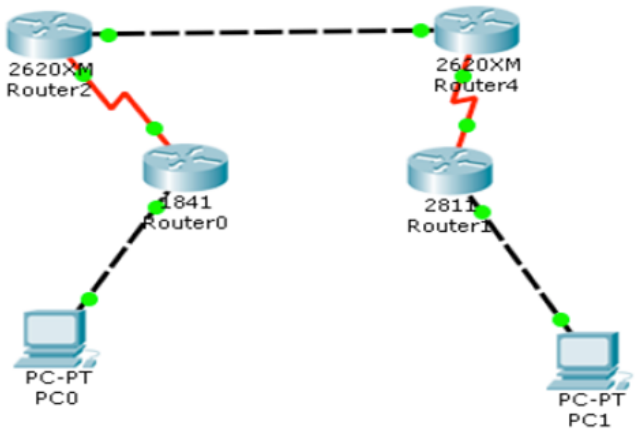
```
R2>enable
R2#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R2(config)#router ospf 1
R2(config-router)#network 192.168.10.0 0.0.0.255 area 0
R2(config-router)#network 10.10.10.4 0.0.0.3 area 0
R3>enable
R3#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R3(config)#router ospf 1
R3(config-router)#network 192.168.1.0 0.0.0.255 area 0
R3(config-router)#network 10.10.10.0 0.0.0.3 area 0
```

OSPF routing configuration has been finished successfully, now test your network whether they can ping with each other or not.

Commands	Descriptions
Router(config)#router ospf 1	Starts OSPF process 1. The process ID is any positive integer value between 1 and 65,535.
Router(config-router)#network 172.16.0.0 0.0.255.255 area 0	OSPF advertises interfaces, not networks. Uses the wildcard mask to determine which interfaces to advertise.
Router(config-if)#ip ospf hellointerval timer 20	Changes the Hello Interval timer to 20 seconds.
Router(config-if)#ip ospf deadinterval 80	Changes the Dead Interval timer to 80 seconds.
NOTE: Hello and Dead Interval timers must match for routers to become neighbors	
Router#show ip protocol	Displays parameters for all protocols running on the router
Router#show ip route	Displays a complete IP routing table
Router#show ip ospf	Displays basic information about OSPF routing processes
Router#show ip ospf interface	Displays OSPF info as it relates to all interfaces
Router#show ip ospf interface fastethernet 0/0	Displays OSPF information for interface fastethernet 0/0
Router#show ip ospf border-routers	Displays border and boundary router information
Router#show ip ospf neighbor	Lists all OSPF neighbors and their states
Router#show ip ospf neighbor detail	Displays a detailed list of neighbors
Router#clear ip route *	Clears entire routing table, forcing it to rebuild
Router#clear ip route a.b.c.d	Clears specific route to network a.b.c.d
Router#clear ip opsf counters	Resets OSPF counters
Router#clear ip ospf process	Resets entire OSPF process, forcing OSPF to re-create neighbors, database, and routing table
Router#debug ip ospf events	Displays all OSPF events
Router#debug ip ospf adjacency	Displays various OSPF states and DR/ BDR election between adjacent routers
Router#debug ip ospf packets	Displays OPSF packets

Lab Exercises

1. Using RIP, configure the below network and verify the connectivity between PC0 and PC1.



2. For the above given scenario, configure the network using OSPF and verify the connectivity between the PC's.

Lab No. 11: Configuring DHCP and NAT On A Multi-Function Device using Packet Tracer

Objectives

1. To configure DHCP using packet tracer.
2. To examine NAT on a Multi-Function Device

Network Address Translation (NAT)

The use of Network Address Translation (NAT) has been widespread for a number of years; this is because it is able to solve a number of problems with the same relatively simple configuration. At its most basic, NAT enables the ability to translate one set of addresses to another; this enables traffic coming from a specific host to appear as though it is coming from another and do it transparently.

NAT Concepts

There are a number of different concepts that must be explained in order to really get a good understanding of how NAT operates, which ultimately makes the configuration of NAT increasingly simple. This section reviews these different concepts and begins with an understanding of how NAT can be used. Some of the main uses for NAT include:

- Translation of non-unique addresses into unique addresses when accessing the Internet: This is one of the most common uses of NAT today; almost every household that has a "router" to access the Internet is using NAT on this device to translate between internal private address and public Internet addresses.
- Translation of addresses when transitioning internal addresses from one address range into another (this is common when the organization of addresses within a company is being changed): This is often done when a company is transitioning

their IP addressing plan; common scenarios include when expanding (and the IP addressing plan was not built sufficiently when the initial addresses were assigned) and when a company is merging with another with potential overlapping addresses.

- When simple TCP load sharing is required across many IP hosts: This is very common, as many highly used servers are not really a single machine but a bank of several machines that utilize load balancing. In this scenario, commonly, a single public address is translated into one of several internal addresses in a round robin fashion. This is not a complete list of every possible way that NAT can be configured but simply a list of the most common ways that it is used in modern networks.

There are a couple of main concepts that also must be reviewed and understood before configuring NAT:

- Inside and Outside Addresses
- NAT types

1. Inside and Outside Addresses In typical NAT configurations, interfaces are placed into one of two categories (or locations): inside or outside. Inside indicates traffic that is coming from within the organizational network. Outside indicates traffic that is coming from an external network that is outside the organizational network. These different categories are then used to define different types of address depending on location of the address and how it is being "seen". These different types include:

   (a) inside local address: This is the inside address as it is seen and used within the organizational network.

   (b) inside global address: This is the inside address as it is seen and used on the outside of the organizational network.

   (c) outside local address: This is the outside address as it seen and used within the organizational network.

(d) outside global address: This is the outside address as it is seen and used on the outside of the organizational network.

2. NAT Types: Another important concept to be familiar with is the different types of NAT and how they are defined. On most networks there are three different types of NAT that are defined:

   (a) Static Translation (Static NAT): This type of NAT is used when a single inside address needs to be translated to a single outside address or vice versa.

   (b) Dynamic Address Translation (Dynamic NAT): This type of NAT is used when an inside address (or addresses) need to be translated to an outside pool of addresses or vice versa.

   (c) Overloading Port Address Translation (PAT): This type of NAT is a variation on dynamic NAT. With dynamic NAT, there is always a one to one relationship between inside and outside addresses; if the outside address pool is ever exhausted, traffic from the next addresses requesting translation will be dropped. With overloading, instead of a one to one relationship, traffic is translated and given a specific outside port number to communicate with; in this situation, many internal hosts can be using the same outside address while utilizing different port numbers.

Static NAT Configuration

There a few steps that are required when configuring static NAT; the number of the commands depends on whether there will be more than one static translation:

1	Enter global configuration mode.	router#**configure terminal**
2	Configure the static NAT translation (this command can be used multiple times depending on the number of static translations required). The **overload** keyword enables the use of PAT.	router(config)#**ip nat inside source static** *local-ip global-ip* [**overload**]
3	Enter interface configuration mode for the inside interface.	router(config)#**interface** *interface-id*
4	Configure the interface as the inside NAT interface.	router(config-if)#**ip nat inside**
5	Enter interface configuration mode for the outside interface.	router(config-if)#**interface** *interface-id*
6	Configure the interface as the outside NAT interface.	router(config-if)#**ip nat outside**
7	Exit configuration mode.	router(config-if)#**end**

Using NAT we can hide real IP address, we can translate private IP address to public IP address and vice versa. As we all know in internet only public IP addresses are used and some IP in every class has been reserved for use in LOCAL AREA CONNECTION say LAN and these ranges of IP are known as Private IP Address. Private Addresses can only be used in LAN and it can't be used in internet. But our PC with private address can communicate with PC or Machine having public IP address using NAT(Network Address Translation).

A public IP address is an IP address that can be accessed over the Internet. Like postal address used to deliver a postal mail to your home, a public IP address is the globally unique IP address assigned to a computing device. Private IP address on the other hand is used to assign computers within your private space without letting them directly expose to the Internet. For example, if you have multiple computers within your home you may want to use private IP addresses to address each computer within your home. In this scenario, your router get the public IP address, and each of the computers, tablets and smartphones connected to your router (via wired or wifi) get a private IP address from your router via DHCP protocol.

Internet Assigned Numbers Authority (IANA) is the organization responsible for registering IP address ranges to organizations and Internet Service Providers (ISPs). To

allow organizations to freely assign private IP addresses, the Network Information Center (InterNIC) has reserved certain address blocks for private use.

The following IP blocks are reserved for private IP addresses.

Class	Starting IP Address	Ending IP Address	# of Hosts
A	10.0.0.0	10.255.255.255	16,777,216
B	172.16.0.0	172.31.255.255	1,048,576
C	192.168.0.0	192.168.255.255	65,536

What is public IP address?

A public IP address is the address that is assigned to a computing device to allow direct access over the Internet. A web server, email server and any server device directly accessible from the Internet are candidate for a public IP address. A public IP address is globally unique, and can only be assigned to an unique device.

What is private IP address?

A private IP address is the address space allocated by InterNIC to allow organizations to create their own private network. There are three IP blocks (1 class A, 1 class B and 1 class C) reserved for a private use. The computers, tablets and smartphones sitting behind your home, and the personal computers within an organizations are usually assigned private IP addresses. A network printer residing in your home is assigned a private address so that only your family can print to your local printer.

When a computer is assigned a private IP address, the local devices sees this computer via it's private IP address. However, the devices residing outside of your local network cannot directly communicate via the private IP address, but uses your router's public IP address to communicate. To allow direct access to a local device which is assigned a private IP address, a Network Address Translator (NAT) should be used.

How to configure NTP Server and NTP Client on cisco Router

Whenever we talks about cisco routers and how to set clock or accurate time on routers it becomes so important because a variety of services depend on it. We need to monitor our routers and secure our routers through server configuration. We use to configure syslog server for monitoring routers log and incident happening over routers. The logging service shows each log entry with the date and time and all details which are directly or indirectly related to NTP Server. It becomes very critical if you're trying to track a specific incident or troubleshoot a problem.
Cisco routers have two types of clocks :

1. A battery-powered hardware clock, referenced as the 'calendar' in the IOS CLI, and
2. A software clock, referenced as the 'clock' in the IOS CLI.

The software clock is the primary source for time data and runs from the moment the system is up and running. The software clock can be updated from a number of sources.

Configure DHCP on a Router

DHCP stands for Dynamic Host Configuration Protocol, as the name indicates it dynamically controls the hosts. DHCP is very common in home networks and in most enterprise networks. DHCP (Dynamic Host Configuration Protocol) is a networking protocol that permits network administrators centrally manage and automate the assignment of network parameters to a client device like PCs or Laptops.

1. DHCPDISCOVER: Server Discovery – Broadcast

   In server discovery process, when a computer or other networked device connects to a DHCP network, it starts trying to discover whether there is any DHCP server

available in the network by sending broadcast to 255.255.255.255 destination address. This is called DHCPDISCOVER.

2. DHCPOFFER: IP Lease Offer – Unicast

   After the DHCP server accepts the DHCPDISCOVER message, it replies with a DHCPOFFER message. This could be Unicast to the MAC address of the client and those packet includes an IPv4 address lease among the pool of IPs, Subnet mask, Gateway, DNS, lease duration.

3. DHCPREQUEST: IP request – Broadcast

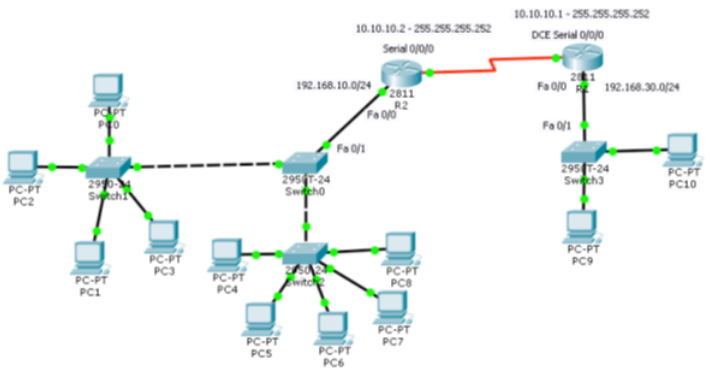
   Clients take the first offer received from DHCP server by broadcasting a DHCP Request packet called DHCPREQUEST. This message allowing the server to know that the client supposed to use the address offered by the server.

4. DHCPACK: IP lease acknowledgment – Unicast

   The server directs an acknowledgment (DHCPACK) message in unicast, confirming that acceptance of the allotted IP for a specified period of time. It also used to renew the lease time.

Solved Example:

1. Configure DHCP on routers for the given topology



2. Configure the Router 1

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#hostname R1
R1(config)#interface serial 0/0/0
R1(config-if)#ip address 10.10.10.1 255.255.255.252
R1(config-if)#no shutdown
R1(config-if)#clock rate 64000
R1(config-if)#
```

3. Configure the Router 2

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#hostname R2
R2(config)#interface serial 0/0/0
R2(config-if)#ip address 10.10.10.2 255.255.255.252
R2(config-if)#no shutdown
R2(config-if)#
R2(config)#interface fastEthernet 0/0
R2(config-if)#ip address 192.168.10.1 255.255.255.0
R2(config-if)#no shutdown
```

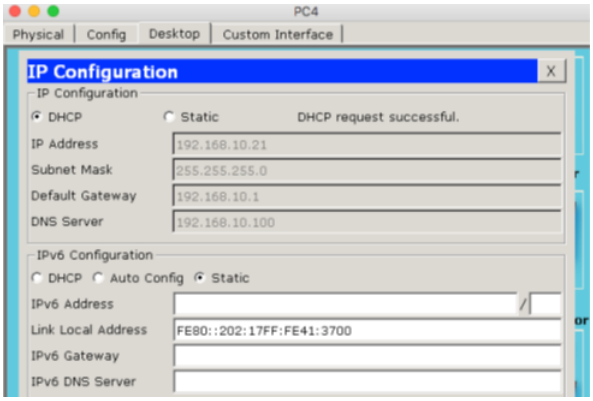
4. Lets config Router 2 as DHCP Server and set the clients to get there IP addresses from DHCP Server.

   In the R2 while you are in the config mode, type the command 'ip dhcp excluded-address 192.168.10.1 192.168.10.20' and then press enter. This command 'ip dhcp excluded-address' will create an exclusive range of IP addresses which reserved for Network Servers and DHCP Server will not assign theme to clients.

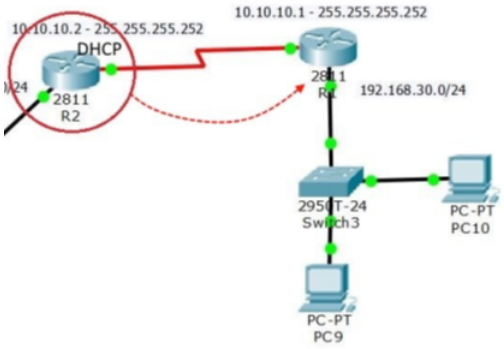
The 'ip dhcp pool' command create a pool for a network. You can create many pools on a router for all Local area network that connected to the router.

```
R2>enable
R2#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R2(config)#ip dhcp excluded-address 192.168.10.1 192.168.10.20
R2(config)#ip dhcp pool mynetwork (you can give any name)
R2(dhcp-config)#
R2(dhcp-config)#network 192.168.10.0 255.255.255.0
R2(dhcp-config)#default-router 192.168.10.1
R2(dhcp-config)#dns-server 192.168.10.100
R2(dhcp-config)#
```

5. Now go to client setting and set the IP Configuration to DHCP and see the client get new IP address from DHCP Server.



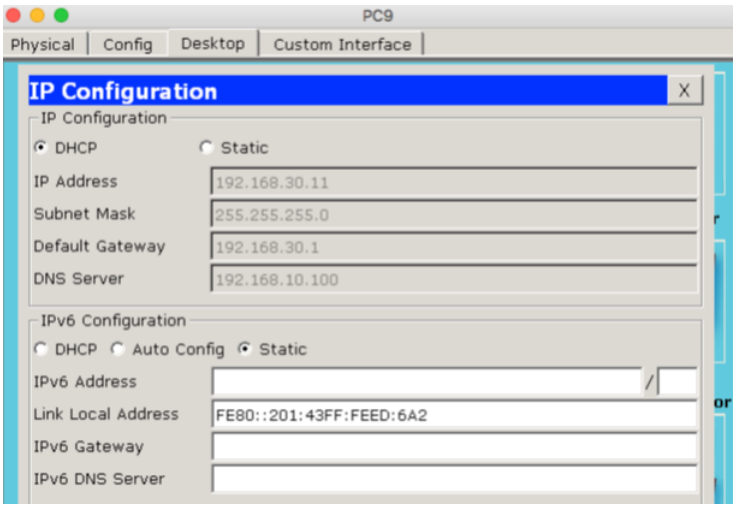
6. DHCP Options on Cisco Router: Remember some DHCP options when you need to provide IP addresses from a DCHP server to clients that are outside of your network or are not in the same Local Area Network. You must use the 'ip helper-address' to forward the DHCP client requests to remote host.



Configure the R1 to relay the DHCP client request. It will not work without routing. So configure Routers with static or dynamic routing. Here I'm testing with RIP.

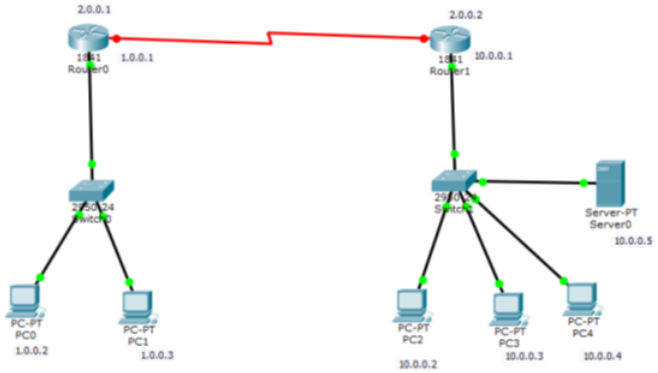
```
R1>enable
R1#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
R1(config)#interface fastEthernet 0/0
R1(config-if)#ip helper-address 10.10.10.2
R1(config-if)#exit
R1(config)#router rip
R1(config-router)#network 10.10.10.0
R1(config-router)#network 192.168.30.0
R1(config-router)#exit
```

7. Go to the client IP configuration setting and see the forwarded request by DHCP Server.



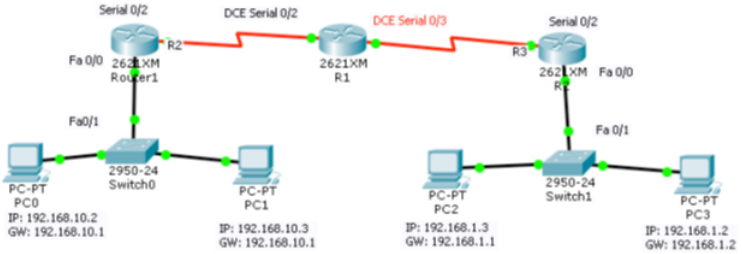
Lab Exercises

1. Configure static NAT for the below given scenario. Also ping a private IP address given and mention your inference.



2. Perform NTP client-server configuration for the below given topology

3. Configure DHCP for the following topology.



Lab No. 12: Wireless Topology and VOIP using Packet Tracer

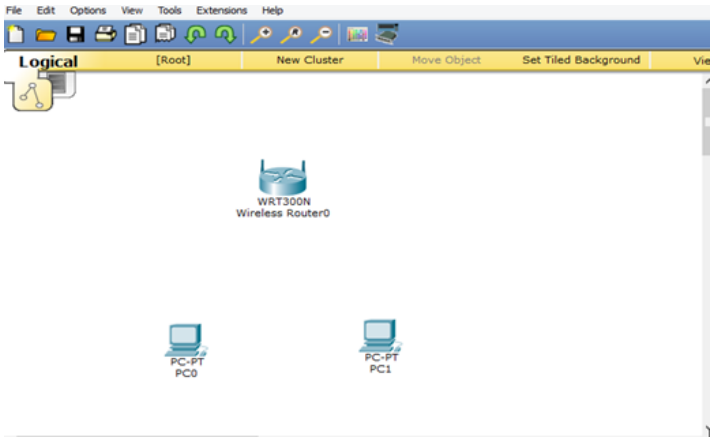
Objectives

1. To create and configure wireless network using packet tracer
2. To configure VOIP

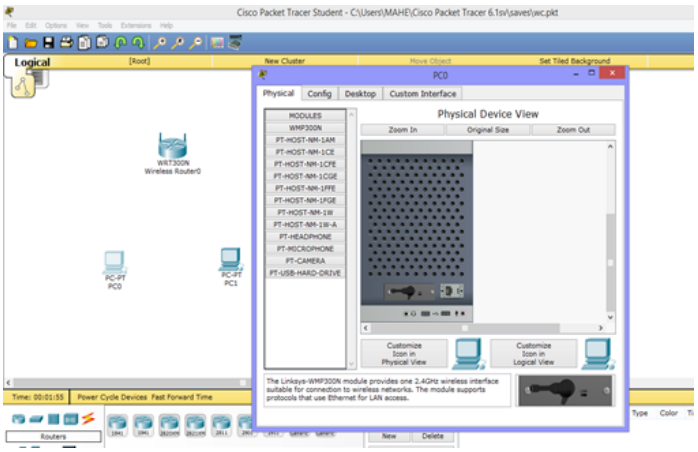
Introduction

Let us create wireless topology on packet tracer.

For this go to the wireless devices and select wireless router, choose some PCs and provide them with wireless module so that they can communicate through router wirelessly.

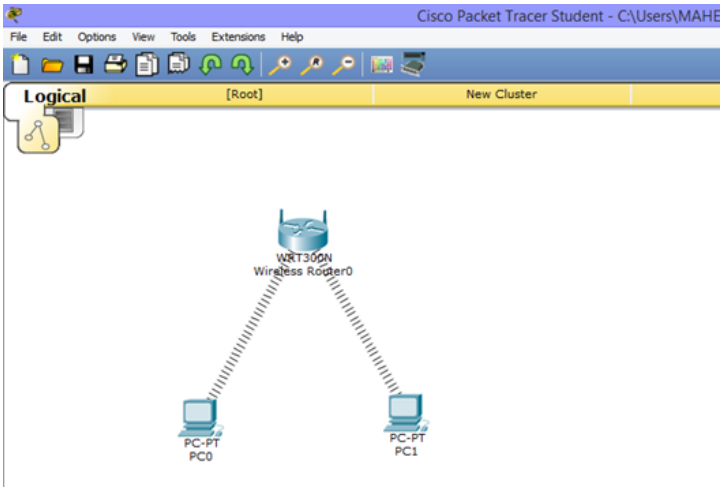


For providing wireless module, go to the PC physical mode Go to PC, and remove wired LAN and install WMP300N Module.

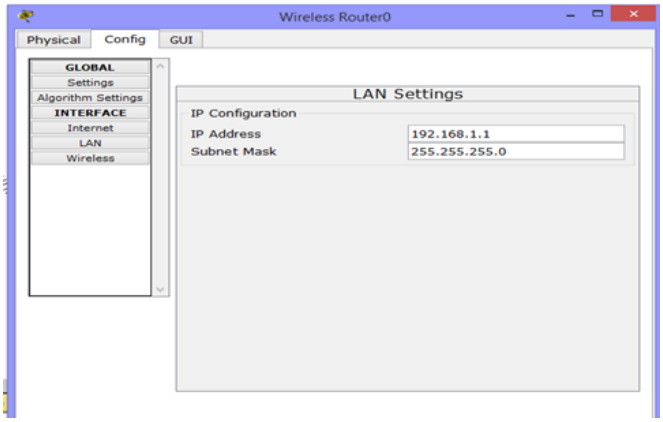


Repeat the procedure for PC1

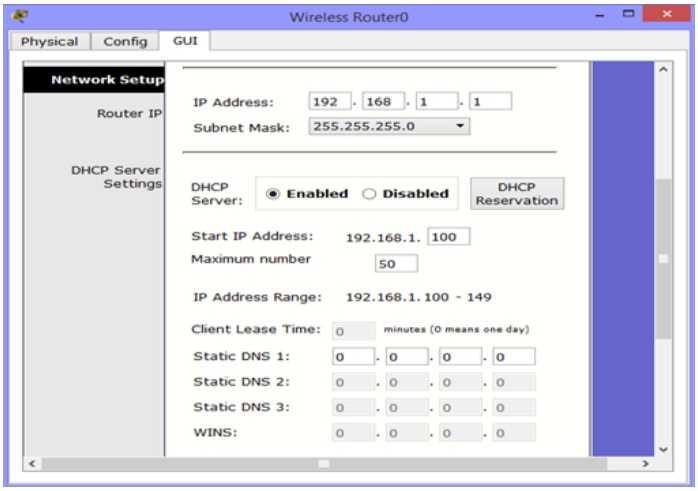
Now devices are connected .



Set the IP address for the router.

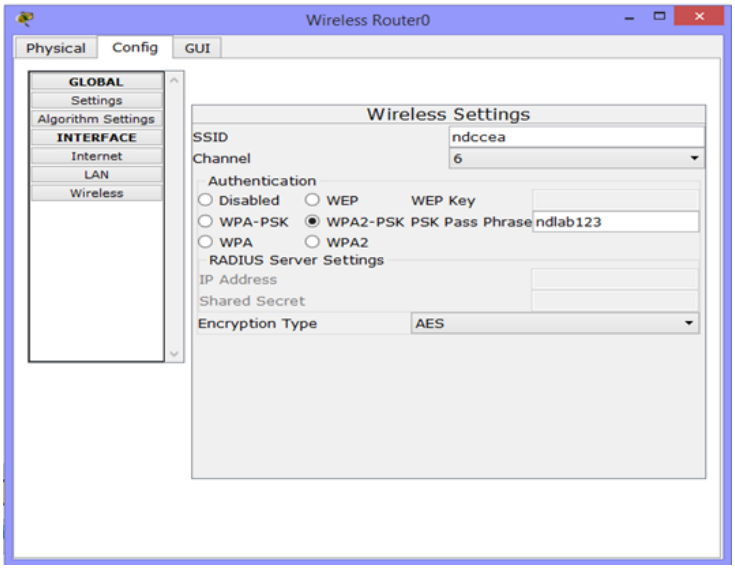


As this wireless router provides us with the DHCP service, so we can obtain IP automatically by using this service for our PCs.

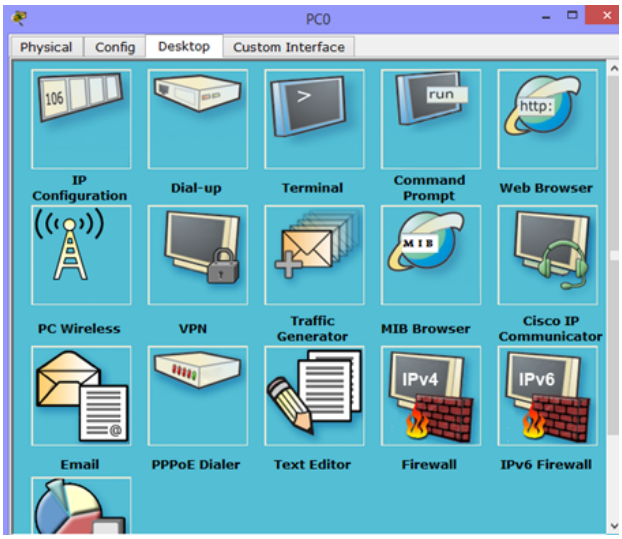


This enables the devices to communicate.

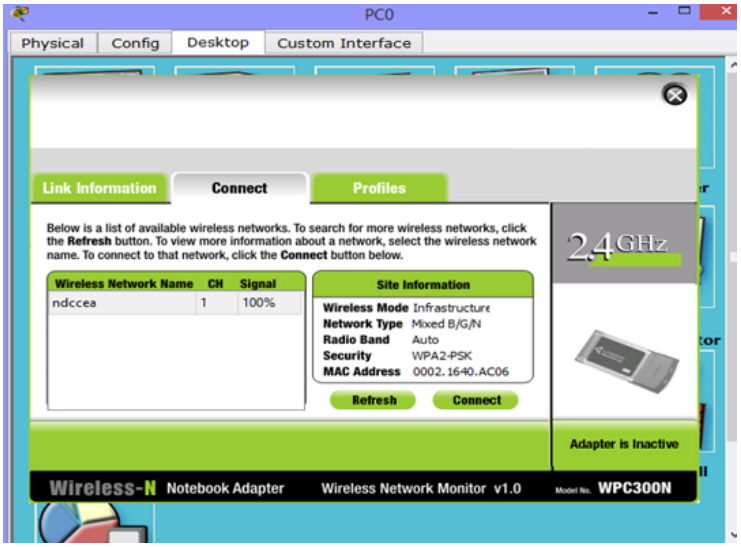
Now, let us apply authentication to our wireless router. For that, go to Config tab, click on Wireless. Provide it with the information (SSID and Passkey) as below.



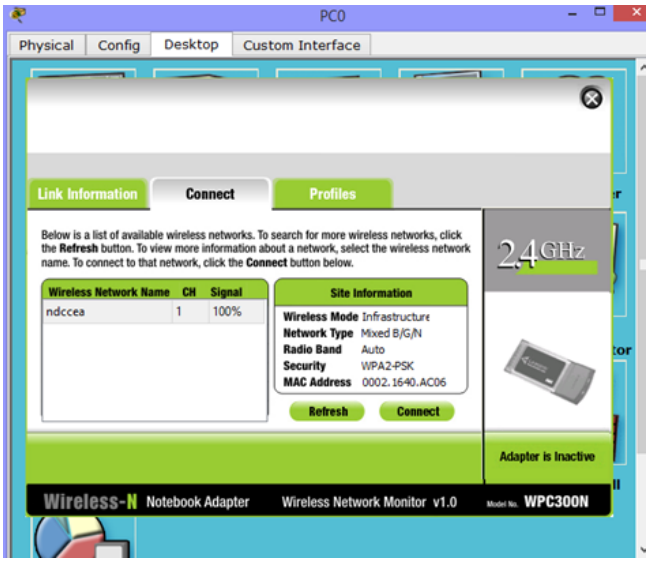
Go to PC desktop mode, Click on PC Wireless.



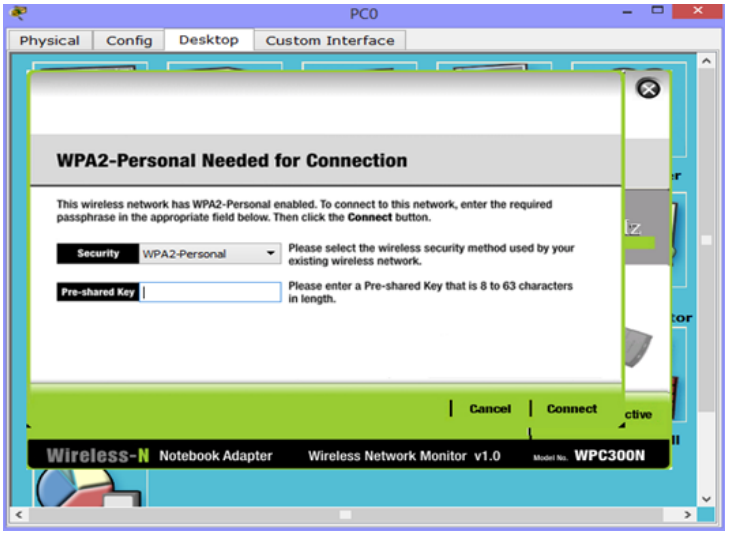
Go to PC desktop mode, Click on PC Wireless.



Go to the connect tab and click refresh.



Now click on connect button and give the passkey you have set in the router.



Now you can see that the PC has been connected to the router, you can also check the signal strength and quality by clicking on Link Information.

Repeat the procedure for PC2.

VOIP(Voice Over IP)

VOIP is an acronym for Voice Over Internet Protocol, or in more common terms phone service over the Internet.

. If you have a reasonable quality Internet connection you can get phone service delivered through your Internet connection instead of from your local phone company. Some people use VOIP in addition to their traditional phone service, since VOIP service providers usually offer lower rates than traditional phone companies, but sometimes doesn't offer 911 service, phone directory listings, 411 service, or other common phone services. While many VoIP providers offer these services, consistent industry-wide means of offering these are still developing.

How does VOIP work? A way is required to turn analog phone signals into digital signals that can be sent over the Internet.This function can either be included into the phone itself or in a separate box like an ATA.

VOIP Using an ATA

Ordinary Phone —- ATA —- Ethernet —- Router —- Internet —- VOIP Service Provider

VOIP using an IP Phone

IP Phone —– Ethernet —– Router —- Internet —- VOIP Service Provider

VOIP connecting directly It is also possible to bypass a VOIP Service Provider and directly connect to another VOIP user.

However, if the VOIP devices are behind NATrouters, there may be problems with this approach

. IP Phone —– Ethernet —– Router —- Internet —- Router —- Ethernet —- IP Phone

Applications using VOIP

Traditional telephony applications, such as outbound call center applications and inbound IVR applications, normally can be run on VOIP

Why use VOIP? There are two major reasons to use VOIP

1. Lower Cost

   In general phone service via VOIP costs less than equivalent service from traditional sources. This is largely a function of traditional phone services either being monopolies or government entities. There are also some cost savings due to using a single network to carry voice and data. This is especially true when users have existing under-utilized network capacity that they can use for VOIP without any additional costs.

   . In the most extreme case, users see VOIP phone calls (even international) as FREE. While there is a cost for their Internet service, using VOIP over this service may not involve any extra charges, so the users view the calls as free. There are a number of services that have sprung up to facilitate this type of ”free” VOIP call. Examples are: FreeWorldDialup and Skype.

2. Increased Functionality

   VOIP makes easy some things that are difficult to impossible with traditional phone networks. Incoming phone calls are automatically routed to your VOIP phone where ever you plug it into the network. Take your VOIP phone with you on a trip, and

anywhere you connect it to the Internet, you can receive your incoming calls.

Call center agents using VOIP phones can easily work from anywhere with a good Internet connection.

Lab Exercises

1. In the below topology, we have three pc connected with Linksys Wireless routers.

   (a) DHCP is configured and enabled on Wireless router

   (b) IP pool for DHCP is 192.168.0.100 to 192.168.0.150

   (c) PC are configured to receive IP from DHCP Server

   (d) No security is configured

   (e) Default SSID is configured to Default

   (f) Topology is working on infrastructure mode

   (g) Default user name and password is admin

   (h) IP of wireless is set to 192.168.0.1

   Given the above information, perform the following tasks.

   (a) Configure Static IP on PC and Wireless Router

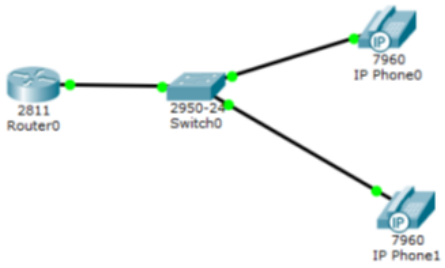
(b) Change SSID to MotherNetwork

(c) Change IP address of router to 10.0.0.1 and 10.0.0.2 of PC0 10.0.0.3 of PC1 10.0.0.4 of PC2

(d) Secure your network by configuring WAP key on Router

(e) Connect PC by using WAP key

2. Configure VOIP for the below given topology and call IP Phone1 from IP Phone0.



References

1. Stevens R., Stephen A. R., Advanced Programming in the UNIX Environment (2e), Pearson Education, 2013.

2. Jesin A, Packet Tracer Network Simulator (1e), Packt Publishing, 2014.