Part- A

1. Write a program for error detecting code using CRC-CCITT (16- bits).

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The CRC calculation or cyclic redundancy check was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation— by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

```
         1 0 1
         ----------------
10011 /  1 1 0 1 1 0 1
1 0 0 1 1 | |
--------- —---| |
1 0 0 0 0 |
0 0 0 0 0 |
-------------- |
1 0 0 0 0 1
1 0 0 1 1
-----------------
1 1 1 0 = 14 remainder
```

- The message bits are appended with c zero bits; this augmented message is the dividend
- A predetermined c+1-bit binary sequence, called the generator polynomial, is the divisor
- The checksum is the c-bit remainder that results from the division operation

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an exclusive or operation.

Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:
Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder.
So, for example, you‘d use a 17-bit generator polynomial whenever a 16-bit checksum is required.

	CRC - CCITT	**CRC – 16**	**CRC - 32**
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	100000100110000010001110110110111

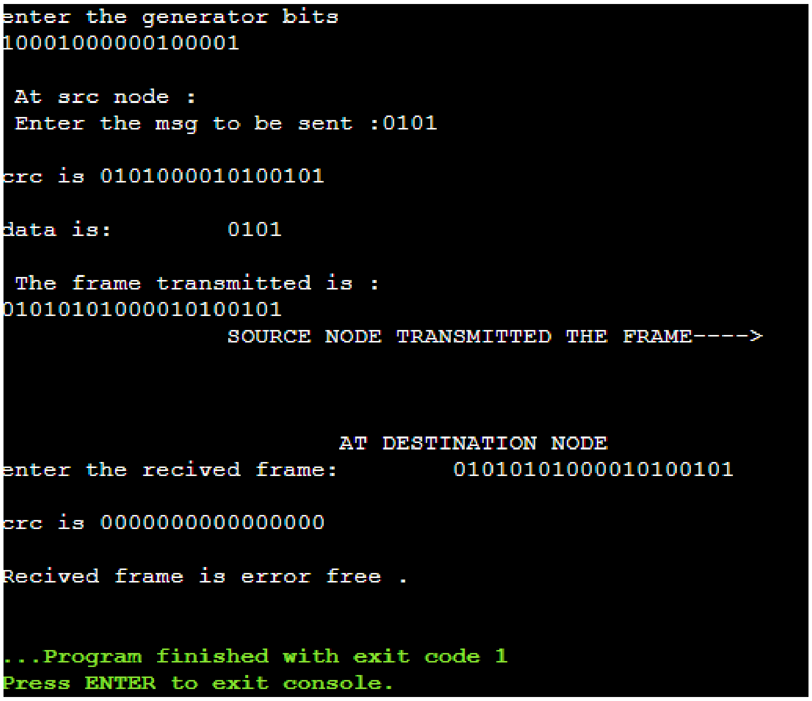
International Standard CRC Polynomials

Program:

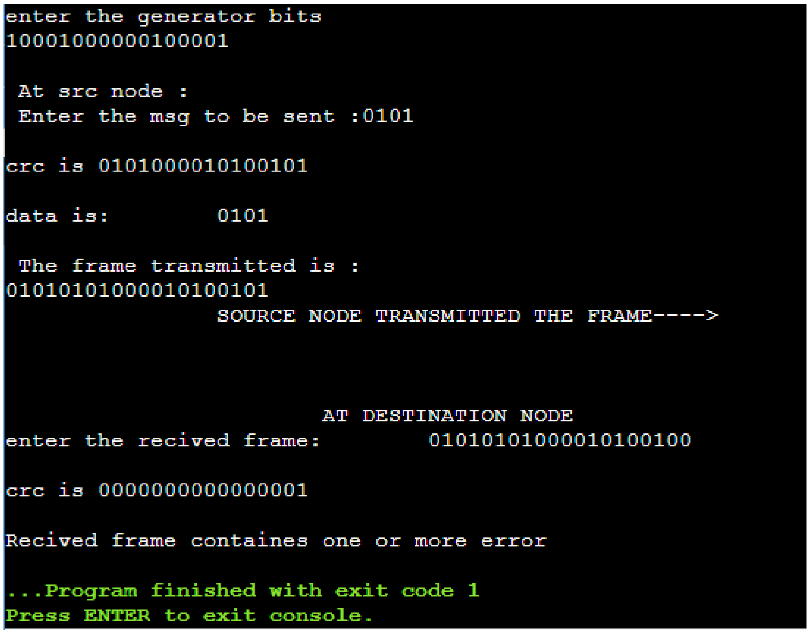
```
#include<stdio.h>

#include<string.h>


char data[100],concatdata[117],src_crc[17],dest_crc[17],frame[120],divident[18];

char divisor[18];

char res[17]="0000000000000000";


void crc_cal(int node)

{

int i,j;

for(j=17;j<=strlen(concatdata);j++)

{

        if(divident[0]=='1')

        {

        for(i=1;i<=16;i++)

        if(divident[i]!=divisor[i])

        divident[i-1]='1';

        else

        divident[i-1]='0';

        }
        else
        {
        for(i=1;i<=16;i++)
        divident[i-1]=divident[i];
        }
        if(node==0)
        divident[i-1]=concatdata[j];
        else
        divident[i-1]=frame[j];
}
divident[i]='\0';
printf("\ncrc is %s\n",divident);
if(node==0)
{
strcpy(src_crc,divident);
}
else
strcpy(dest_crc,divident);
}
int main()
{
int i;
printf("enter the generator bits\n");
gets(divisor);
if(strlen(divisor)<17 || strlen(divisor)>17)
{
printf("please enter the geneartor length of 17 bits\n");
exit(0);
}
printf("\n At src node :\n Enter the msg to be sent :");
gets(data);
strcpy(concatdata,data);
strcat(concatdata,"0000000000000000");
```

```
for(i=0;i<=16;i++)
divident[i]=concatdata[i];
divident[i]='\0';
crc_cal(0);
printf("\ndata is:\t");
puts(data);
printf("\n The frame transmitted is :\t");
printf("\n%s%s",data,src_crc);
printf("\n\t\tSOURCE NODE TRANSMITTED THE FRAME---->");
printf("\n\n\n\n\t\t\tAT DESTINATION NODE\nenter the recived frame:\t");
gets(frame);
for(i=0;i<=16;i++)
divident[i]=frame[i];
divident[i]='\0';
crc_cal(1);
if((strcmp(dest_crc,res))==0)
printf("\nRecived frame is error free .\n ");
else
printf("\nRecived frame containes one or more error ");
return 1;
}
```

Output:



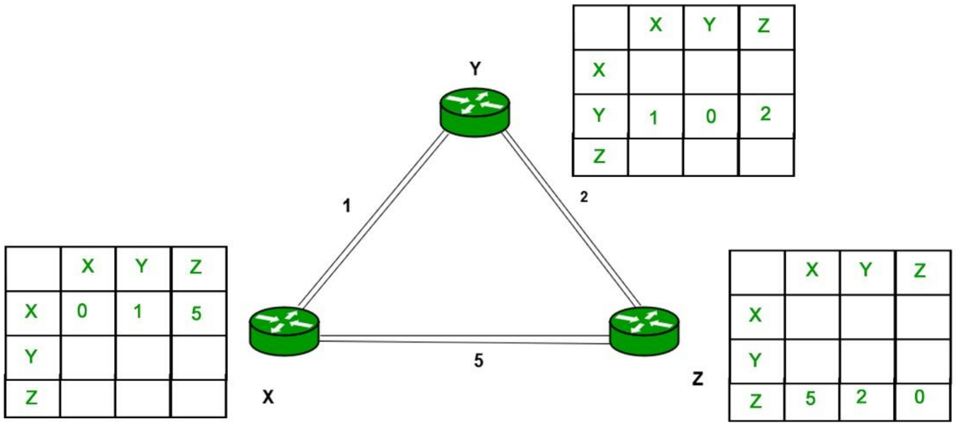
```
enter the generator bits
10001000000100001

 At src node :
 Enter the msg to be sent :0101

crc is 0101000010100101

data is:         0101

 The frame transmitted is :
01010101000010100101
                SOURCE NODE TRANSMITTED THE FRAME---->



                        AT DESTINATION NODE
enter the recived frame:        01010101000010100101

crc is 0000000000000000

Recived frame is error free .


...Program finished with exit code 1
Press ENTER to exit console.
```



```
enter the generator bits
10001000000100001

 At src node :
 Enter the msg to be sent :0101

crc is 0101000010100101

data is:         0101

 The frame transmitted is :
01010101000010100101
                SOURCE NODE TRANSMITTED THE FRAME---->



                        AT DESTINATION NODE
enter the recived frame:        01010101000010100100

crc is 0000000000000001

Recived frame containes one or more error

...Program finished with exit code 1
Press ENTER to exit console.
```

2. Write a program to find the shortest path between vertices using Distance vector algorithm.

Consider the below network:



	X	Y	Z
X	0	1	5
Y			
Z			

	X	Y	Z
X			
Y	1	0	2
Z			

	X	Y	Z
X			
Y			
Z	5	2	0

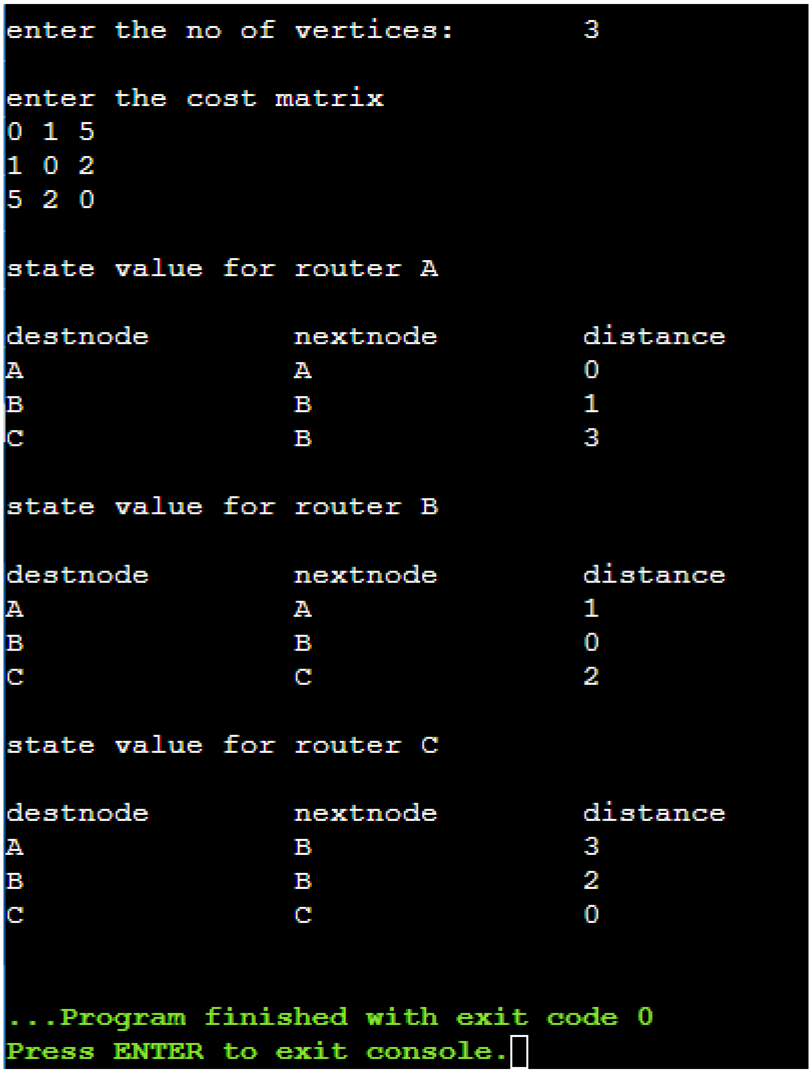
Algorithm:

```
At each node x,

Initialization

for all destinations y in N:
$D_x(y) = c(x,y)$     // If y is not a neighbor then $c(x,y) = \infty$
for each neighbor w
$D_w(y) = ?$     for all destination y in N.
for each neighbor w
send distance vector $D_x = [\ D_x(y)\ : y \text{ in } N\ ]$ to w
loop
  wait(until I receive any distance vector from some neighbor w)
  for each y in N:
  $D_x(y) = \min_v\{c(x,v)+D_v(y)\}$
If $D_x(y)$ is changed for any destination y
Send distance vector $D_x = [\ D_x(y) : y \text{ in } N\ ]$ to all neighbors
```

Program:

```
#include<stdio.h>
struct rtable
{
int dist[20],nextnode[20];
}table[20];
 int cost[10][10],n;
 void distvector()
{
int i,j,k,count=0;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
table[i].dist[j]=cost[i][j];
table[i].nextnode[j]=j;
}          }
do
{
count=0;
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
for(k=0;k<n;k++)
{
if(table[i].dist[j]>cost[i][k]+table[k].dist[j])
{
table[i].dist[j]=table[i].dist[k]+table[k].dist[j];
table[i].nextnode[j]=k;
count++;
}
}
}
}
}while(count!=0);
}
int main()
{
int i,j;
printf("\nenter the no of vertices:\t");
scanf("%d",&n);
printf("\nenter the cost matrix\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&cost[i][j]);
distvector();
for(i=0;i<n;i++)
{
printf("\nstate value for router %c \n",i+65);
printf("\ndestnode\tnextnode\tdistance\n");
for(j=0;j<n;j++)
{
if(table[i].dist[j]==99)
printf("%c\t\t-\t\t infinite\n",j+65);
else
printf("%c\t\t%c\t\t%d\n",j+65,table[i].nextnode[j]+65,table[i].dist[j]);
}
}
return 0;
 }
```

Output:



```
enter the no of vertices:       3

enter the cost matrix
0 1 5
1 0 2
5 2 0

state value for router A

destnode        nextnode        distance
A               A               0
B               B               1
C               B               3

state value for router B

destnode        nextnode        distance
A               A               1
B               B               0
C               C               2

state value for router C

destnode        nextnode        distance
A               B               3
B               B               2
C               C               0


...Program finished with exit code 0
Press ENTER to exit console.
```

3. Using TCP sockets, write a client – server program to make the interaction between the client and the server.

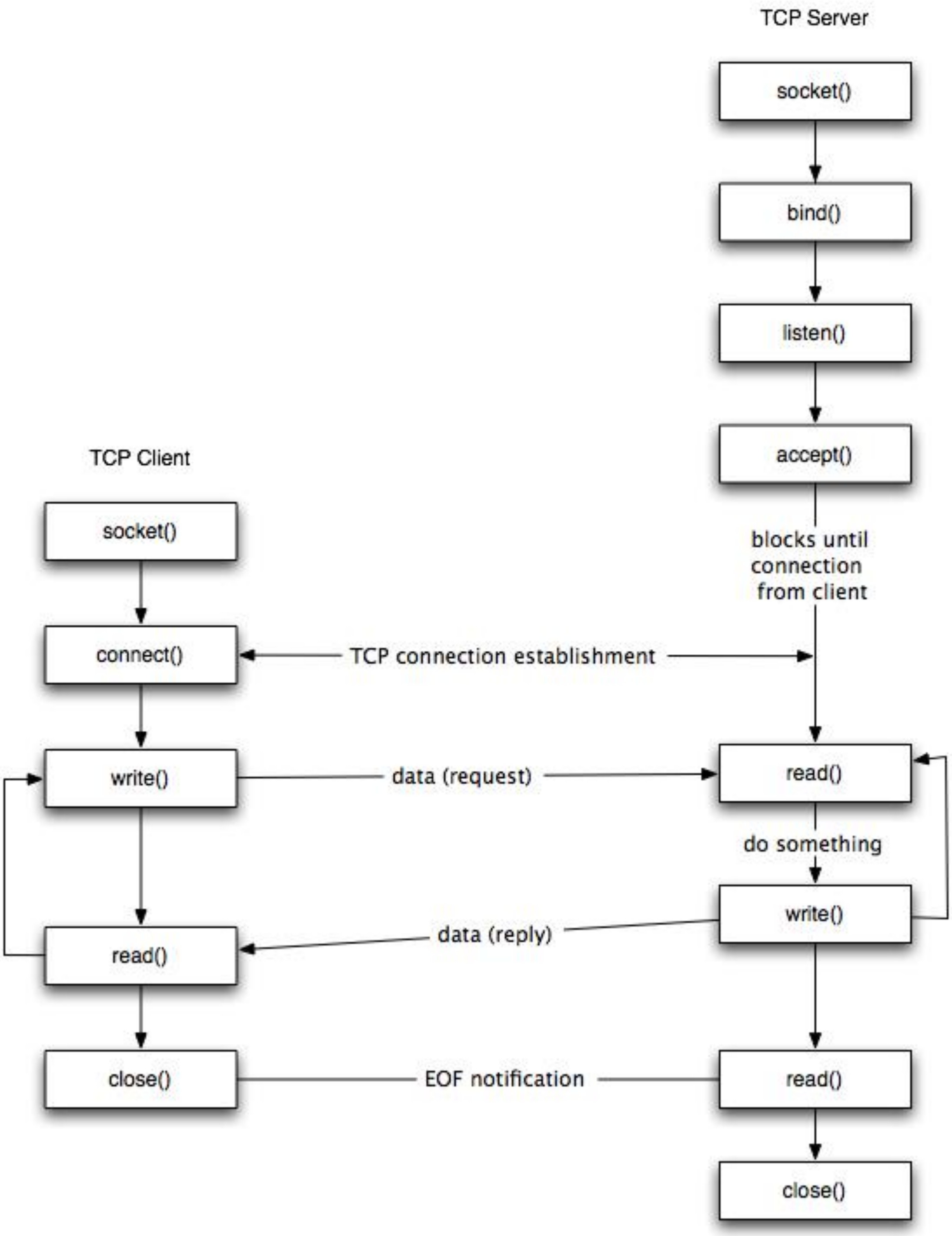


Fig: TCP Socket lifecycle diagram.

Stages for server

1. Socket creation:

int sockfd = socket(domain, type, protocol)

- **sockfd:** socket descriptor, an integer (like a file-handle)
- **domain:** integer, specifies communication domain. We use AF_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF_INET and AF_I NET 6 for processes connected by IPV6.
- **type:** communication type
  SOCK STREAM: TCP(reliable, connection oriented)
  SOCK_DGRAM: UDP(unreliable, connectionless)
- **protocol:** Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

2. Setsockopt: This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

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

3. Bind:

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

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

4. Listen:

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

5. Accept:

*int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);*

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Code for TCP socket Server

```
#include<stdio.h>

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

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/fcntl.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
int fd,sockfd,newsockfd,clilen,portno,n;
struct sockaddr_in seradd,cliadd;
char buffer[4096];
if(argc<2)
{
fprintf(stderr,"\n\n No port\n");
exit(1);
}
portno=atoi(argv[1]);
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
error("\n error opening socket.\n");
bzero((char *)&seradd,sizeof(seradd));
seradd.sin_family=AF_INET;
seradd.sin_addr.s_addr=(htonl)INADDR_ANY;
seradd.sin_port=htons(portno);
if(bind(sockfd,(struct sockaddr *)&seradd,sizeof(seradd))<0)
perror("\n IP addr cannt bind");
listen(sockfd,5);
clilen=sizeof(cliadd);
printf("\n Server waiting for clint request\n");
while(1)
{
newsockfd=accept(sockfd,(struct sockaddr *)&cliadd,&clilen);
if(newsockfd<0)
perror("\n Server cannot accept connection request ");
bzero(buffer,4096);
read(newsockfd,buffer,4096);
fd=open(buffer,O_RDONLY);
if(fd<0)
perror("\n File  doesnot exist");
while(1)
{
n=read(fd,buffer,4096);
if(n<=0)
exit(0);
write(newsockfd,buffer,n);
printf("\n File transfer completet\n");
}
close(fd);
close(newsockfd);
}
return 0;
}
```

Stages for Client

- **Socket connection:** Exactly same as that of server's socket creation
- **Connect:** The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Code for TCP socket client:

```
#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<sys/fcntl.h>

#include<stdlib.h>

#include<string.h>

#include<arpa/inet.h>


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

{

int sockfd,portno,n;

struct sockaddr_in seradd;

char buffer[4096],*serip;

if(argc<4)

{

fprintf(stderr,"usage %s serverip filename port",argv[0]);

exit(0);

}

serip=argv[1];

portno=atoi(argv[3]);

sockfd=socket(AF_INET,SOCK_STREAM,0);

if(sockfd<0)

perror("\n Error in creating socket.\n");

perror("\n Client on line.");

bzero((char *)&seradd,sizeof(seradd));

seradd.sin_family=AF_INET;

seradd.sin_addr.s_addr=inet_addr(serip);

seradd.sin_port=htons(portno);

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

perror("\n Error in connection setup \n");
write(sockfd,argv[2],strlen(argv[2])+1);
bzero(buffer,4096);
n=read(sockfd,buffer,4096);
if(n<=0)
{
perror("\n File not found");
exit(0);
}
write (1,buffer,n);
}
```

Output:

cc client.c

./a.out

```
Client:Hello  message  sent
Hello  from server
```

cc server.c

./a.out

```
Server:Hello   from client
Hello  message  sent
```

4. Implement the above program using as message queues or FIFOs as IPC channels.

Algorithm (Client Side)

1.Start.

2.Open well known server FIFO in write mode.

3.Write the pathname of the file in this FIFO and send the request.

4.Open the client specified FIFO in read mode and wait for reply.

5.When the contents of the file are available in FIFO, display it on the terminal

6.Stop.

Algorithm (Server Side)

1.Start.

2.Create a well known FIFO using mkfifo()

3.Open FIFO in read only mode to accept request from clients.

4.When client opens the other end of FIFO in write only mode, then read the contents and store it in buffer.

5.Create another FIFO in write mode to send replies.

6.Open the requested file by the client and write the contents into the client specified FIFO and terminate the connection.

7.Stop.

Server.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#define FIFO1 "fifo1"
#define FIFO2 "fifo2"

int main()
{
char p[100],c[5000,ch;
int num,fd,fd2,f1;
mknod(FIFO1,S_IFIFO|0666,0);
mknod(FIFO2,S_IFIFO|0666,0);

printf("\n Server online...\n");

fd=open(FIFO1,O_RDONLY);

fd2=open(FIFO2,O_WRONLY);

printf("Server online\n waiting for client \n\n");

if((num=read(fd,p,100))==-1)

perror("\n Read Error ");

else

{

p[num]='\0';

printf("\n File is %s .\n",p);

if((f1=open(p,O_RDONLY))<0)

{

write(fd2,"File not found",15);

return 1;

}

else

{

stdin=fdopen(f1,"r");

num=0;

while((ch=fgetc(stdin))!=EOF)

c[num++]=ch;

c[num]=0;

printf(" Server: Transfering the contents of :%s ",p);

if(num=write(fd2,c,strlen(c))==-1)

printf("\n Error in writting to FIFO\n");
else

printf("\n File transfer completed \n");

}}}
```

Client.c

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<unistd.h>

#define FIFO1 "fifo1"

#define FIFO2 "fifo2"


int main()

{

char p[100],c[5000,ch;

int num,fd,fd2,f1;

mknod(FIFO1,S_IFIFO|0666,0);

mknod(FIFO2,S_IFIFO|0666,0);

printf("\n Server online...\n");

fd=open(FIFO1,O_RDONLY);

fd2=open(FIFO2,O_WRONLY);

printf("Server online\n waiting for client \n\n");

if((num=read(fd,p,100))==-1)

perror("\n Read Error ");
else
{
p[num]='\0';
printf("\n File is %s .\n",p);
if((f1=open(p,O_RDONLY))<0)
{
write(fd2,"File not found",15);
return 1;
}
else
{
stdin=fdopen(f1,"r");
num=0;
while((ch=fgetc(stdin))!=EOF)
c[num++]=ch;
c[num]=0;
printf(" Server: Transfering the contents of :%s ",p);
if(num=write(fd2,c,strlen(c))==-1)
printf("\n Error in writting to FIFO\n");
else
printf("\n File transfer completed \n");
}}}
```

SERVER OUTPUT

```
[root@localhost ~]# vi fifo_server.c

[root@localhost ~]# cc -o fifo_server fifo_server.c
```

```
[root@localhost ~]# ./fifo_serverWaiting for connection Request...Connection Established...Client has requested file dvr.c

[root@localhost ~]#
```

CLIENTOUTPUT

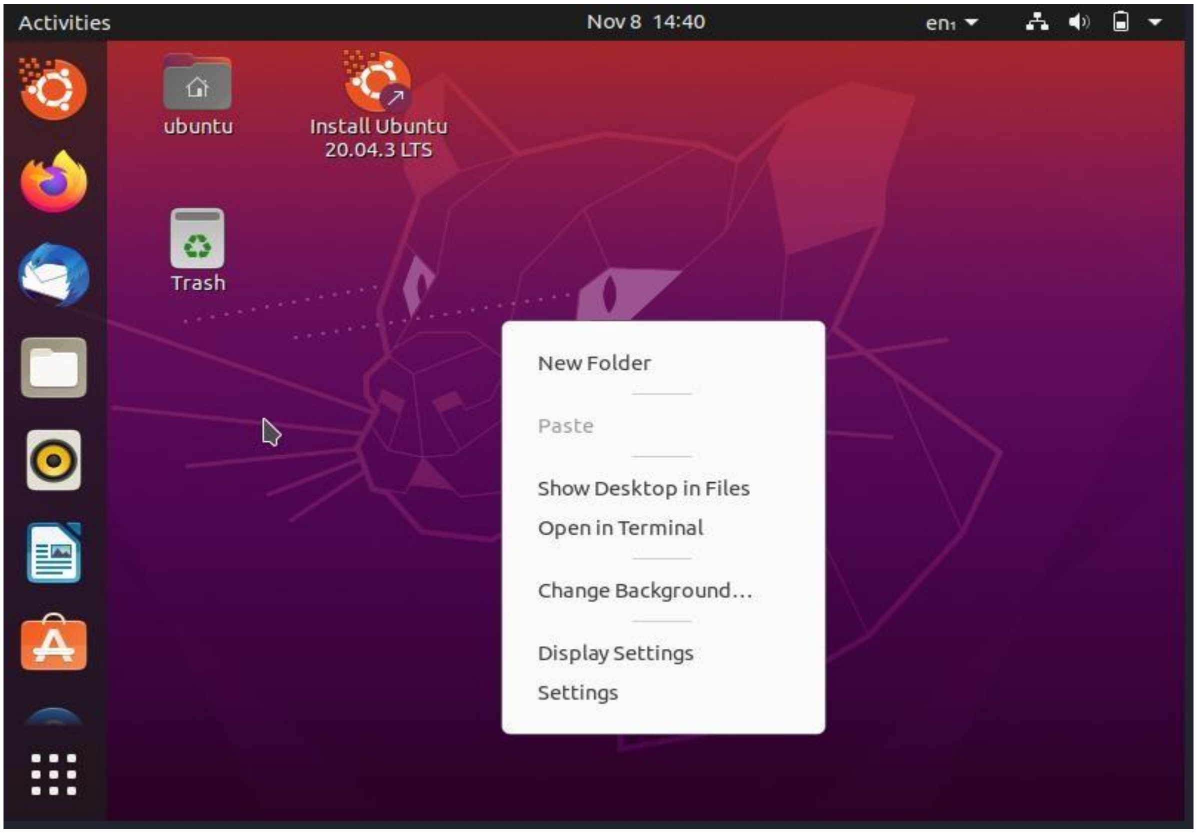
```
[root@localhost ~]# vi fifo_client.c

[root@localhost ~]# cc -o fifo_client fifo_client.c

[root@localhost ~]# ./fifo_clientTrying to Connect to Server...Connected...

Enter the filename to request from server: dvr.c

Waiting for Server to reply...

#include<stdio.h>struct node{unsigned int dist[20];unsigned int nexthop[20];}rt[10];
```

Part B- Network Simulation

1. **Using WIRESHARK observe the data transferred in client server communication using UDP**

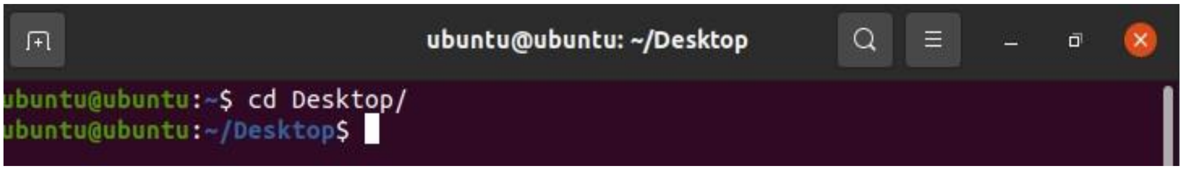
and identify the UDP datagram.

1. **UDP Program Execution Steps:**
   Steps :
   - Open Terminal using **CTRL+ALT+T** or right-click and select **Open in Terminal**.



- Once opened, type the following command in the terminal:

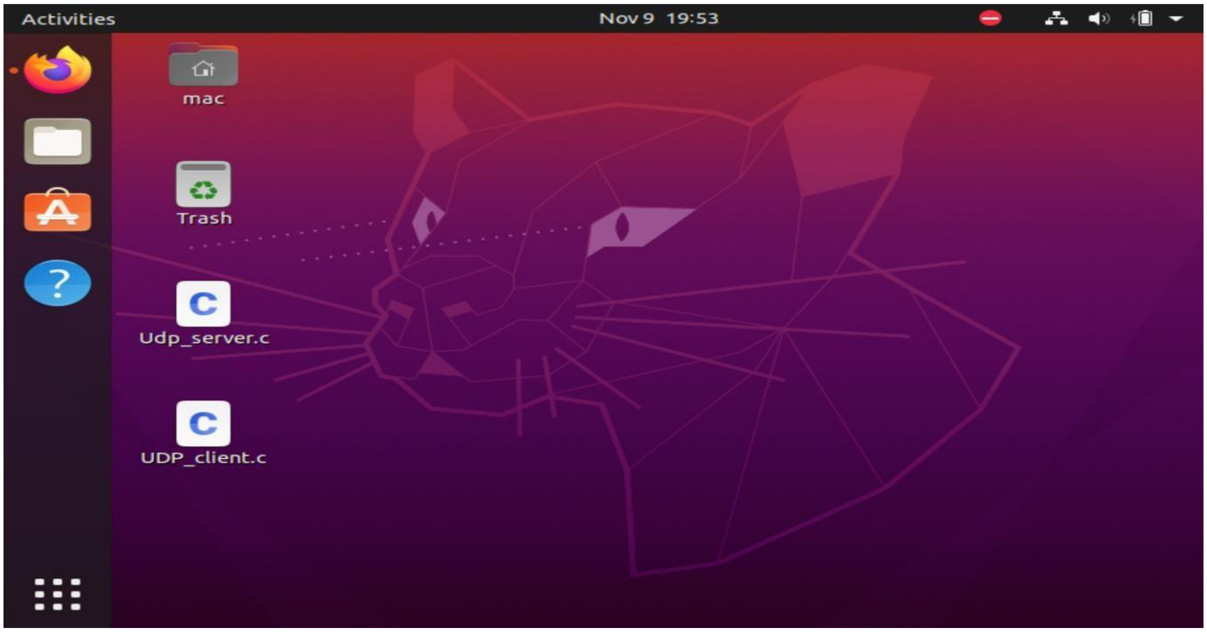
  cd Desktop/



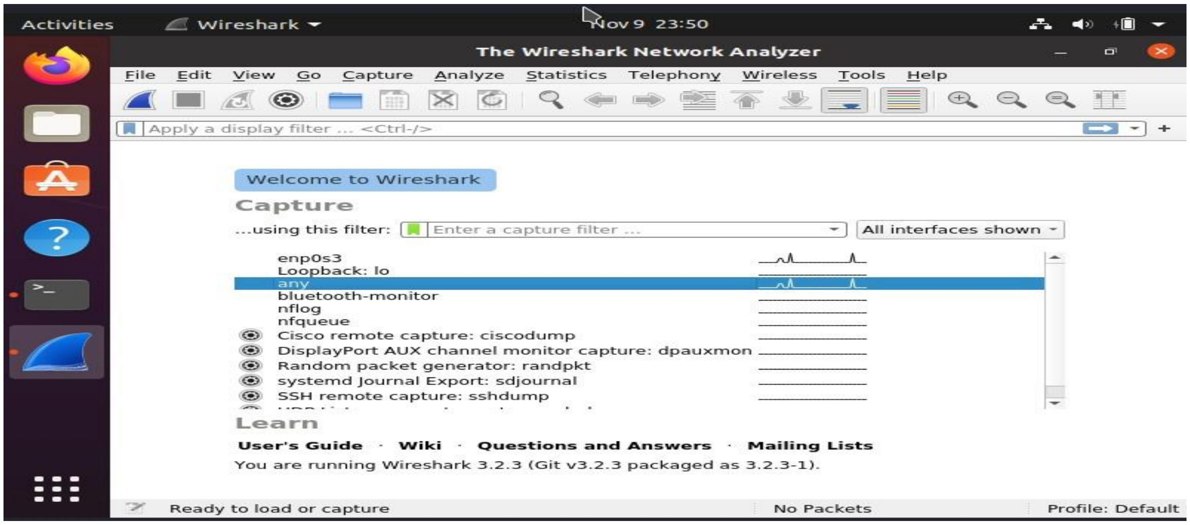
```
ubuntu@ubuntu:~$ cd Desktop/
ubuntu@ubuntu:~/Desktop$ 
```

This command will change your default terminal path to desktop. **(Note: Linux iscase-sensitive, so type the command as it is)**

[Before going through the next sequence of commands, make sure thatboth the UDP programs are located on the desktop]



- Before executing the program, open **Wireshark** and double-click on **any-interface** for packet capture process to start.


Activities
Wireshark
Nov 9 23:50
The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Welcome to Wireshark
Capture
…using this filter:
Enter a capture filter …
All interfaces shown
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

enp0s3
Loopback: lo
any
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump

You are running Wireshark 3.2.3 (Git v3.2.3 packaged as 3.2.3-1).

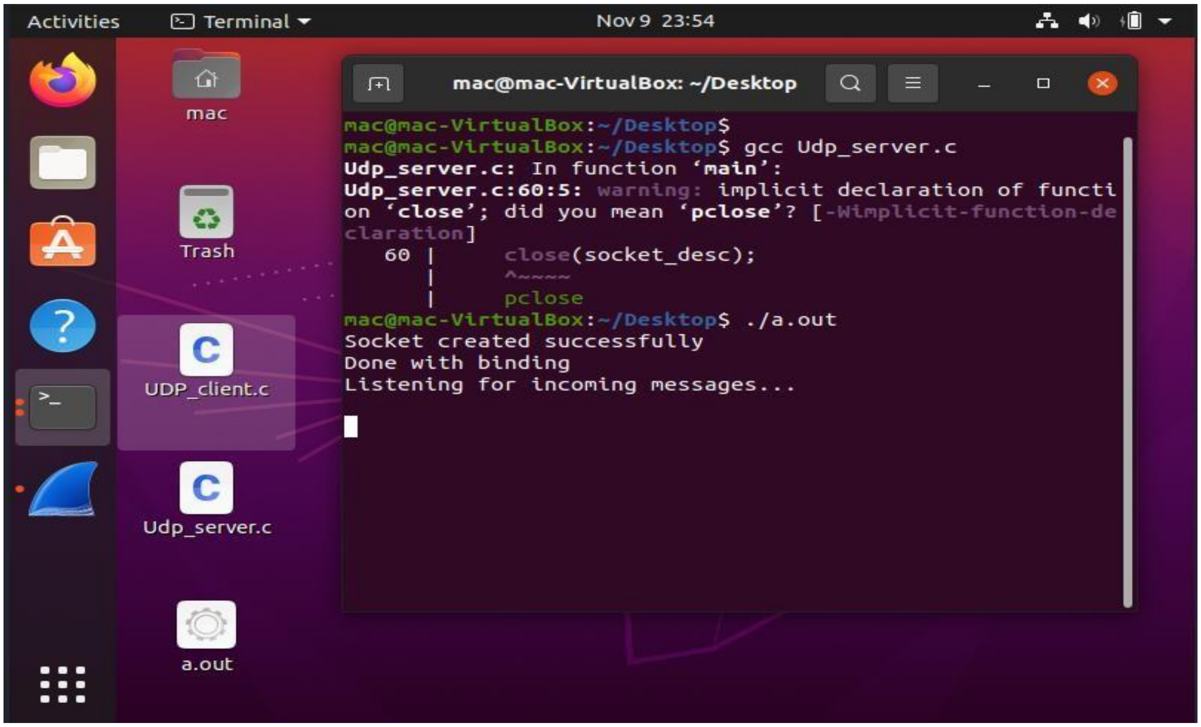
Ready to load or capture | No Packets | Profile: Default

- To compile the C program, type the following command in the same order asfollows in the terminal:

  gcc Udp_server.c

- To execute the program type, object code file name in the terminal as follows :

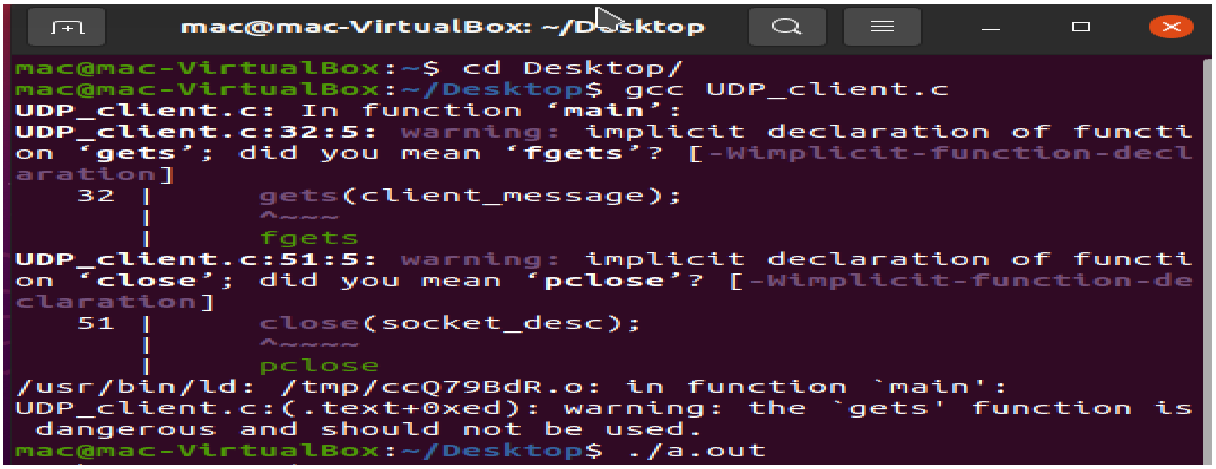
  ./a.out



```
mac@mac-VirtualBox:~/Desktop$
mac@mac-VirtualBox:~/Desktop$ gcc Udp_server.c
Udp_server.c: In function ‘main’:
Udp_server.c:60:5: warning: implicit declaration of functi
on ‘close’; did you mean ‘pclose’? [-Wimplicit-function-de
claration]
   60 |     close(socket_desc);
      |     ^~~~~
      |     pclose
mac@mac-VirtualBox:~/Desktop$ ./a.out
Socket created successfully
Done with binding
Listening for incoming messages...
```

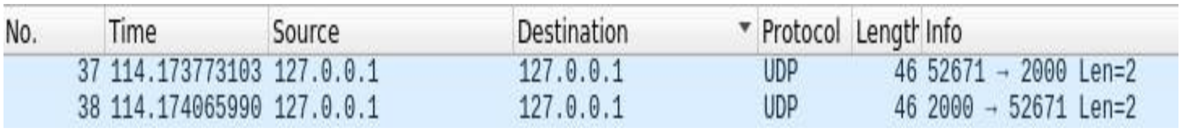
- Open another terminal and change its path to Desktop using Cd command and run the c code as shown below:

  cd Desktop/
  gcc UDP_client.c
  ./a.out



```
mac@mac-VirtualBox:~$ cd Desktop/
mac@mac-VirtualBox:~/Desktop$ gcc UDP_client.c
UDP_client.c: In function ‘main’:
UDP_client.c:32:5: warning: implicit declaration of functi
on ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-decl
aration]
   32 |     gets(client_message);
      |     ^~~~
      |     fgets
UDP_client.c:51:5: warning: implicit declaration of functi
on ‘close’; did you mean ‘pclose’? [-Wimplicit-function-de
claration]
   51 |     close(socket_desc);
      |     ^~~~~
      |     pclose
/usr/bin/ld: /tmp/ccQ79BdR.o: in function `main':
UDP_client.c:(.text+0xed): warning: the `gets' function is
 dangerous and should not be used.
mac@mac-VirtualBox:~/Desktop$ ./a.out
```

- Once the client program is executed, it will ask you to enter the appropriate message that will be sent to the server.
- After the execution of the program, go to Wireshark and stop the capturing process and check for the packet that has the source and the destination address as **127.0.0.1.**



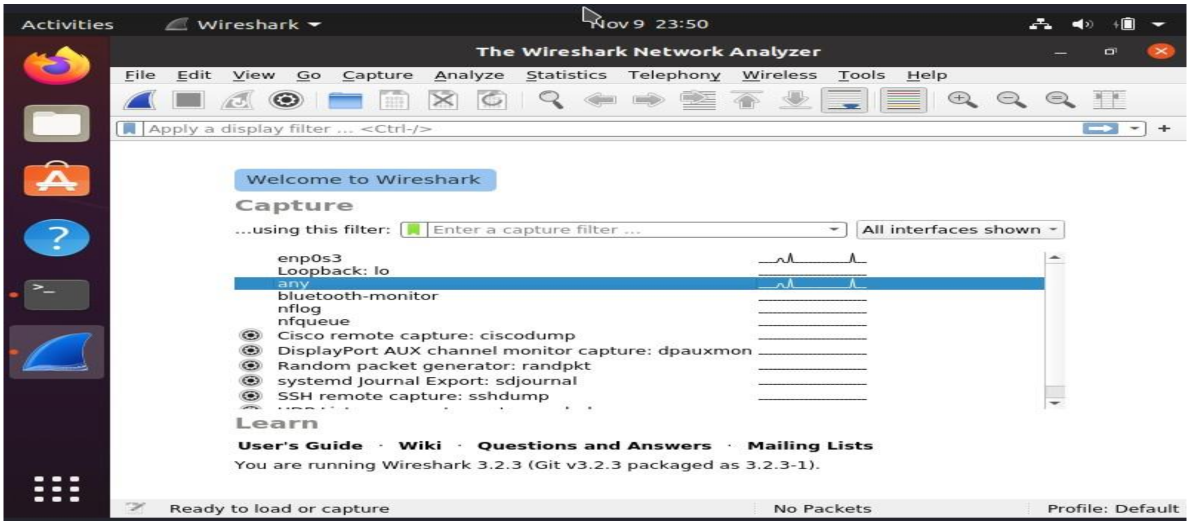
No.	Time	Source	Destination	Protocol	Length	Info
37	114.173773103	127.0.0.1	127.0.0.1	UDP	46	52671 → 2000 Len=2
38	114.174065990	127.0.0.1	127.0.0.1	UDP	46	2000 → 52671 Len=2

- Analyze the UDP Packets.

2. Capturing UDP Packets with browser :

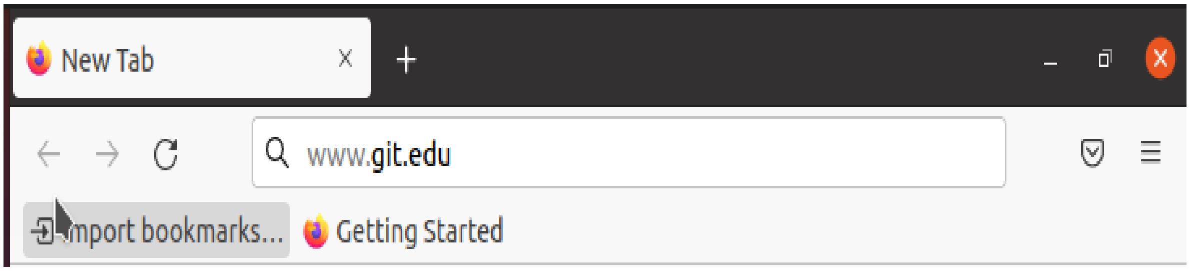
Steps:

- Open Wireshark and double-click on **any-interface** to start the packet captureprocess.

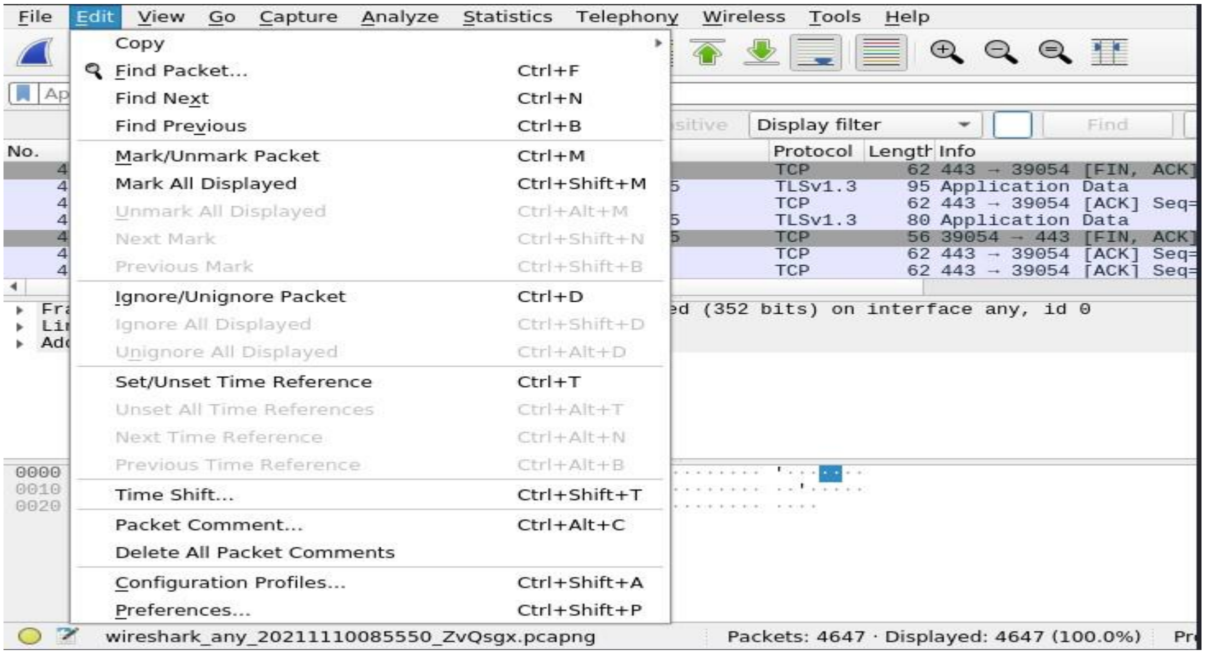


Activities
Wireshark
Nov 9 23:50
The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Welcome to Wireshark
Capture
…using this filter:
Enter a capture filter …
All interfaces shown
enp0s3
Loopback: lo
any
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists
You are running Wireshark 3.2.3 (Git v3.2.3 packaged as 3.2.3-1).
Ready to load or capture
No Packets
Profile: Default

- Open the browser and enter any website's fully qualified domain name in thebrowser address bar and hit enter.



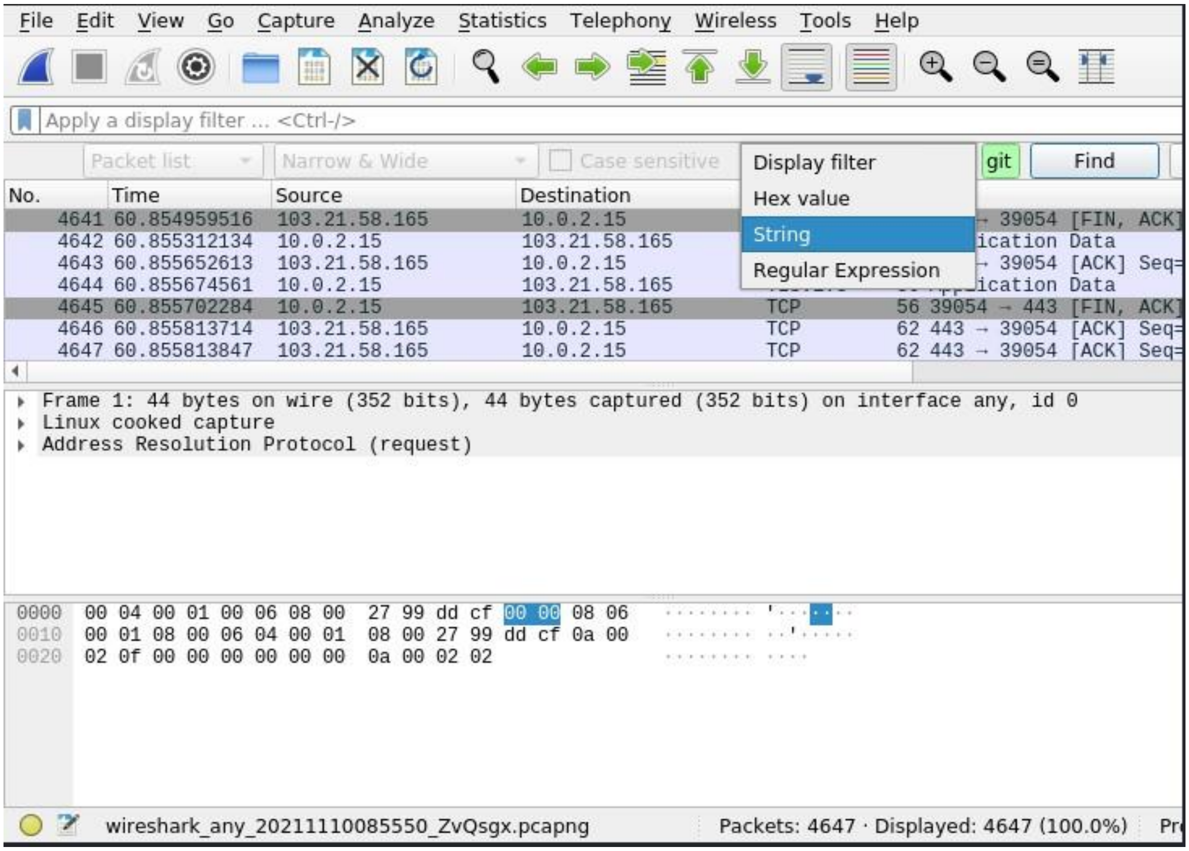
- After the site is fully loaded, stop the capturing process in Wireshark go to edit inthe menu bar and select find packet option or just press CTRL+F.


File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Tools
Help
Copy
Find Packet... Ctrl+F
Find Next Ctrl+N
Find Previous Ctrl+B
Mark/Unmark Packet Ctrl+M
Mark All Displayed Ctrl+Shift+M
Unmark All Displayed Ctrl+Alt+M
Next Mark Ctrl+Shift+N
Previous Mark Ctrl+Shift+B
Ignore/Unignore Packet Ctrl+D
Ignore All Displayed Ctrl+Shift+D
Unignore All Displayed Ctrl+Alt+D
Set/Unset Time Reference Ctrl+T
Unset All Time References Ctrl+Alt+T
Next Time Reference Ctrl+Alt+N
Previous Time Reference Ctrl+Alt+B
Time Shift... Ctrl+Shift+T
Packet Comment... Ctrl+Alt+C
Delete All Packet Comments
Configuration Profiles... Ctrl+Shift+A
Preferences... Ctrl+Shift+P
Display filter
Find
No.
Protocol
Length
Info

```
TCP      62 443 → 39054 [FIN, ACK]
TLSv1.3  95 Application Data
TCP      62 443 → 39054 [ACK] Seq=
TLSv1.3  80 Application Data
TCP      56 39054 → 443 [FIN, ACK]
TCP      62 443 → 39054 [ACK] Seq=
TCP      62 443 → 39054 [ACK] Seq=
ed (352 bits) on interface any, id 0
```

wireshark_any_20211110085550_ZvQsgx.pcapng   Packets: 4647 · Displayed: 4647 (100.0%)

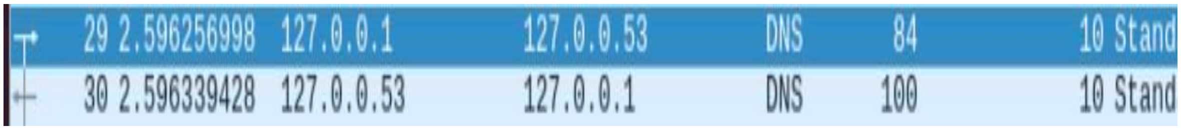
- In Find Packet menu bar, select the **String** option in the **display filter drop-down** menu and enter the name of the website in the next box and click on find.



```
No.  Time           Source          Destination
4641 60.854959516   103.21.58.165   10.0.2.15                        → 39054 [FIN, ACK]
4642 60.855312134   10.0.2.15       103.21.58.165                    ication Data
4643 60.855652613   103.21.58.165   10.0.2.15                        → 39054 [ACK] Seq=
4644 60.855674561   10.0.2.15       103.21.58.165                    ication Data
4645 60.855702284   10.0.2.15       103.21.58.165   TCP     56 39054 → 443 [FIN, ACK]
4646 60.855813714   103.21.58.165   10.0.2.15       TCP     62 443 → 39054 [ACK] Seq=
4647 60.855813847   103.21.58.165   10.0.2.15       TCP     62 443 → 39054 [ACK] Seq=

▸ Frame 1: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface any, id 0
▸ Linux cooked capture
▸ Address Resolution Protocol (request)

0000  00 04 00 01 00 06 08 00  27 99 dd cf 00 00 08 06   ........ '.......
0010  00 01 08 00 06 04 00 01  08 00 27 99 dd cf 0a 00   ........ ..'.....
0020  02 0f 00 00 00 00 00 00  0a 00 02 02               ........ ....
```

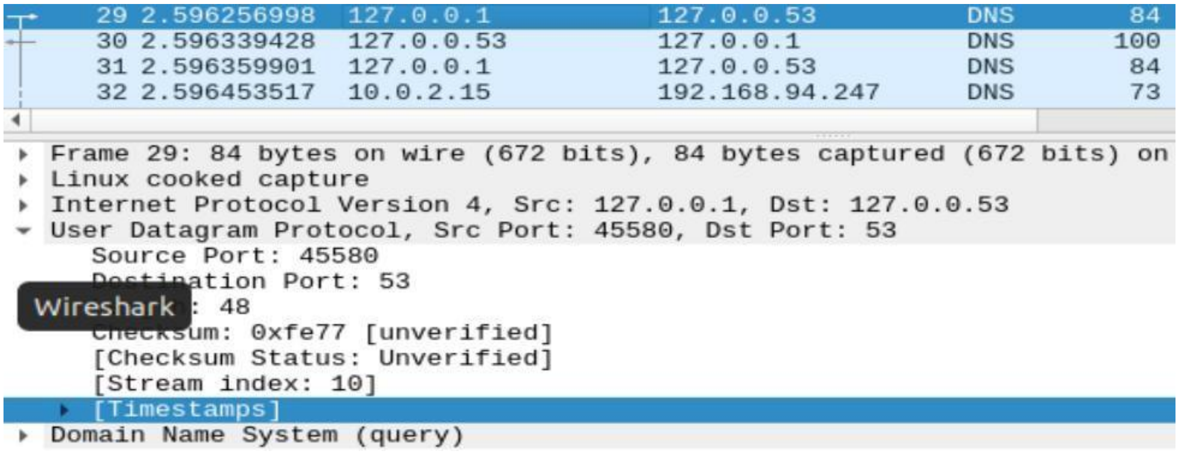
wireshark_any_20211110085550_ZvQsgx.pcapng   Packets: 4647 · Displayed: 4647 (100.0%)

- The arrow indicating towards the packet is the **request packet**, and the arrowcoming out from the packet is the **response packet**.



```
29 2.596256998  127.0.0.1     127.0.0.53    DNS   84    10 Stand
30 2.596339428  127.0.0.53    127.0.0.1     DNS   100   10 Stand
```

Click on any request or response DNS packet and exam



```
29 2.596256998  127.0.0.1     127.0.0.53       DNS    84
30 2.596339428  127.0.0.53    127.0.0.1        DNS   100
31 2.596359901  127.0.0.1     127.0.0.53       DNS    84
32 2.596453517  10.0.2.15     192.168.94.247   DNS    73

▸ Frame 29: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.53
▾ User Datagram Protocol, Src Port: 45580, Dst Port: 53
    Source Port: 45580
    Destination Port: 53
            : 48
    Checksum: 0xfe77 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 10]
  ▸ [Timestamps]
▸ Domain Name System (query)
```

2. Using WIRESHARK analyzethree way handshaking connection establishment, data transfer and connection termination in client server communication using TCP.

TCP Program Execution Steps:

Steps :

● Open Terminal using **CTRL+ALT+T** or right-click and select **Open in Terminal**. ● Once opened, type the following command in the terminal:

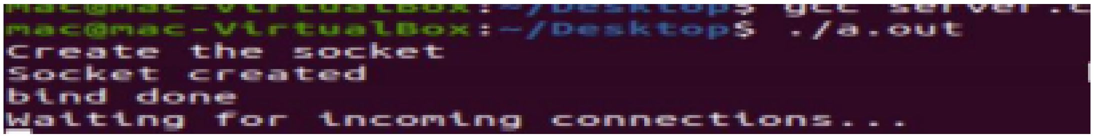
cd Desktop/

[Before going through the next sequence of commands, make sure that both the TCP programs are located on the desktop]

- Before executing the program, open **Wireshark** and double-click on **any-interface** for packet capture process to start.
- To compile the C program, type the following command in the same order as follows in the terminal:

gcc server.c

- To execute the program type, object code file name in the terminal as follows : **./a.out**



```
mac@mac-VirtualBox:~/Desktop$ gcc server.c
mac@mac-VirtualBox:~/Desktop$ ./a.out
Create the socket
Socket created
bind done
Waiting for incoming connections...
```

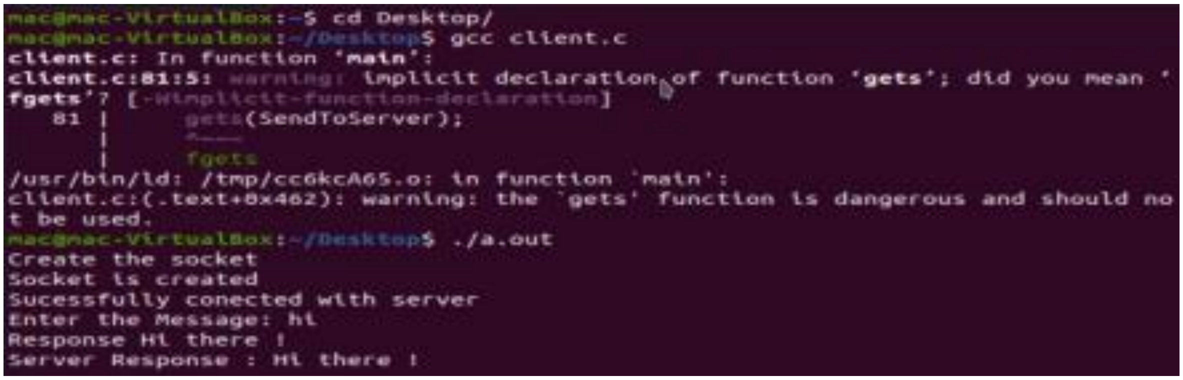
- Open another terminal and change its path to Desktop using Cd command and run the c code as shown below:

cd Desktop/

gcc client.c

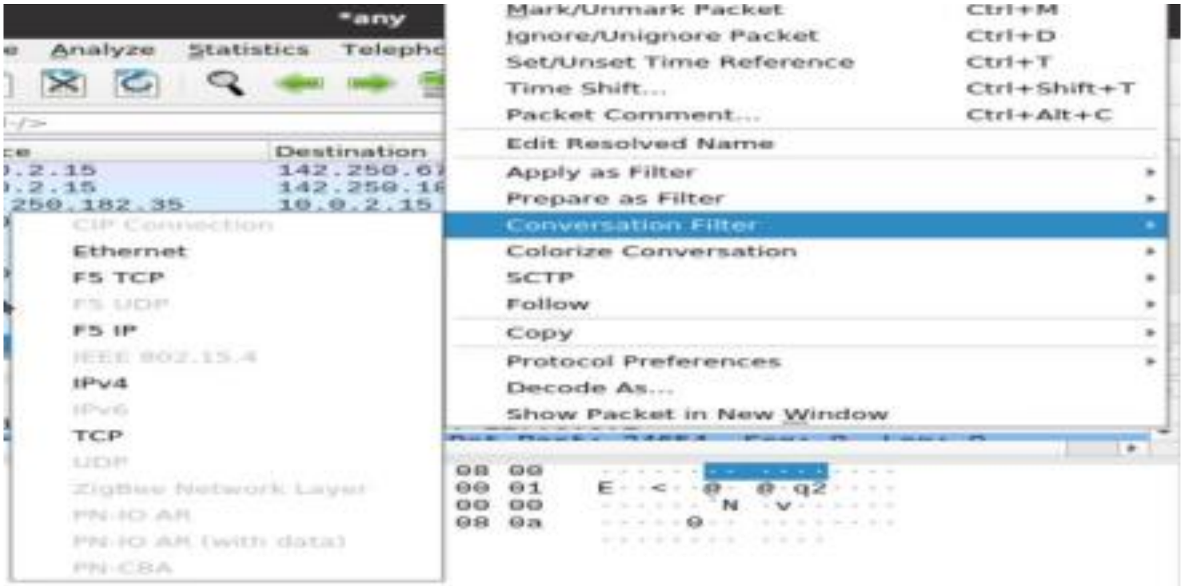
./a.out

- Once the client program is executed, it will ask you to enter a message type **hi without any space** and hit enter.
- The server will replay back with **hi there!** Message.



```
mac@mac-VirtualBox:~$ cd Desktop/
mac@mac-VirtualBox:~/Desktop$ gcc client.c
client.c: In function 'main':
client.c:81:5: warning: implicit declaration of function 'gets'; did you mean '
fgets'? [-Wimplicit-function-declaration]
   81 |     gets(SendToServer);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/cc6kcA65.o: in function `main':
client.c:(.text+0x462): warning: the `gets' function is dangerous and should no
t be used.
mac@mac-VirtualBox:~/Desktop$ ./a.out
Create the socket
Socket is created
Sucessfully conected with server
Enter the Message: hi
Response Hi there !
Server Response : Hi there !
```

- Once the program execution is completed, stop the capturing process in Wireshark.
- In Wireshark, check for the packets that has the source and destination address as **127.0.0.1** and select any one of these packets, right-click and hover on conversation filter and select TCP.



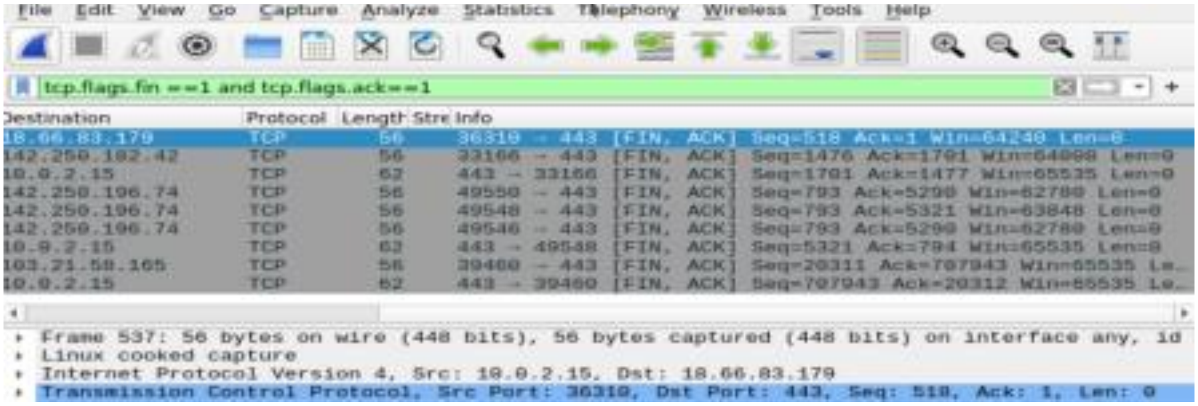
- Once done analyze the TCP Packets.

2. Capturing TCP Packets with browser : Steps:

- Open Wireshark and double-click on **any-interface** to start the packet capture process.
- Open the browser and enter any website's fully qualified domain name in the browser address bar and hit enter.
- After the site is fully loaded, stop the capturing process, in Wireshark.

- Type the following in, apply a filter column and hit-enter :

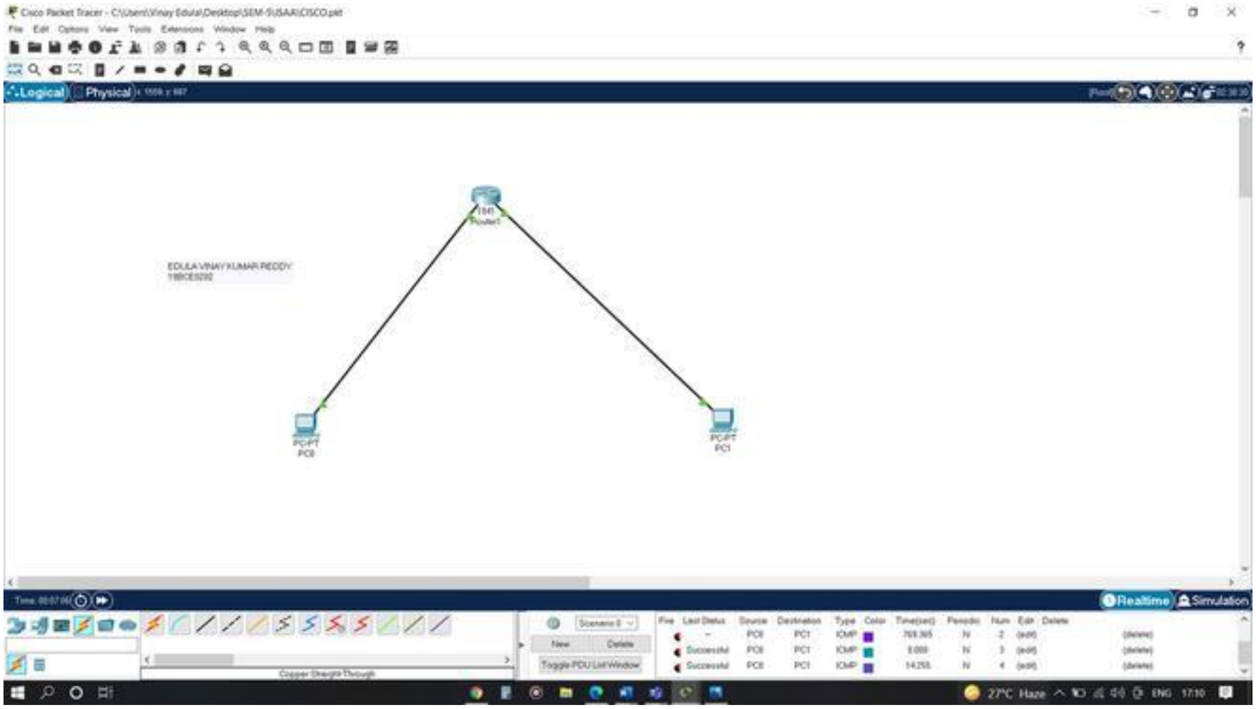
  tcp.flags.fin==1 and tcp.flags.ack ==1


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.flags.fin ==1 and tcp.flags.ack==1

Destination        Protocol Length Stre Info
18.66.83.179       TCP      56         36310 → 443 [FIN, ACK] Seq=518 Ack=1 Win=64240 Len=0
142.250.182.42     TCP      56         33166 → 443 [FIN, ACK] Seq=1476 Ack=1701 Win=64000 Len=0
10.0.2.15          TCP      62         443 → 33166 [FIN, ACK] Seq=1701 Ack=1477 Win=65535 Len=0
142.250.196.74     TCP      56         49550 → 443 [FIN, ACK] Seq=793 Ack=5290 Win=62780 Len=0
142.250.196.74     TCP      56         49548 → 443 [FIN, ACK] Seq=793 Ack=5321 Win=63840 Len=0
142.250.196.74     TCP      56         49546 → 443 [FIN, ACK] Seq=793 Ack=5290 Win=62780 Len=0
10.0.2.15          TCP      62         443 → 49548 [FIN, ACK] Seq=5321 Ack=794 Win=65535 Len=0
103.21.58.165      TCP      56         39460 → 443 [FIN, ACK] Seq=20311 Ack=707943 Win=65535 Le…
10.0.2.15          TCP      62         443 → 39460 [FIN, ACK] Seq=707943 Ack=20312 Win=65535 Le…

▸ Frame 537: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface any, id
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 18.66.83.179
▸ Transmission Control Protocol, Src Port: 36310, Dst Port: 443, Seq: 518, Ack: 1, Len: 0
```

- Select any one of these listed packets, right-click and hover on conversation filter and select TCP.
- Once done analyze the TCP Packets.

3. Demonstrate the router and pc configuration using packet tracer and verify it using realtime PDU transmission.



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	--	PC0	PC1	ICMP	■	769.365	N	2	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	■	0.000	N	3	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	■	14.255	N	4	(edit)	(delete)

Procedure:

Step-1(Configuring Router1):

1. Select the router and Open CLI.
2. Press ENTER to start configuring Router1.
3. Type enable to activate the privileged mode.
4. Type config t(configure terminal) to access the configuration menu.
5. Configure interfaces of Router1:

Router1 Command Line Interface:

```
Router>enable

Router#config t

Enter configuration commands, one per line.  End with CNTL/Z.

Router(config)#interface FastEthernet0/0

Router(config-if)#ip address 192.168.10.1 255.255.255.0

Router(config-if)#no shutdown

Router(config-if)#

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

```
Router(config-if)#interface FastEthernet0/1
Router(config-if)#ip address 192.168.20.1 255.255.255.0
Router(config-if)#no shutdown
```

Step-2(Configuring PCs):

1. Assign IP Addresses to every PC in the network.
2. Select the PC, Go to the desktop and select IP Configuration and assign an IP address, Default gateway, Subnet Mask
3. Assign the default gateway of PC0 as 192.168.10.1.
4. Assign the default gateway of PC1 as 192.168.20.1.

Step-3(Connecting PCs with Router):

1. Connect FastEthernet0 port of PC0 with FastEthernet0/0 port of Router1 using a copper straight-through cable.
2. Connect FastEthernet0 port of PC1 with FastEthernet0/1 port of Router1 using a copper straight-through cable.

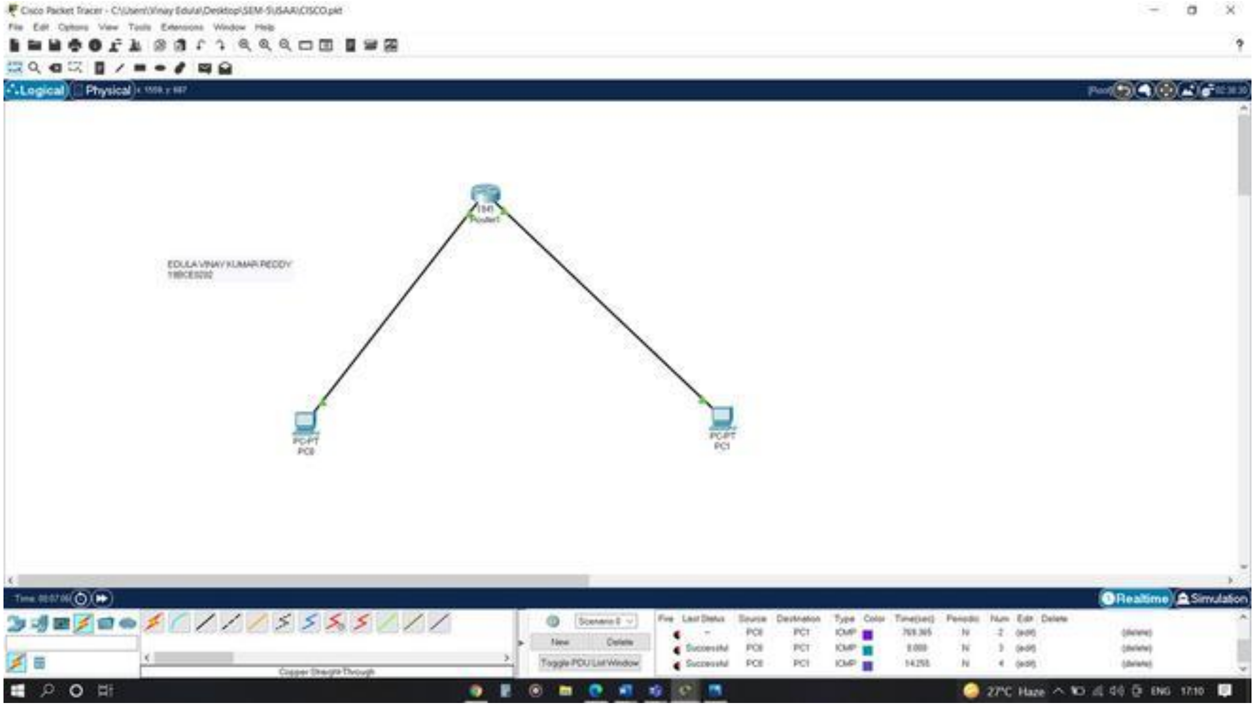
Router Configuration Table:

Device Name	IP address FastEthernet0/0	Subnet Mask	IP Address FastEthernet0/1	Subnet Mask
Router1	192.168.10.1	255.255.255.0	192.168.20.1	255.255.255.0

PC Configuration Table:

Device Name	IP address	Subnet Mask	Gateway
PC 0	192.168.10.2	255.255.255.0	192.168.10.1
PC 1	192.168.20.2	255.255.255.0	192.168.20.1

Designed Network topology:

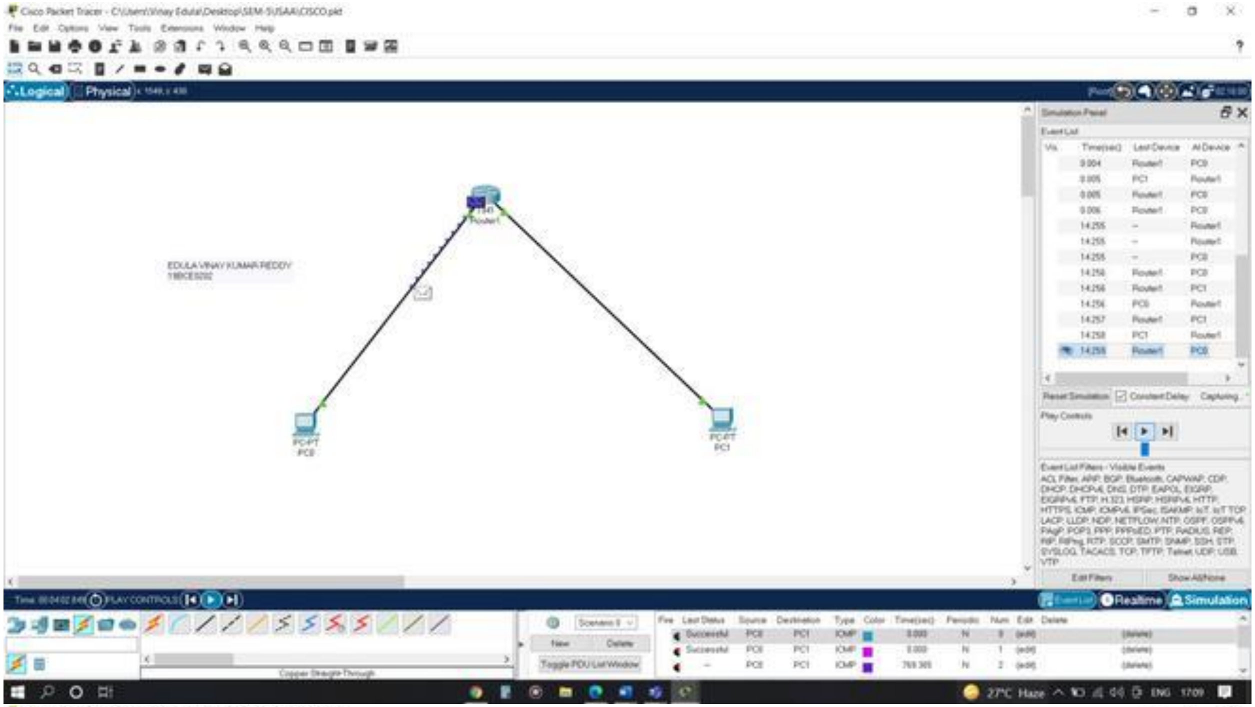


Cisco Packet Tracer - C:\Users\Vinay Edula\Desktop\SEM-5\SAA\CISCO.pkt

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	--	PC0	PC1	ICMP	🟪	769.365	N	2	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	🟦	8.000	N	3	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	🟪	14.255	N	4	(edit)	(delete)

Simulation of Designed Network Topology:

Sending a PDU From PC0 to PC1:

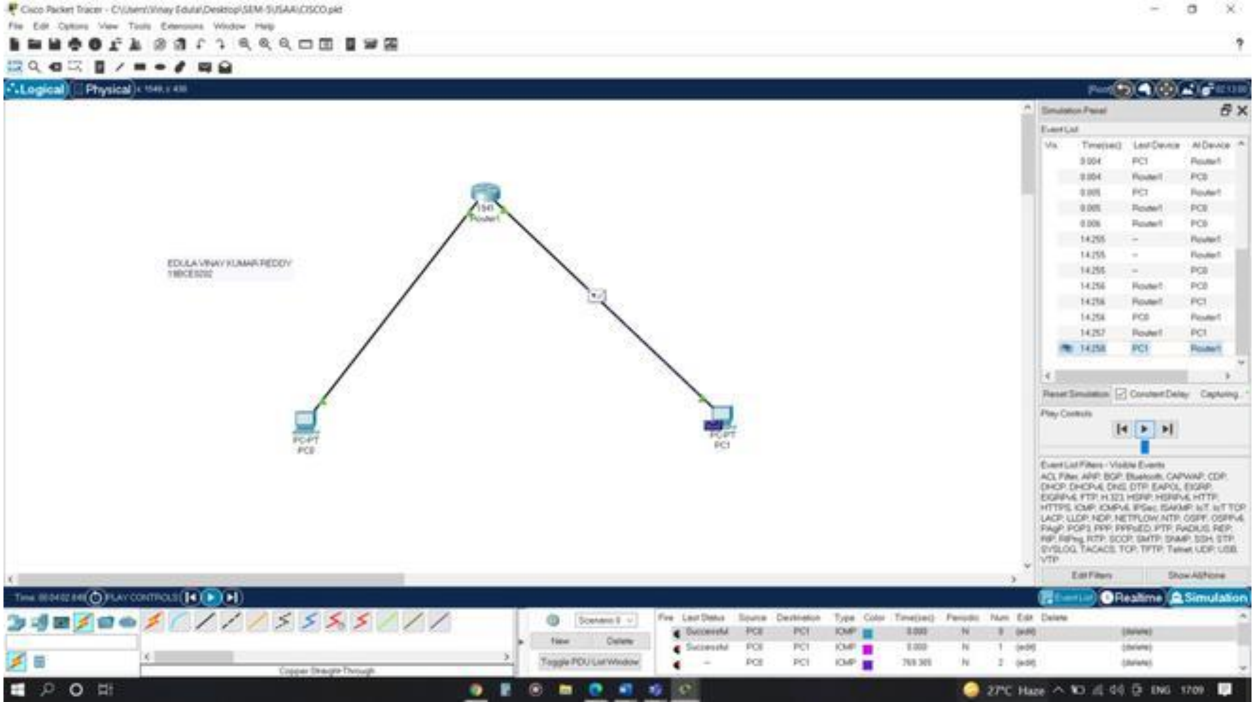


Vis.	Time(sec)	Last Device	At Device
	0.004	Router1	PC0
	0.005	PC1	Router1
	0.005	Router1	PC0
	0.006	Router1	PC0
	14.255	--	Router1
	14.255	--	Router1
	14.255	--	PC0
	14.256	Router1	PC0
	14.256	Router1	PC1
	14.256	PC0	Router1
	14.257	Router1	PC1
	14.258	PC1	Router1
✉	14.258	Router1	PC0

☑ Constant Delay

ACL Filter, ARP, BGP, Bluetooth, CAPWAP, CDP, DHCP, DHCPv6, DNS, DTP, EAPOL, EIGRP, EIGRPv6, FTP, H.323, HSRP, HSRPv6, HTTP, HTTPS, ICMP, ICMPv6, IPSec, ISAKMP, IoT, IoT TCP, LACP, LLDP, NDP, NETFLOW, NTP, OSPF, OSPFv6, PAgP, POP3, PPP, PPPoED, PTP, RADIUS, REP, RIP, RIPng, RTP, SCCP, SMTP, SNMP, SSH, STP, SYSLOG, TACACS, TCP, TFTP, Telnet, UDP, USB, VTP

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC0	PC1	ICMP	🟦	0.000	N	0	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	🟪	0.000	N	1	(edit)	(delete)
●	--	PC0	PC1	ICMP	🟪	769.365	N	2	(edit)	(delete)



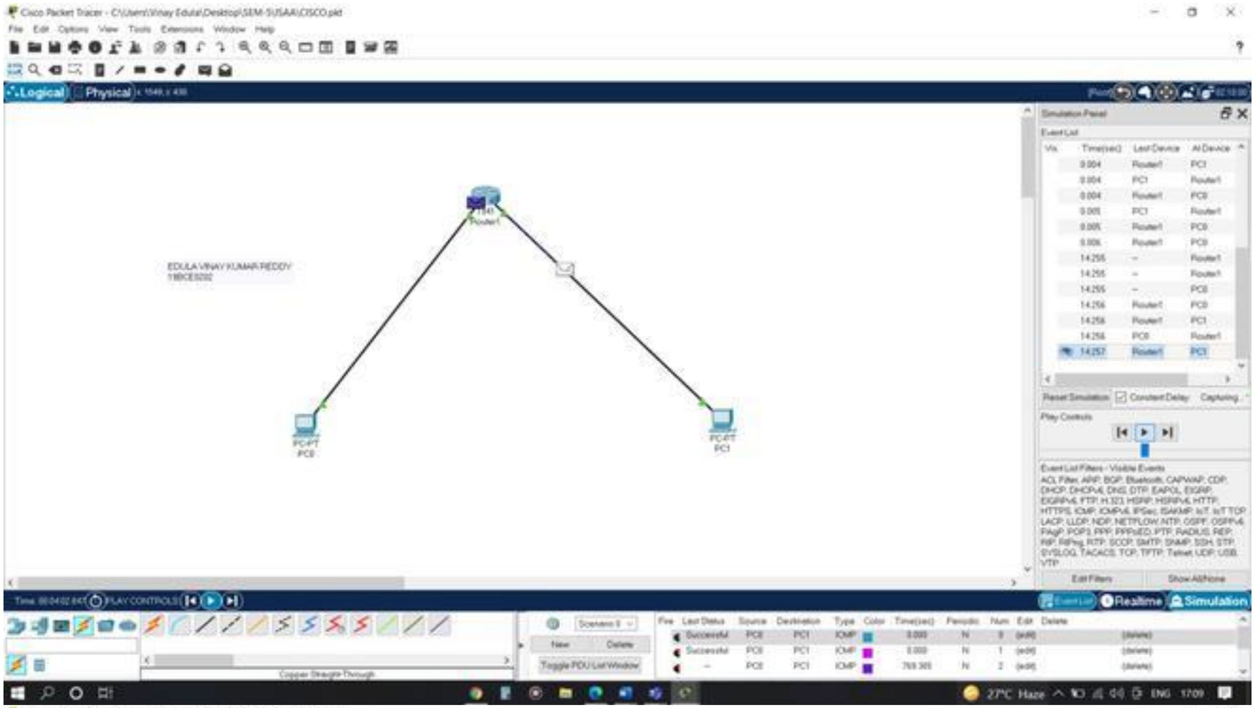
Vis.	Time(sec)	Last Device	At Device
	0.004	PC1	Router1
	0.004	Router1	PC0
	0.005	PC1	Router1
	0.005	Router1	PC0
	0.006	Router1	PC0
	14.255	--	Router1
	14.255	--	Router1
	14.255	--	PC0
	14.256	Router1	PC0
	14.256	Router1	PC1
	14.256	PC0	Router1
	14.257	Router1	PC1
✉	14.258	PC1	Router1

☑ Constant Delay

ACL Filter, ARP, BGP, Bluetooth, CAPWAP, CDP, DHCP, DHCPv6, DNS, DTP, EAPOL, EIGRP, EIGRPv6, FTP, H.323, HSRP, HSRPv6, HTTP, HTTPS, ICMP, ICMPv6, IPSec, ISAKMP, IoT, IoT TCP, LACP, LLDP, NDP, NETFLOW, NTP, OSPF, OSPFv6, PAgP, POP3, PPP, PPPoED, PTP, RADIUS, REP, RIP, RIPng, RTP, SCCP, SMTP, SNMP, SSH, STP, SYSLOG, TACACS, TCP, TFTP, Telnet, UDP, USB, VTP

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC0	PC1	ICMP	🟦	0.000	N	0	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	🟪	0.000	N	1	(edit)	(delete)
●	--	PC0	PC1	ICMP	🟪	769.365	N	2	(edit)	(delete)

Acknowledgment From PC1 to PC0:



Cisco Packet Tracer - C:\Users\Vinay Edula\Desktop\SEM-5\USAA\CISCO.pkt

Simulation Panel

Event List

Vis.	Time(sec)	Last Device	At Device
	0.004	Router1	PC1
	0.004	PC1	Router1
	0.004	Router1	PC0
	0.005	PC1	Router1
	0.005	Router1	PC0
	0.006	Router1	PC0
	14.255	--	Router1
	14.255	--	Router1
	14.255	--	PC0
	14.256	Router1	PC0
	14.256	Router1	PC1
	14.256	PC0	Router1
	14.257	Router1	PC1

Reset Simulation ☑ Constant Delay Capturing...

Play Controls

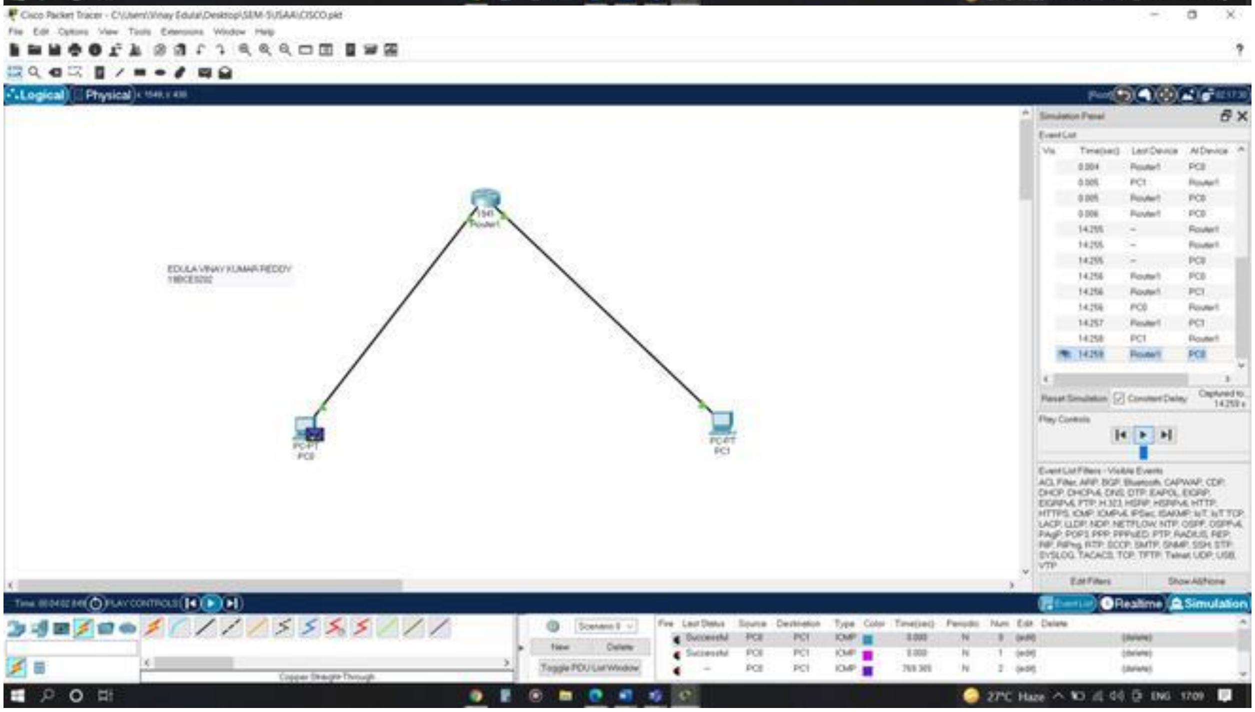
Event List Filters - Visible Events

ACL Filter, ARP, BGP, Bluetooth, CAPWAP, CDP, DHCP, DHCPv6, DNS, DTP, EAPOL, EIGRP, EIGRPv6, FTP, H.323, HSRP, HSRPv6, HTTP, HTTPS, ICMP, ICMPv6, IPSec, ISAKMP, IoT, IoT TCP, LACP, LLDP, NDP, NETFLOW, NTP, OSPF, OSPFv6, PAgP, POP3, PPP, PPPoED, PTP, RADIUS, REP, RIP, RIPng, RTP, SCCP, SMTP, SNMP, SSH, STP, SYSLOG, TACACS, TCP, TFTP, Telnet, UDP, USB, VTP

Edit Filters Show All/None

Scenario 0

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC0	PC1	ICMP	■	0.000	N	0	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	■	0.000	N	1	(edit)	(delete)
●	--	PC0	PC1	ICMP	■	769.365	N	2	(edit)	(delete)



Cisco Packet Tracer - C:\Users\Vinay Edula\Desktop\SEM-5\USAA\CISCO.pkt

Simulation Panel

Event List

Vis.	Time(sec)	Last Device	At Device
	0.004	Router1	PC0
	0.005	PC1	Router1
	0.005	Router1	PC0
	0.006	Router1	PC0
	14.255	--	Router1
	14.255	--	Router1
	14.255	--	PC0
	14.256	Router1	PC0
	14.256	Router1	PC1
	14.256	PC0	Router1
	14.257	Router1	PC1
	14.258	PC1	Router1
	14.259	Router1	PC0

Reset Simulation ☑ Constant Delay Captured to: 14.259 s

Play Controls

Event List Filters - Visible Events

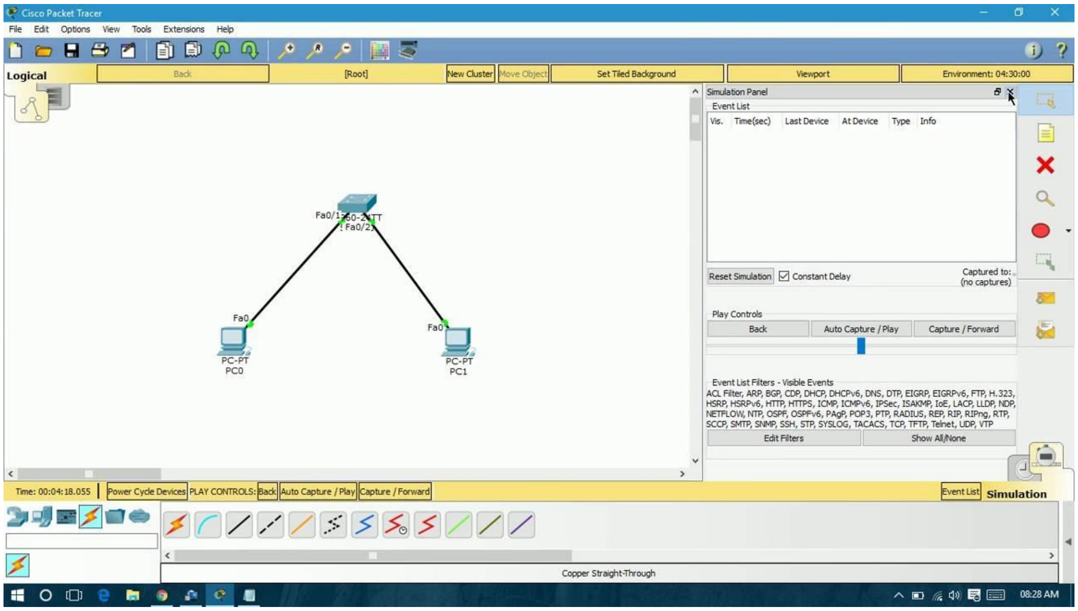
ACL Filter, ARP, BGP, Bluetooth, CAPWAP, CDP, DHCP, DHCPv6, DNS, DTP, EAPOL, EIGRP, EIGRPv6, FTP, H.323, HSRP, HSRPv6, HTTP, HTTPS, ICMP, ICMPv6, IPSec, ISAKMP, IoT, IoT TCP, LACP, LLDP, NDP, NETFLOW, NTP, OSPF, OSPFv6, PAgP, POP3, PPP, PPPoED, PTP, RADIUS, REP, RIP, RIPng, RTP, SCCP, SMTP, SNMP, SSH, STP, SYSLOG, TACACS, TCP, TFTP, Telnet, UDP, USB, VTP

Edit Filters Show All/None

Scenario 0

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC0	PC1	ICMP	■	0.000	N	0	(edit)	(delete)
●	Successful	PC0	PC1	ICMP	■	0.000	N	1	(edit)	(delete)
●	--	PC0	PC1	ICMP	■	769.365	N	2	(edit)	(delete)

3. Demonstrate the switch configuration using packet tracer and verify using realtime PDU transmission



☑ Constant Delay

Event List Filters - Visible Events: ACL Filter, ARP, BGP, CDP, DHCP, DHCPv6, DNS, DTP, EIGRP, EIGRPv6, FTP, H.323, HSRP, HSRPv6, HTTP, HTTPS, ICMP, ICMPv6, IPSec, ISAKMP, IoE, LACP, LLDP, NDP, NETFLOW, NTP, OSPF, OSPFv6, PAgP, POP3, PTP, RADIUS, REP, RIP, RIPng, RTP, SCCP, SMTP, SNMP, SSH, STP, SYSLOG, TACACS, TCP, TFTP, Telnet, UDP, VTP

Cisco Switch IOS Commands Enter global configuration mode.

S1# configure terminal Enter interface configuration mode.

S1(config)# interface vlan 99 Configure the management interface IP address.

S1(config-if)# ip address 172.17.99.11 255.255.255.0

Enable the management interface.

S1(config-if)# no shutdown Return to the privileged EXEC mode.

S1(config-if)# end Save the running configuration file to the startup configuration file.

S1# copy running-config startup-config

Examine the startup configuration file. To view the contents of the startup configuration file, issue the show startup-config command in privileged EXEC mode.

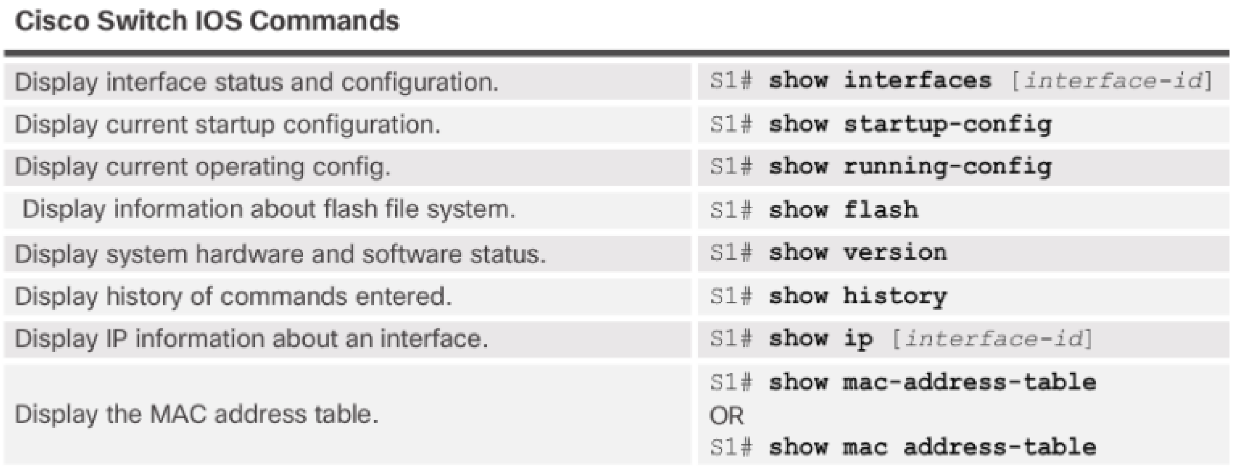
Switch#show startup-config startup-config is not present

Let’s make one configuration change to the switch and then save it. Type the following commands:

```
Switch#configure terminal Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)#hostname S1 S1(config)#exit S1#
```

To save the contents of the running configuration file to non-volatile RAM (NVRAM), issue the the command copy running-config startup-config.

```
Switch#copy running-config startup-config Destination filename [startup-config]? (enter)
Building configuration... [OK]
```



Cisco Switch IOS Commands

Display interface status and configuration.	S1# **show interfaces** [*interface-id*]
Display current startup configuration.	S1# **show startup-config**
Display current operating config.	S1# **show running-config**
Display information about flash file system.	S1# **show flash**
Display system hardware and software status.	S1# **show version**
Display history of commands entered.	S1# **show history**
Display IP information about an interface.	S1# **show ip** [*interface-id*]
Display the MAC address table.	S1# **show mac-address-table** OR S1# **show mac address-table**

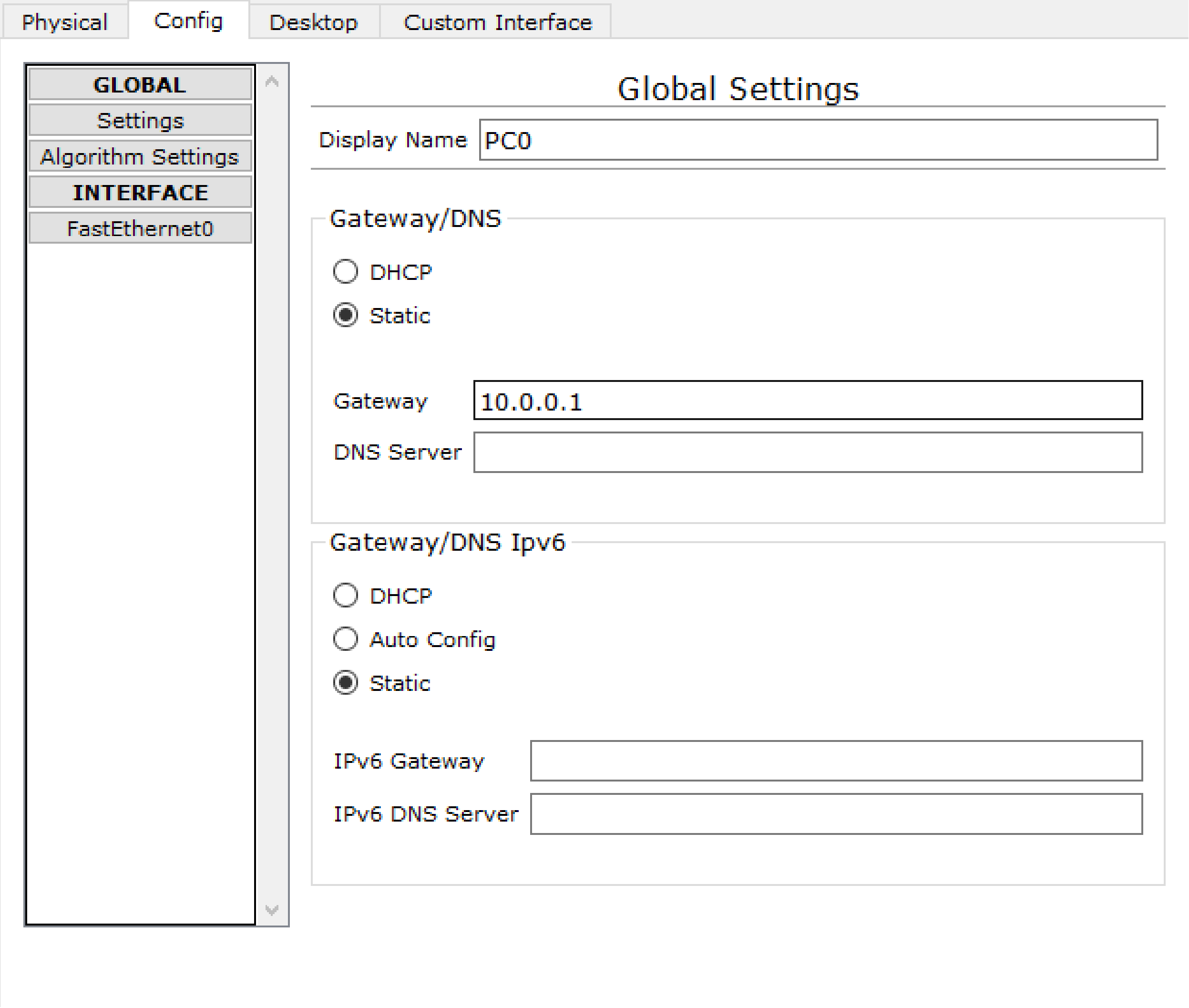
Configure an IP Address to PC

There are following steps involved to configure an IP Address to PC:

Step1: Open the Cisco Packet Tracer.

Step2: Drag and drop PC from the bottom of the interface into the middle of the working area.

Step3: Click on PC ->Config Gateway like 10.0.0.1



Global Settings

Display Name: PC0

Gateway/DNS

○ DHCP
● Static

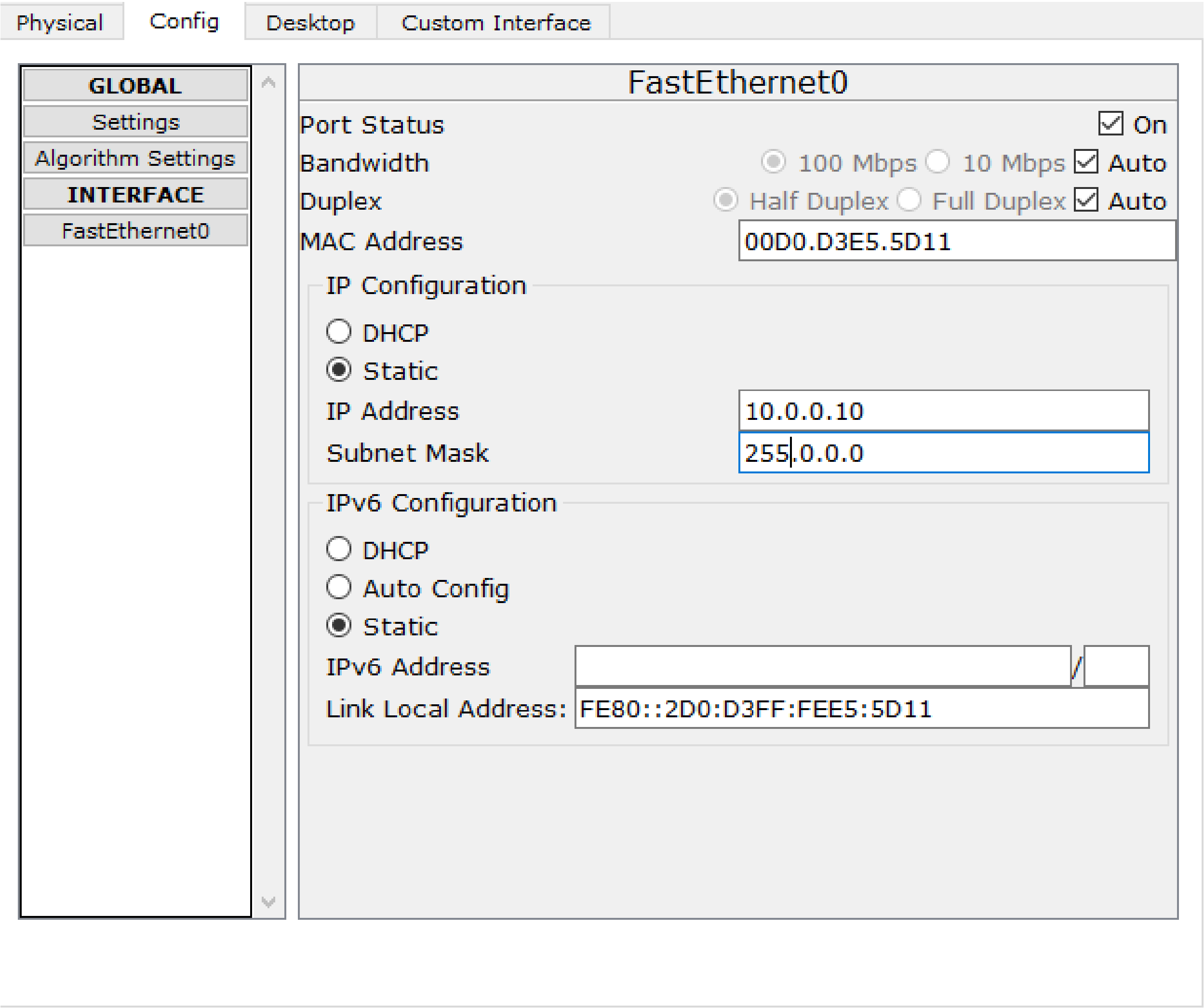
Gateway: 10.0.0.1
DNS Server:

Gateway/DNS Ipv6

○ DHCP
○ Auto Config
● Static

IPv6 Gateway:
IPv6 DNS Server:

Click on FastEthernet to assign an IP address and subnetmask to the PC, and close PC window. In our case IP is 10.0.0.10 and subnet mask is 255.0.0.0.0,



FastEthernet0

Port Status ☑ On

Bandwidth ● 100 Mbps ○ 10 Mbps ☑ Auto

Duplex ● Half Duplex ○ Full Duplex ☑ Auto

MAC Address 00D0.D3E5.5D11

IP Configuration

○ DHCP

● Static

IP Address 10.0.0.10

Subnet Mask 255.0.0.0

IPv6 Configuration

○ DHCP

○ Auto Config

● Static

IPv6 Address /

Link Local Address: FE80::2D0:D3FF:FEE5:5D11

Basics of NS3 simulation

Introduction In this lab, we will be using the Network Simulator, NS3, available from www.nsnam.org. NS3 is a powerful program, however we will only be looking at some basic features. NS3 simulations are built in C++.

Installation:

Following are the basic steps which must be followed for installing NS3
1. Install prerequisite packages
2. Download ns3 codes
3. Build ns3
4. Validate ns3

Prerequisite packages for Linux are as follows:

1.Minimal requirements for Python: gcc g++ python
2. Debugging and GNU Scientific Library (GSL) support: gdbpython-dev
3. valgrind gsl-bin libgsl0-dev libgsl0ldbl Network Simulation Cradle (nsc): flex bison
Reading pcap packet traces: tcpdump
4.Database support for statistics framework: sqlite sqlite3
5. Xml-based version of the config store: libxml2
6.A GTK-based configuration system: libgtk2.0-0
7. Experimental with virtual machines and ns-3: vtun lxc

Detail steps are as follows:

1.sudo apt-get update / dnf update
2.sudo apt-get upgrade / dnf upgrade
3 Once ubuntu/fedora is installed run following command opening the terminal(ctrl+alt+T) window.
4.To install prerequisites dependancy packages- Type the following command in terminal window.

```
sudo apt-get/ dnf install gcc g++ python python-dev mercurial bzr gdb valgrind gsl-bin libgsl0-dev libgsl0ldbl flex bison tcpdump sqlite sqlite3 libsqlite3-dev libxml2 libxml2-dev libgtk2.0-0 libgtk2.0-dev uncrustify doxygen graphviz imagemagick texlive texlive-latex-extra texlive-generic-extra texlive-generic-recommended texinfo dia texlive texlive-latex-extra texlive-extra-utils texlive-generic-recommended texi2html python-pygraphviz python-kiwi python-pygoocanvas libgoocanvas-dev python-pygccxml
```

5.After downloading NS3 on the drive, extract all the files in the NS3 folder, which you have created.
6.Then you can find build.py along with other files in NS3 folder.
Then to build the examples in ns-3 run :

```
./build.py --enable-examples –enable-tests
```

If the build is successful then it will give output
"Build finished successfully".
7. Now run the following command on the terminal window,to configure with waf(build tool)

```
./waf -d debug --enable-examples --enable-tests configure
```

To build with waf(optional)

```
./waf
```

8.To test everything allright run the following command on the terminal window,

```
./test.py
```

If the tests are ok the installation is done
9.Now after installing ns3 and testing it run some programs first to be ns3 user:

make sure you are in directory where waf script is available then run

Program in NS3 to connect two nodes

Node

Because in any network simulation, we will need nodes. So ns-3 comes with NodeContainer that you can use to manage all the nodes (Add, Create, Iterate, etc.).

```
// Create two nodes to hold.
NodeContainer nodes;
nodes.Create (2);
```

Channel and NetDevice

In the real world, they correspond to network cables (or wireless media) and peripheral cards (NIC). Typically these two things are intimately tied together. In the first example, we are usingPointToPointHelper that wraps the Channel and NetDevice.

```
// Channel: PointToPoint, a direct link with `DataRate` and `Delay` specified.
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

Then we need to install the devices. The internal of Install is actually more complicated, but for now, let’s just skip the magic behind the scene.

```
// NetDevice: installed onto the channel
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
```

Protocols

Internet and IPv4. Since Internet is the current largest network to study, ns-3 has a particular focus on it. The InternetStackHelper will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.

```
// Protocol Stack: Internet Stack
InternetStackHelper stack;
stack.Install (nodes);
```

To assign IP addresses, use a helper and set the base. The low level ns-3 system actually remembers all of the IP addresses allocated and will generate a fatal error if you accidentally cause the same address to be generated twice.

```
// Since IP Address assignment is so common, the helper does the dirty work!
// You only need to set the base.
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
// Assign the address to devices we created above
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

Applications

Every application needs to have Start and Stop function so that the simulator knows how to schedule it. Other functions are application-specific. We will use UdpEchoServer and UdpEchoClientfor now.

```
// Application layer: UDP Echo Server and Client
// 1, Server:
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
// 2, Client:
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

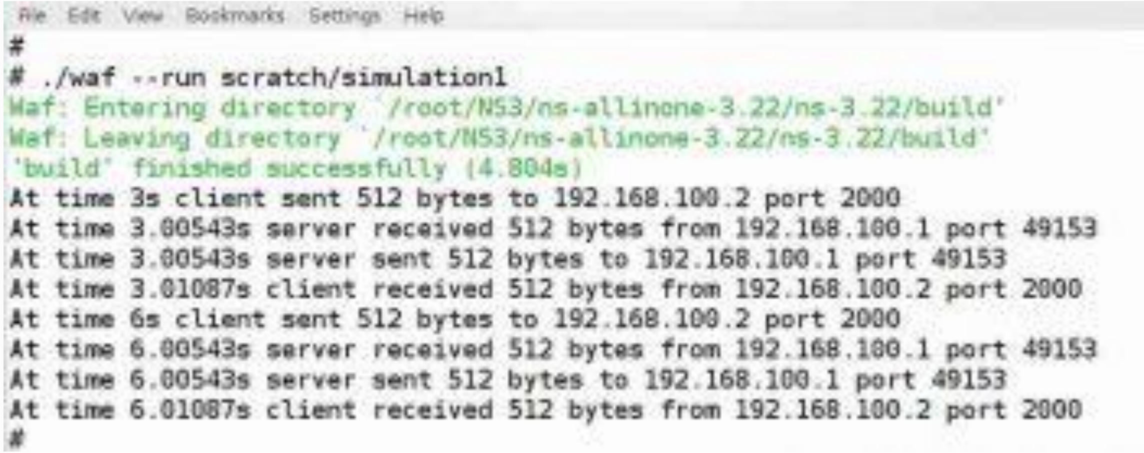
Simulation

```
// Start Simulation
Simulator::Run ();
Simulator::Destroy ();

return 0;
```

5. Simulate a Full duplex connection in an wired network using NS3.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"

// Default Network Topology
//
// 10.1.1.0
// n0 -------------  n1
// point-to-point
//

using namespace ns3;

NS_LOG_COMPONENT_DEFINE  ("FirstScriptExample");

int
main (int argc, char *argv[])
{
 CommandLine cmd (__FILE__);
 cmd.Parse (argc, argv);

 Time::SetResolution (Time::NS);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

NodeContainer nodes;
nodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate",StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);

InternetStackHelper stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

Output:



```
#
# ./waf --run scratch/simulation1
Waf: Entering directory `/root/NS3/ns-allinone-3.22/ns-3.22/build'
Waf: Leaving directory `/root/NS3/ns-allinone-3.22/ns-3.22/build'
'build' finished successfully (4.804s)
At time 3s client sent 512 bytes to 192.168.100.2 port 2000
At time 3.00543s server received 512 bytes from 192.168.100.1 port 49153
At time 3.00543s server sent 512 bytes to 192.168.100.1 port 49153
At time 3.01087s client received 512 bytes from 192.168.100.2 port 2000
At time 6s client sent 512 bytes to 192.168.100.2 port 2000
At time 6.00543s server received 512 bytes from 192.168.100.1 port 49153
At time 6.00543s server sent 512 bytes to 192.168.100.1 port 49153
At time 6.01087s client received 512 bytes from 192.168.100.2 port 2000
#
```

5. Simulate a simple Wireless UDP application using NS3.

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"

// Default Network Topology
//
// Wifi 10.1.3.0
// AP
// * * * *
// | | | | 10.1.1.0
// n5 n6 n7 n0 --------------  n1 n2 n3 n4
// point-to-point | | | |
// ================
// LAN 10.1.2.0

using namespace ns3;

NS_LOG_COMPONENT_DEFINE  ("ThirdScriptExample");

int
main (int argc, char *argv[])
{
 bool verbose = true;
 uint32_t nCsma  = 3;
 uint32_t nWifi = 3;
 bool tracing = false;

 CommandLine cmd (__FILE__);
 cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
 cmd.AddValue ("nWifi",  "Number of wifi STA devices", nWifi);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

cmd.Parse (argc,argv);

// The underlying restriction of 18 is due to the grid position
// allocator's configuration; the grid layout will exceed the
// bounding box if more than 18 nodes are provided.
if (nWifi > 18)
{
std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box" << std::endl;
return 1;
}

if (verbose)
{
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}

NodeContainer p2pNodes;
p2pNodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");

WifiHelper wifi;

NetDeviceContainer staDevices;
mac.SetType ("ns3::StaWifiMac",
"Ssid", SsidValue (ssid),
"ActiveProbing", BooleanValue (false));
staDevices = wifi.Install (phy, mac, wifiStaNodes);

NetDeviceContainer apDevices;
mac.SetType ("ns3::ApWifiMac",
"Ssid", SsidValue (ssid));
apDevices = wifi.Install (phy, mac, wifiApNode);

MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
"MinX", DoubleValue (0.0),
"MinY", DoubleValue (0.0),
"DeltaX", DoubleValue (5.0),
"DeltaY", DoubleValue (10.0),
"GridWidth", UintegerValue (3),
"LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
"Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);

InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

```

```
ApplicationContainer clientApps =
echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing)
{
phy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
pointToPoint.EnablePcapAll ("third");
phy.EnablePcap ("third", apDevices.Get (0));
csma.EnablePcap ("third", csmaDevices.Get (0), true);
}
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

Output:

```
cp examples/third.cc scratch/mythird.cc
  ./waf
  ./waf --run scratch/mythird
```

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
  Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
  'build' finished successfully (0.407s)
  Sent 1024 bytes to 10.1.2.4
  Received 1024 bytes from 10.1.3.3
  Received 1024 bytes from 10.1.2.4
third-0-0.pcap  third-0-1.pcap  third-1-0.pcap  third-1-1.pcap
```

You should see some wifi-looking contents you haven’t seen here before:

```
  reading from file third-0-1.pcap, link-type IEEE802_11 (802.11)
  0.000025 Beacon () [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
  0.000263 Assoc Request () [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
  0.000279 Acknowledgment RA:00:00:00:00:00:07
  0.000357 Assoc Response AID(0) :: Succesful
  0.000501 Acknowledgment RA:00:00:00:00:00:0a
  0.000748 Assoc Request () [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
  0.000764 Acknowledgment RA:00:00:00:00:00:08
  0.000842 Assoc Response AID(0) :: Succesful
  0.000986 Acknowledgment RA:00:00:00:00:00:0a
  0.001242 Assoc Request () [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
  0.001258 Acknowledgment RA:00:00:00:00:00:09
  0.001336 Assoc Response AID(0) :: Succesful
  0.001480 Acknowledgment RA:00:00:00:00:00:0a
  2.000112 arp who-has 10.1.3.4 (ff:ff:ff:ff:ff:ff) tell 10.1.3.3
  2.000128 Acknowledgment RA:00:00:00:00:00:09
  2.000206 arp who-has 10.1.3.4 (ff:ff:ff:ff:ff:ff) tell 10.1.3.3
  2.000487 arp reply 10.1.3.4 is-at 00:00:00:00:00:0a
  2.000659 Acknowledgment RA:00:00:00:00:00:0a
```

```
2.002169 IP 10.1.3.3.49153 > 10.1.2.4.9: UDP, length 1024
2.002185 Acknowledgment RA:00:00:00:00:00:09
2.009771 arp who-has 10.1.3.3 (ff:ff:ff:ff:ff:ff) tell 10.1.3.4
2.010029 arp reply 10.1.3.3 is-at 00:00:00:00:00:09
2.010045 Acknowledgment RA:00:00:00:00:00:09
2.010231 IP 10.1.2.4.9 > 10.1.3.3.49153: UDP, length 1024
2.011767 Acknowledgment RA:00:00:00:00:00:0a
2.500000 Beacon () [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
5.000000 Beacon () [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
7.500000 Beacon () [6.0* 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit] IBSS
```