SREE CHAITANYA COLLEGE OF ENGINEERING
LMD Colony, Thimmapur, Karimnagar

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN

LAB MANUAL

Academic Year 2023-24BTECH III –I

R18-REGULATION

COMPUTER NETWORKS LAB

Prepared by

T.NARENDER

Verified by

HOD /CSD

Course Objectives

1. To understand the working principle of various communication protocols.
2. To understand the network simulator environment and visualize a network topology and observe its performance
3. To analyze the traffic flow and the contents of protocol frames

Course Outcomes

1. Implement data link layer farming methods
2. Analyze error detection and error correction codes.
3. Implement and analyze routing and congestion issues in network design.
4. Implement Encoding and Decoding techniques used in presentation layer
5. To be able to work with different network tools

CS506PC: COMPUTER NETWORKS LAB SYLLABUS

LIST OF EXPERIMENTS

1. Implement the data link layer framing methods such as character, character-stuffing and bit Stuffing.
2. Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP
3. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.
4. Implement Dijsktra's algorithm to compute the shortest path through a network
5. Take an example subnet of hosts and obtain a broadcast tree for the subnet.
6. Implement distance vector routing algorithm for obtaining routing tables at each node.
7. Implement data encryption and data decryption
8. Write a program for congestion control using Leaky bucket algorithm.
9. Write a program for frame sorting technique used in buffers.
10. **Wire shark**

i. Packet Capture Using Wire shark

ii. Starting Wire shark

iii. Viewing Captured Traffic

iv. Analysis and Statistics & Filters.

11. How to run Nmap scan
12. Operating System Detection using Nmap
13. Do the following using NS2 Simulator

i-Introduction

ii. Simulate to Find the Number of Packets Dropped

iii. Simulate to Find the Number of Packets Dropped by TCP/UDP

iv. Simulate to Find the Number of Packets Dropped due to Congestion

v. Simulate to Compare Data Rate& Throughput.

vi. Simulate to Plot Congestion for Different Source/Destination

vii. Simulate to Determine the Performance with respect to Transmission of Packets

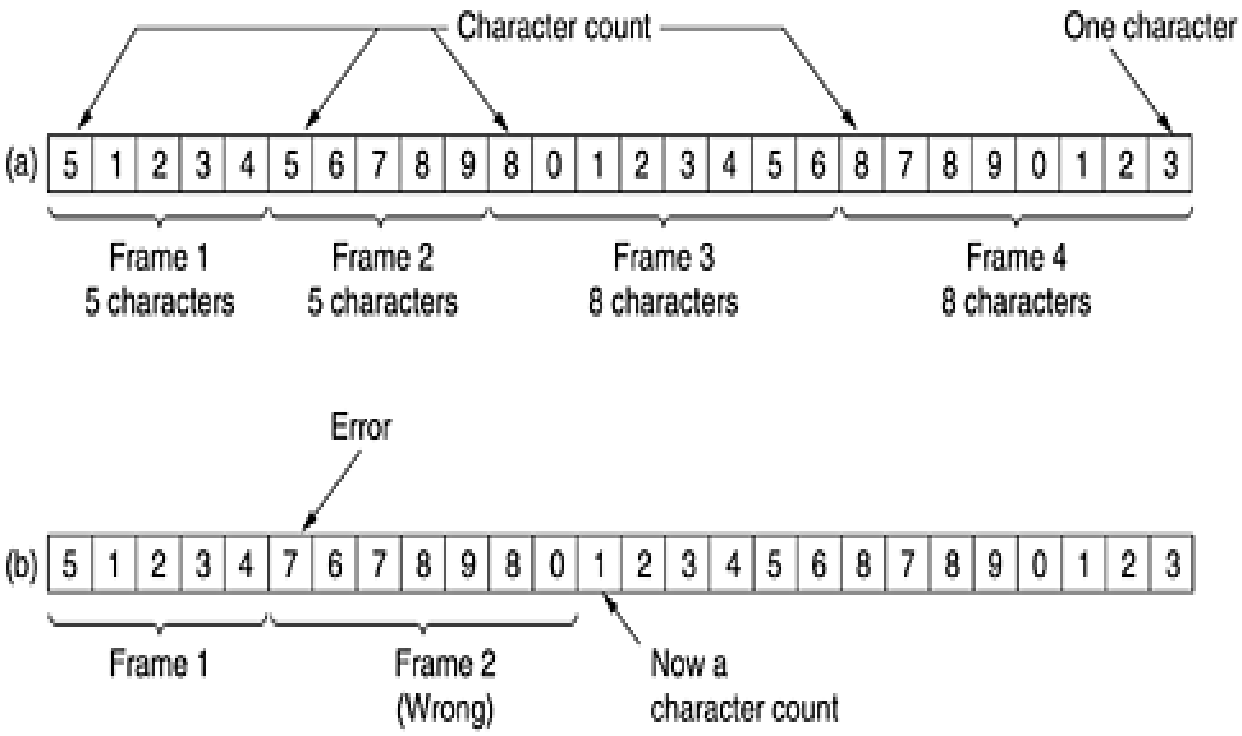
EXPERIMENT-1

Aim:

Implement the data link layer framing methods such as character count,

Character stuffing and bit stuffing.

Character count



A character stream. (a) Without errors. (b) With one error.

Program:

// Character count

```
#include<stdio.h>
#include<string.h>
char input[10][20];
int get_input();
void make_frames(int);
int count_chars(int s);
void main()
{
        int no_of_words=get_input();
        make_frames(no_of_words);
}
int get_input()
{
        int answer;
        int i=0;
        do{
                printf("\nEnter the Word:");
                scanf("%s",input[i]);
                fflush(stdin);
                printf("\nDo you want to continue: (yes: 1 or no: 0)?:");
                scanf("%d",&answer);
                i++;
```

```
        }
while(answer!=0);
        return i;
}
void make_frames(int num_words){
        int i=0;
        printf("\nThe Transmitted Data is:\n\t");

        for(;i<num_words;i++)
                printf("%d%s",(count_chars(i)+1),input[i]);
        printf("\n\n");
}
int count_chars(int index)
{
        int i=0;
        while(input[index][i]!='\0')
                i++;
        return i;
}
```

Output:

Enter the Word: cat

Do you want to continue: (y: 1/n: 0)?:1

Enter the Word: dog

Do you want to continue: (y: 1/n: 0)?:1

Enter the Word: apple

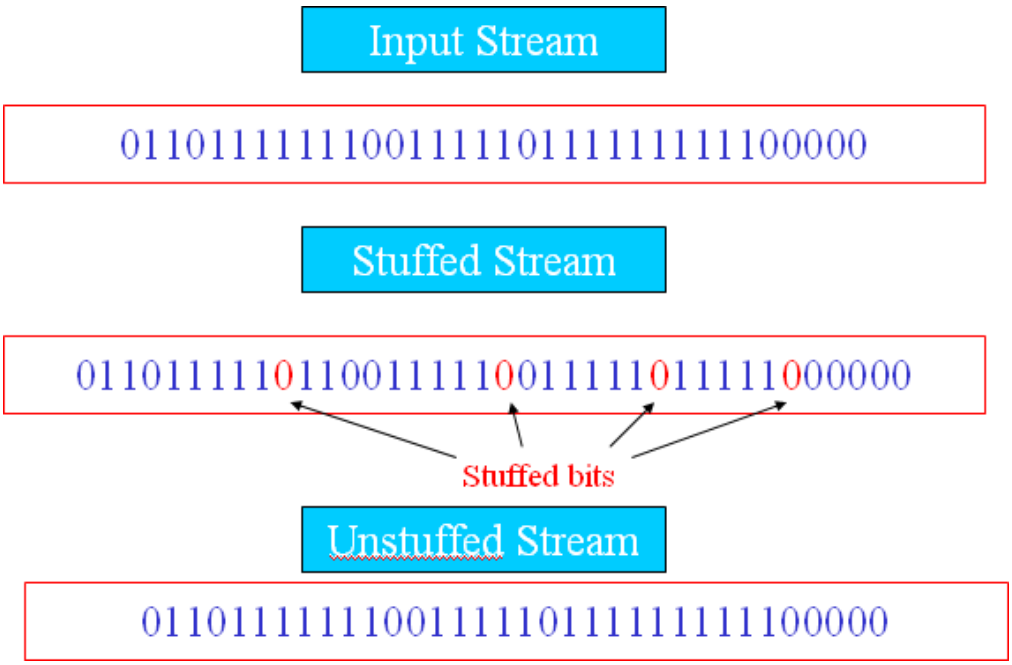
Do you want to continue: (y: 1/n: 0)?:0

The Transmitted Data is:

4cat4dog6apple

Bit Stuffing

This method allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. At the start and end of each frame is a flag byte consisting of the special bit pattern 01111110. Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a zero bit into the outgoing bit stream. This technique is called bit stuffing. When the receiver sees five consecutive 1s in the incoming data stream, followed by a zero bit, it automatically destuffs the 0 bit. The boundary between two frames can be determined by locating the flag pattern.



Program Bit Stuffing

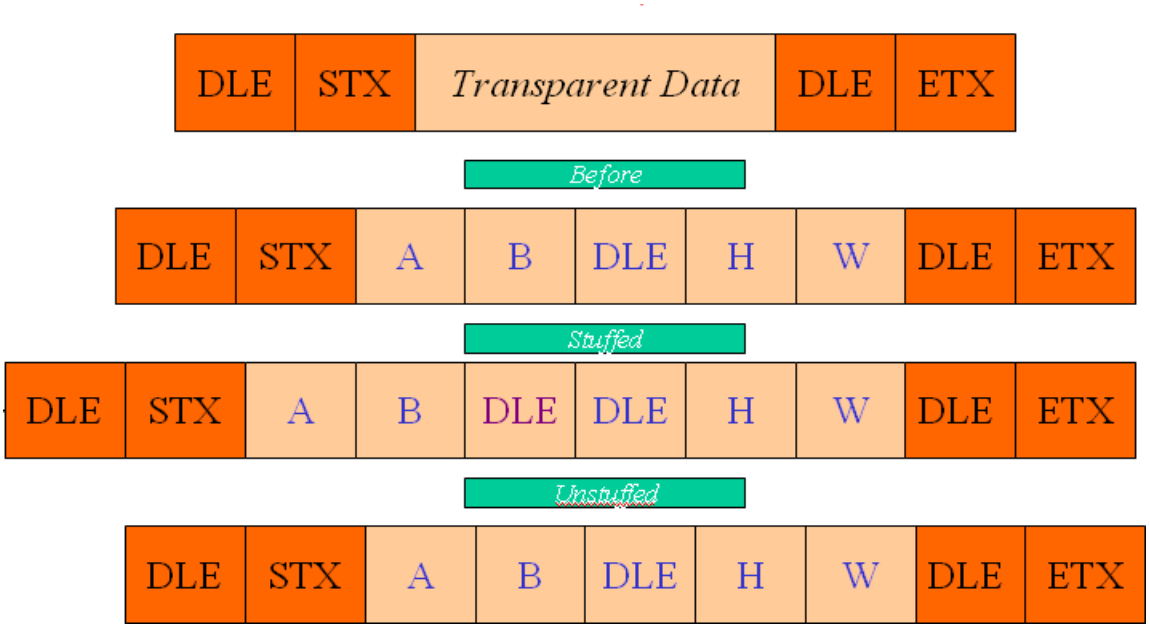
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
        int i, j,count=0,nl;
        char str[100];
        printf("enter the bit string:  ");
        gets(str);
        for (i=0;i<strlen(str);i++) {
                count=0;
                //the following code search the six ones in given string
                for (j=i;j<=(i+5);j++) {
                        if(str[j]=='1') {
                                count++;
                        }
                }
                //if there is six ones then following code execute to bit stuffing after five ones
                if(count==6) {
                        nl=strlen(str)+2;
                        for (;nl>=(i+5);nl--)
                 {
                                str[nl]=str[nl-1];
                        }
                        str[i+5]='0';
```

```
                    i=i+7;
                }
        }
        puts(str);
        getch();
}
```

output

```
enter the bit string:  0100011111111100000


01000111110111100000
```

Byte stuffing




```
#include<stdio.h>

#include<string.h>

#include<conio.h>

void main ()

{

int j,l,m,c,k;

char a[50],b[50];

printf("Enter the string:");

scanf("%s",a);

strcpy(b,"DLESTX");

m=strlen(a);
```

```
for(j=0;j<m;)

{

if(a[j]=='d')

{

if(a[j+1]=='l')

{

if(a[j+2]=='e')

{

c=j+2;

for(l=0;l<3;l++)

{

for(k=m;k>c;k--)

{

a[k]=a[k-1];

}

m++;

a[m]='\0';

c+=1;

}

a[j+3]='d';

a[j+4]='l';

a[j+5]='e';

a[m]='\0';

j+=5;
```

```
}

}

}

j++;

}

strcat(b,a);

strcat(b,"DLEETX");

printf("\n%s",b);

printf("\nReceiver side:");

m=strlen(a);

for(j=0;j<m;)

{

if(a[j]=='d')

{

if(a[j+1]=='l')

{

if(a[j+2]=='e')

{

c=j;

for(l=0;l<3;l++)

{

for(k=c;k<m;k++)

a[k]=a[k+1];

}
```

```
c++;

}

j=c;

}

}

j++;

}

printf("\n%s",a);

getch();

}
```

output

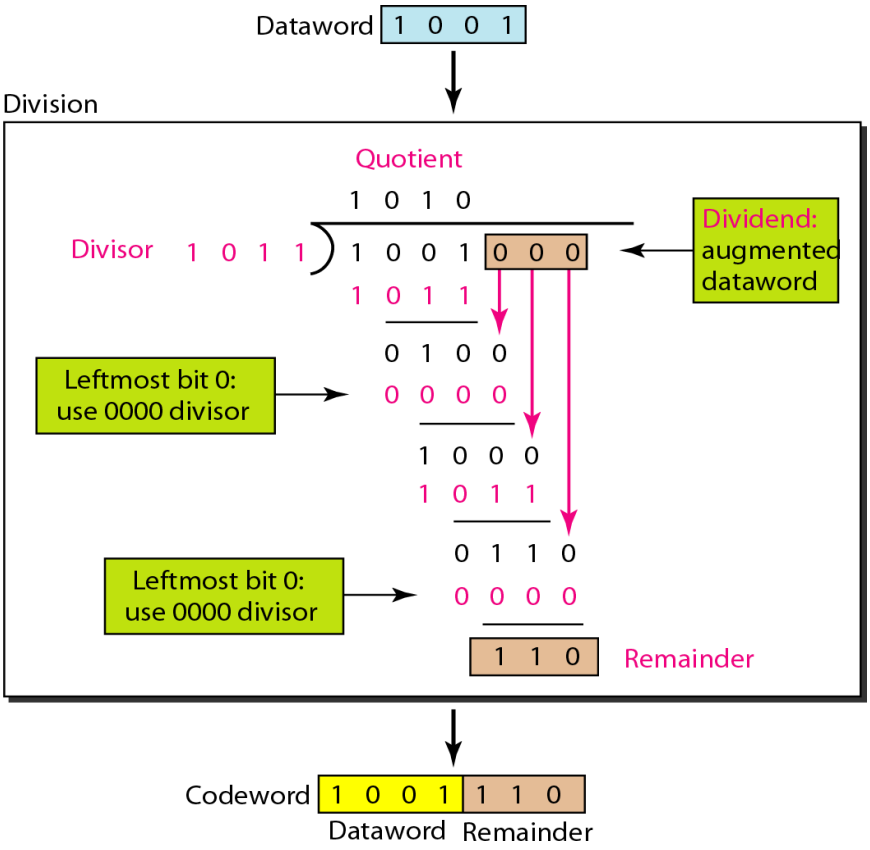
```
Enter the string:abcdidlekj



DLESTXabcdidledlekjDLEETX

Receiver side:

abcdidlekj
```

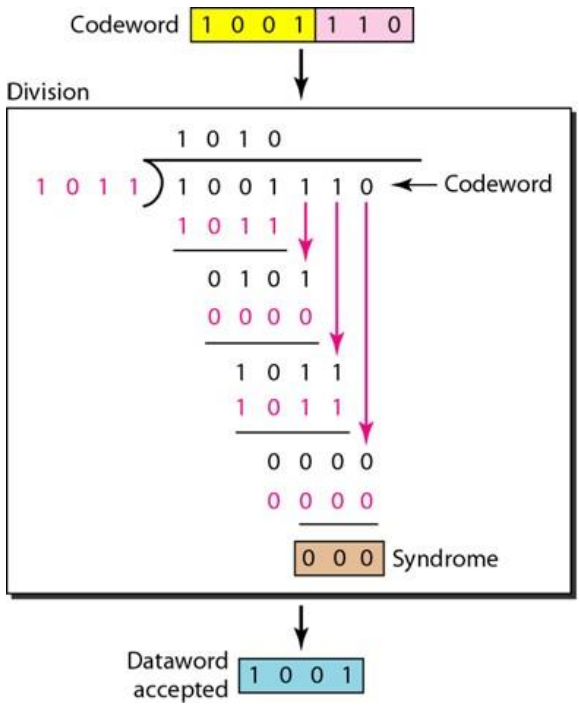
EXPERIMENT-2

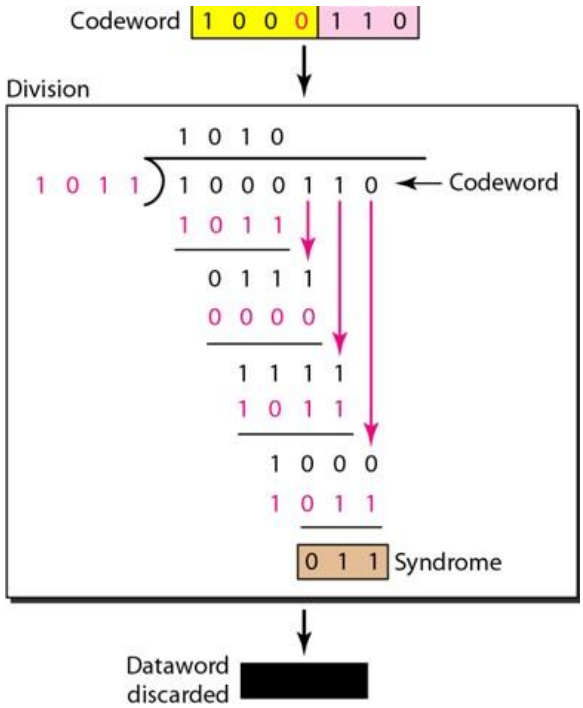
Aim:

Implement the error correcting code Cyclic Redundancy Check (CRC) of data link layer using various polynomials like CRC-CRC 12, CRC 16 and CRC CCIP.

CRC is a very effective error detection technique. This Cyclic Redundancy Check is the most powerful and easy to implement technique. Unlike checksum scheme, which is based on addition, CRC is based on binary division. In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number. At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted. A remainder indicates that the data unit has been damaged in transit and therefore must be rejected




Codeword 1 0 0 1 1 1 0
Division
1 0 1 0
1 0 1 1) 1 0 0 1 1 1 0 ← Codeword
1 0 1 1
0 1 0 1
0 0 0 0
1 0 1 1
1 0 1 1
0 0 0 0
0 0 0 0
0 0 0 Syndrome
Dataword accepted 1 0 0 1


Codeword 1 0 0 0 1 1 0
Division
1 0 1 0
1 0 1 1) 1 0 0 0 1 1 0 ← Codeword
1 0 1 1
0 1 1 1
0 0 0 0
1 1 1 1
1 0 1 1
1 0 0 0
1 0 1 1
0 1 1 Syndrome
Dataword discarded

Program:

```
#include <stdio.h>
#include <string.h>
#define N strlen(g)
char t[28],cs[28],g[28];
int a,e,c,b;
void xor()
{
	for(c=1;c<N;c++)
		cs[c]=((cs[c]==g[c])?'0':'1');
}
void crc()
{
	for(e=0;e<N;e++)
		cs[e]=t[e];
	do
	{
		if(cs[0]=='1')
			xor();
		for(c=0;c<N-1;c++)
			cs[c]=cs[c+1];
			cs[c]=t[e++];
	}
```

```
while(e<=a+N-1);
}
int main()
{
int flag=0;
do{
        printf("\n1.crc12\n2.crc16\n crc ccip\n4.exit\n\n Enter your option.");
        scanf("%d",&b);
        switch(b)
        {
                case 1:strcpy(g,"1100000001111");
                break;
                case 2: strcpy(g,"11000000000000101");
                break;
                case 3:strcpy (g,"10001000000100001");
                break;
                case 4:return 0;
        }
printf("\n enter data:");
scanf("%s",t);
printf("\n_____\n");
printf("\n generating polynomial:%s",g);
a=strlen(t);
for(e=a;e<a+N-1;e++)
```

```
        t[e]='0';
printf("\n____________________\n");
printf("mod-ified data is:%s",t);
printf("\n_________________\n");
crc();
printf("checksum is:%s",cs);
for(e=a;e<a+N-1;e++)
        t[e]=cs[e-a];
printf("\n_________________\n");
printf("\n final codeword is : %s",t);
printf("\n_________________\n");
printf("\ntest error detection 0(yes) 1(no)?:");
scanf("%d",&e);
if(e==0)
{
        do{
                printf("\n\tenter the position where error is to be inserted:");
                scanf("%d",&e);
        }
        while(e==0||e>a+N-1);
        t[e-1]=(t[e-1]=='0')?'1':'0';
        printf("\n_________________\n");
        printf("\n\terroneous data:%s\n",t);
}
```

```
crc();
for(e=0;(e<N-1)&&(cs[e]!='1');e++);
if(e<N-1)
        printf ("error detected\n\n");
else
        printf ("\n no error detected \n\n");
printf("\n_____");

}while(flag!=1);
}
```

Output:

1.crc12

2.crc16

3.crc ccit

4.exit

Enter your option.1

enter data: 1100110011100011

---

generating polynomial: 1100000001111

---

mod-ified data is: 1100110011100011000000000000

---

checksum is: 110111011000

---

final codeword is: 1100110011100011110111011000

---

test error detection 0(yes) 1(no)?:1

no error detected

---

1. crc12

2. crc16

3. crc ccit

4. exit

Enter your option.2

enter data: 11001100111000

---

generating polynomial: 11000000000000101

Mod-ified data is: 11001100111000000000000000000

Checksum is: 11111111110110000

---

final codeword is: 1100110011100011111111110110000000000000000101

Test error detection 0(yes) 1(no)?:1

No error detected

1.crc12

2.crc16

3.crc ccit

4.exit

Enter your option.3

enter data:11001100111000

.....-

generating polynomial:10001000000100001

----

mod-ified data is:11001100111000000000000000000

.....-

checksum is:11100111100111010

.....-

final codeword is : 11001100111000111001111001110100

----

test error detection 0(yes) 1(no)?:0

Enter the position where error is to be inserted:3

.....-

erroneous data:11101100111000111001111001110100100000010000 1

error detected

.....-

1. crc12

2. crc16

3. crc ccit

4. exit

Enter your option.4

Result:

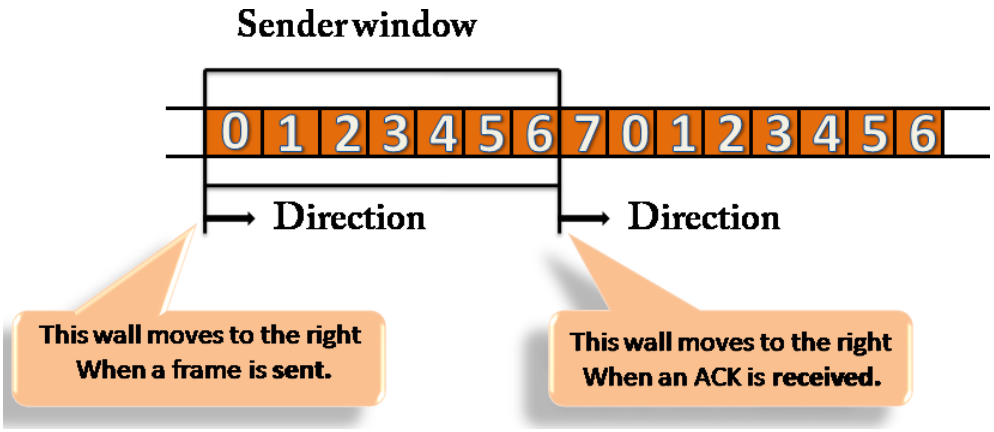
Thus the program for cyclic redundancy check is executed.

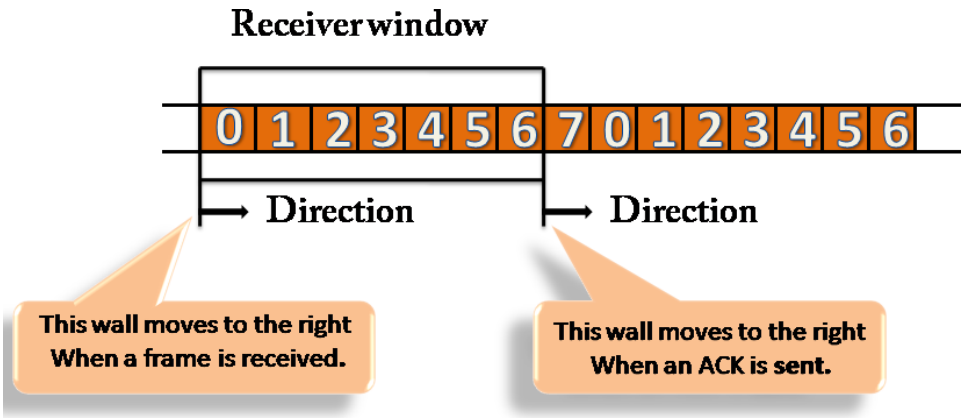
EXPERIMENT-3

Aim:

Develop a simple data link layer that performs the flow control using the sliding window protocol and loss recovery using the Go-Back-N mechanism

SLIDING WINDOW PROTOCOL





PROGRAM

```
#include<stdio.h>
 int main()
{
   int w,i,f,frames[50];

   printf("Enter window size: ");
   scanf("%d",&w);

   printf("\nEnter number of frames to transmit: ");
   scanf("%d",&f);

   printf("\nEnter %d frames: ",f);

   for(i=1;i<=f;i++)
      scanf("%d",&frames[i]);

   printf("\nWith sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)\n\n");
   printf("After sending %d frames at each stage sender waits for acknowledgement sent by the receiver\n\n",w);

   for(i=1;i<=f;i++)
   {
```

```
        if(i%w==0)
        {
            printf("%d\n",frames[i]);
            printf("Acknowledgement of above frames sent is received by sender\n\n");
        }
        else
            printf("%d ",frames[i]);
    }

    if(f%w!=0)
        printf("\nAcknowledgement of above frames sent is received by sender\n");

    return 0;
}
```

Output

Enter window size: 3

Enter number of frames to transmit: 5

Enter 5 frames: 12 5 89 4 6

With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)

After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver

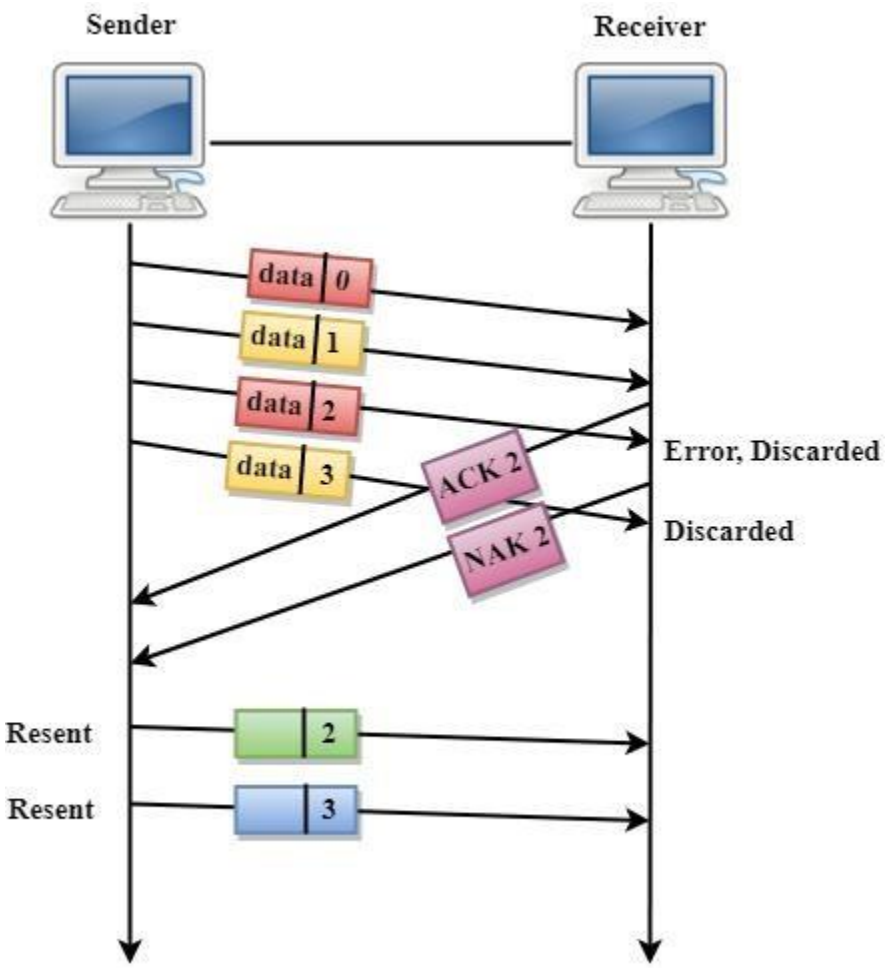
12 5 89

Acknowledgement of above frames sent is received by sender

4 6

Acknowledgement of above frames sent is received by sender

loss recovery using the Go-Back-N mechanism



Program

```
#include<stdio.h>

        int main()

{

        int windowsize,sent=0,ack,i;
```

```
        printf("enter window size\n");
        scanf("%d",&windowsize);
        while(1)
        {
            for( i = 0; i < windowsize; i++)
                {
                    printf("Frame %d has been transmitted.\n",sent);
                    sent++;
                    if(sent == windowsize)
                        break;
                }
            printf("\nPlease enter the last Acknowledgement received.\n");
                scanf("%d",&ack);
                if(ack == windowsize)
                break;
                else
                sent = ack;
        }
return 0;
}
```

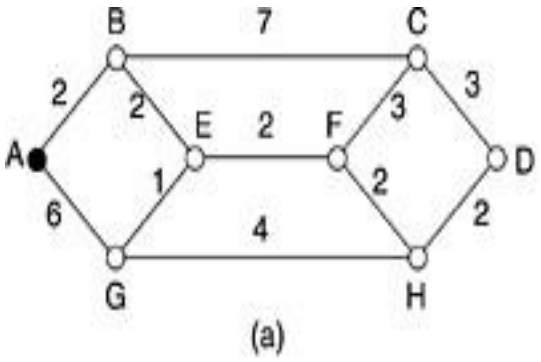
Output:-

```
enter window size
8
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
2
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.
Please enter the last Acknowledgement received.
8
```

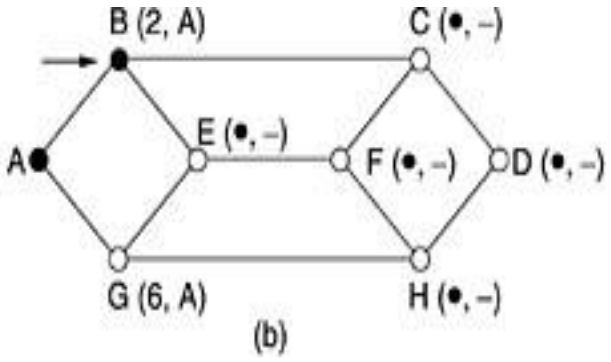
EXPERIMENT-4

Aim:

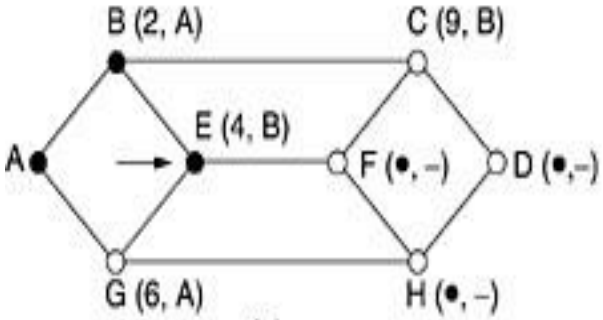
Implement Dijkstra‘s algorithm to compute the Shortest path through a graph.



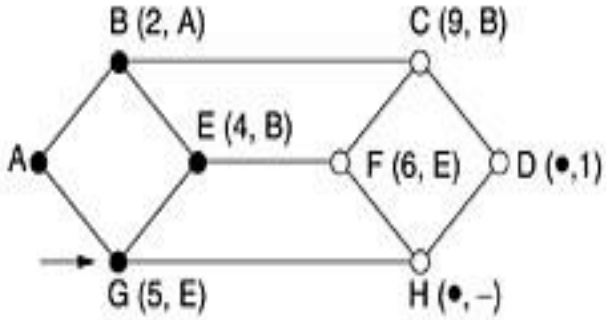
(a)



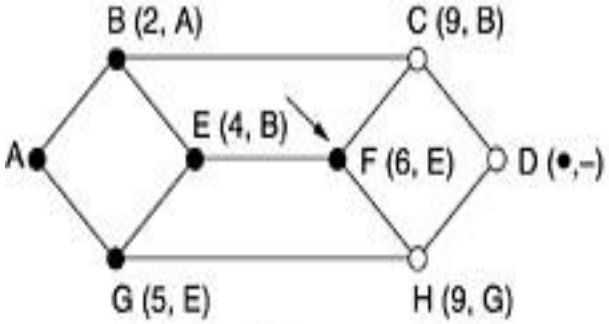
(b)



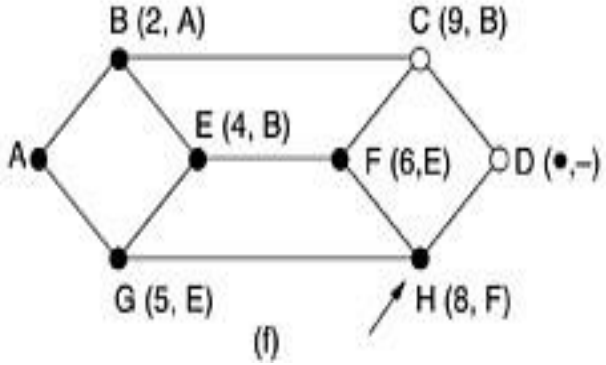
(c)



(d)



(e)



(f)

Finally, Destination ‘D’ is relabeled as D(10,H). The path is **(D-H-F-E-B-A)** as follows:

D (10, H) = H (8, F)

= F (6, E)

= E (4, B)

= B (2, A)

= A

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_NODES 1024
#define INFINITY 1000
int n=8,cost=0,dist[8][8]={{0,2,0,0,0,0,6,0},
                {2,0,7,0,2,0,0,0},
                {0,7,0,3,0,3,0,0},
            {0,0,3,0,0,0,0,2},
            {0,2,0,0,0,2,1,0},
            {0,0,3,0,2,0,0,2},
            {6,0,0,0,1,0,0,4},
            {0,0,0,2,0,2,4,0}};
```

```
int shortest_dist(int s,int t,int path[])
{
        int i,k,min;
        struct state
        {
                int  pre;
                int length;
                int label;
        }
state[1024];

        struct state *p;
                for(p=&state[0];p<&state[n];p++)
        {
                p->pre=-1;
                p->length=INFINITY;
                p->label=0;
        }

        state[0].length=0;
        state[0].label=1;
        state[0].pre=-1;
        k=t;
        do
        {
        for(i=0;i<n;i++)
                if(dist[k][i]!=0 && state[i].label==0)
                {
```

```
                    if(state[k].length+dist[k][i]<state[i].length)
                    {
                        state[i].pre=k;
                        state[i].length=state[k].length+dist[k][i];
                    }
                }

        k=0;
        min=INFINITY;
        for(i=0;i<n;i++)

                if(state[i].label==0 && state[i].length<min)
                {
                    min=state[i].length;
                    k=i;
                }
                state[k].label=1;
        }
while(k!=s);
        i=0;
        k=s;
        do
        {
                path[i++]=k;
                k=state[k].pre;
                cost+=state[k].length;
        }
while(k>=0);
```

```
return  i;
}
void main()
{
        int i,j,m,path[102],q,p;
        printf("\nEnter Number of nodes(1-8): ");
        scanf("%d",&n);
        printf("\nEnter Source Vertex(1-8): ");
        scanf("%d",&p);

        printf("\nEnter Destination vertex(1-8): ");
        scanf("%d",&q);
        m=shortest_dist(q-1,p-1,path);
        for(i=0;i<m;i++)
        {
                printf(" %c-> ",path[i]+'A');
        }
        printf("\nCost is:  %d \n",cost);
}
```

Output:

```
Enter Number of nodes(1-8): 7
Enter Source Vertex(1-8): 1
Enter Destination vertex(1-8): 7
G->  E->  B->  A->
Cost is:  6
```

EXPERIMENT-5

Aim:

Take an example subnet of hosts. Obtain broadcast tree for it.

Sending a packet to all destinations simultaneously is called broadcasting. The set of optimal routes from a source to all destinations form a tree rooted at the source. Such a tree is called a sink tree. It is illustrated in Fig., where the distance metric is the number of hops. Note that a sink tree is not necessarily unique; other trees with the same path lengths may exist. The goal of all routing algorithms is to discover and use the sink trees for all routers

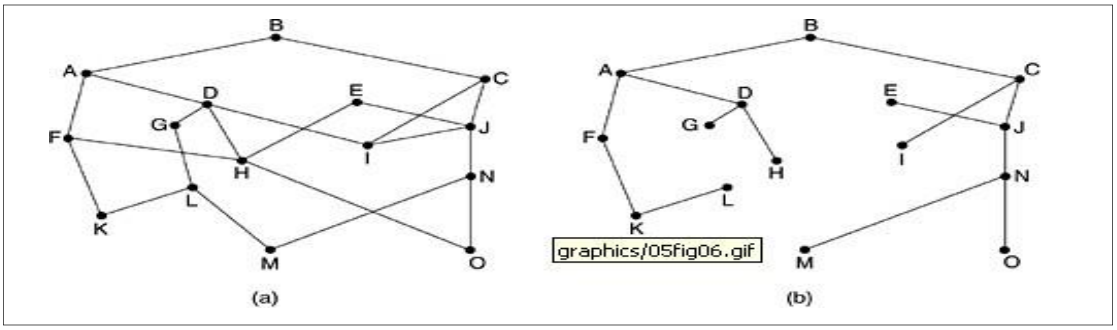


Figure: a) subnet b) sink tree for router B

A broadcast algorithm makes explicit use of the sink tree for the router initiating the broadcast. A sink tree is a subset of the subnet that includes all the routers but contains no loops. If each router knows which of its lines belong to the sink tree, it can copy an incoming broadcast packet onto all the sink tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some sink tree for the method to be applicable.

Program:

```
#include<stdio.h>

#include<stdlib.h>

#define MAX_NODES 20

#define INFINITY 1000
```

```
void run_prim(int i,int num_nodes,int *hops);
int dist[MAX_NODES][MAX_NODES];
int path[MAX_NODES][MAX_NODES];
int main()
{
        int num_nodes=0;
        int i=0,j=0;
        int path_len,hops[MAX_NODES];

        printf("enter the number of nodes:");
        scanf("%d",&num_nodes);
        fflush(stdin);
        printf("enter the connection matrix, 0 if not connected\n");
        printf("otherwise,the distance\n");
        for(;i<num_nodes;i++)
        {
                printf("enter the distances for node num:%d\n",i);
                for(j=0;j<num_nodes;j++)
                {
                        if(i==j)
                        {
                                dist[i][j]=0;
                                continue;
                        }
                        printf("distance from %d to %d= ",i,j);
                        scanf("%d",&dist[i][j]);
```

```
                fflush(stdin);
                if(dist[i][j]==0)
                        dist[i][j]=INFINITY;
        }
    }

    printf("\nenter the root node");
    scanf("%d",&i);
    run_prim(i,num_nodes,hops);
    for(j=0;j<num_nodes;j++)
    {
        for(i=0;i<hops[j];i++)
            printf("->%d",path[j][i]);
        printf("\n");
    }
    return 0;
}
void run_prim(int s,int n,int *hops)
{
    struct state
    {
    int prev;
        int length;
        enum {perm,tent} label;
    }
state[MAX_NODES];
    int i,j,k,min;
```

```
        int count=0;
        struct state *p;
        for(p=&state[0];p<&state[n];p++)
        {
                p->prev= -1;
                p->length= INFINITY;
                p->label= tent;
        }
        state[s].length= 0;state[s].label= perm;
        k=s;
        do
        {
        for(i=0;i<n;i++)
                        if(dist[k][i]!=0&&state[i].label==tent)
                        {
if(state[k].length+dist[k][i]<state[i].length)
{
state[i].prev=k;
state[i].length=state[k].length+dist[k][i];
                                }
                        }
k=0;
                min=INFINITY;
                for(i=0;i<n;i++)
                        if(state[i].label==tent&&state[i].length<min)
                        {
                                min=state[i].length;
```

```
                        k=i;
                    }
                state[k].label=perm;
                count++;
        }while(count<n);

        for(j=0;j<n;j++)
        {
                i=0;k=j;
                do
                {
                        path[j][i++]=k;
                        k=state[k].prev;
                }

                while(k>=0);
                hops[j]=i;
        }
}
```

Output:

```
enter the number of nodes:4
enter the connection matrix, 0 if not connected, otherwise the distance
enter the distances for node num:0
distance from 0 to 1= 2
distance from 0 to 2= 1
```

distance from 0 to 3= 6

enter the distances for node num:1

distance from 1 to 0= 5

distance from 1 to 2= 1

distance from 1 to 3= 3

enter the distances for node num:2

distance from 2 to 0= 5

distance from 2 to 1= 1

distance from 2 to 3= 3

enter the distances for node num:3

distance from 3 to 0= 5

distance from 3 to 1= 4

distance from 3 to 2= 6

enter the root node2

->0->2

->1->2

->2

->3->2

Result:

Thus the program to obtain broadcast tree for an example subnet of hosts is executed.

EXPERIMENT-6

Aim:

Implement distance vector routing algorithm for obtaining routing tables at each node.

```
#include<stdio.h>

struct node
{
unsigned dist[20];
unsigned from[20];
}
rt[10];
int main()
{
int dmat[20][20];
int n,i,j,k,count=0;
printf("enter the number of nodes:");
scanf("%d",&n);
printf("enter the cost matrix :\n");

for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
scanf("%d",&dmat[i][j]);
dmat[i][i]=0;
rt[i].dist[j]=dmat[i][j];
rt[i].from[j]=j;
}

do
{
count=0;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
for(k=0;k<n;k++)
if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
```

```
rt[i].from[j]=k;
count++;
}

}
while(count!=0);

for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
printf("\n state value for router %d is\n",i+1);
for(j=0;j<n;j++)
{
printf("\n node %d via %d Distance %d", j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n");
}
```

Output:

```
enter the number of nodes:3
enter the root matrix:
1 2 4
2 3 5
4 5 6
state value for router 1 is

node 1 via 1 distance 0
node 2 via 2 distance 2
node 3 via 3 distance 4

state value for router 2 is

node 1 via 1 distance 2
node 2 via 2 distance 0
node 3 via 3 distance 5

state value for router 3 is

node 1 via 1 distance 4
node 2 via 2 distance 5
node 3 via 3 distance 0
```

Result:

Thus the program for obtain Routing table art each node using distance vector routing algorithm is executed.

EXPERIMENT-7

7. Implement data encryption and data decryption

```
#include <stdio.h>
int main()
{
  int i, x;
  char str[100];
  printf("\nPlease enter a string:\t");
  gets(str);
  printf("\nPlease choose following options:\n");
  printf("1 = Encrypt the string.\n");
  printf("2 = Decrypt the string.\n");
  scanf("%d", &x);
  //using switch case statements
  switch(x)
  {
  case 1:
    for(i = 0; (i < 100 && str[i] != '\0'); i++)
      str[i] = str[i] + 3; //the key for encryption is 3 that is added to ASCII value
```

```
    printf("\nEncrypted string: %s\n", str);
    break;
  case 2:
    for(i = 0; (i < 100 && str[i] != '\0'); i++)
      str[i] = str[i] - 3; //the key for encryption is 3 that is subtracted to ASCII value
    printf("\nDecrypted string: %s\n", str);
    break;
  default:
    printf("\nError\n");
  }
  return 0;
}
```

Output:

Encryption:

```
Please enter a string:  hello

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
1

Encrypted string: khoor

Process returned 0 (0x0)   execution time : 8.564 s
Press any key to continue.
```

Decryption:

```
Please enter a string:  khoor

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
2

Decrypted string: hello

Process returned 0 (0x0)   execution time : 4.288 s
Press any key to continue.
```

EXPERIMENT-8

Write a program for congestion control using Leaky bucket algorithm.

```
#include<stdio.h>

int min(int x,int y)
{
	if(x < y)
	{
		return x;
	}
	else
		return y;
}

int main()
{
	int drop=0, mini, nsec, cap, count = 0;
int i,inp[25],process;
	printf("Enter the Bucket Size:\n");
	scanf("%d",&cap);
	printf("Enter the Processing Rate:\n");
	scanf("%d",&process);
	printf("Enter The No. Of Seconds You Want To Stimulate:\n");
	scanf("%d",&nsec);

	for(i=0;i<nsec;i++)
	{
		printf("Enter the Size of the Packet Entering at %d sec:\n",i+1);
		scanf("%d",&inp[i]);
	}

	printf("\nSeconds|Packet Recieved|Packet Sent|PacketLeft|Packet Dropped|\n");
	printf("\n");
```

```
        for(i=0;i<nsec;i++)
        {
                count+=inp[i];
                if(count>cap)
                {
                        drop=count-cap;
                        count=cap;
                }
                printf("%d",i+1);
                printf("\t%d",inp[i]);
                mini=min(count,process);
                printf("\t%d",mini);
                count=count-mini;
                printf("\t%d",count);
                printf("\t%d\n",drop);
                drop=0;
        }


        for(;count!=0;i++)
        {
                if(count>cap)
                {
                        drop=count-cap;
                        count=cap;
                }
                printf("%d",i+1);
                printf("\t0");
                mini=min(count,process);
                printf("\t%d",mini);
                count=count-mini;
                printf("\t%d",count);
                printf("\t%d\n",drop);

        }
}
```

OUTPUT:

Enter the Bucket Size: 5

Enter the Processing Rate: 2

Enter The No. Of Seconds You Want To Stimulate: 3

Enter the Size of the Packet Entering at 1 sec: 5

Enter the Size of the Packet Entering at 2 sec: 4

Enter the Size of the Packet Entering at 3 sec: 3

Seconds	Packet Received	Packet Sent	Packet Left	Packet Dropped
1	5	2	3	0
2	4	2	3	2
3	3	2	3	1
4	0	2	1	0
5	0	1	0	0

Experiment 9

9. Write a program for frame sorting technique used in buffers

```
#include<stdio.h>
#include<string.h>
#define FRAM_TXT_SIZ 3
#define MAX_NOF_FRAM 127
char str[FRAM_TXT_SIZ*MAX_NOF_FRAM];
struct frame // structure maintained to hold frames
{
char text[FRAM_TXT_SIZ];
int seq_no;
}
fr[MAX_NOF_FRAM], shuf_ary[MAX_NOF_FRAM];
int assign_seq_no() //function which splits message
{
 int k=0,i,j; //into frames and assigns sequence no
for(i=0; i < strlen(str); k++)
{
 fr[k].seq_no = k;
```

```
for(j=0; j < FRAM_TXT_SIZ && str[i]!='\0'; j++)
fr[k].text[j] = str[i++];
}
printf("\nAfter assigning sequence numbers:\n");
for(i=0; i < k; i++)
printf("%d:%s ",i,fr[i].text);
return k; //k gives no of frames
}
void generate(int *random_ary, const int limit) //generate array of random nos
{
 int r, i=0, j;
while(i < limit)
{ r = random() % limit;
for(j=0; j < i; j++)
if( random_ary[j] == r )
break;
if( i==j ) random_ary[i++] = r;
}
 }
void shuffle( const int no_frames ) // function shuffles the frames
{
int i, k=0, random_ary[no_frames];
generate(random_ary, no_frames);
for(i=0; i < no_frames; i++)
shuf_ary[i] = fr[random_ary[i]];
printf("\n\nAFTER SHUFFLING:\n");
for(i=0; i < no_frames; i++)
```

```
printf("%d:%s ",shuf_ary[i].seq_no,shuf_ary[i].text);
}
void sort(const int no_frames) // sorts the frames
{
int i,j,flag=1;
struct frame hold;
for(i=0; i < no_frames-1 && flag==1; i++) // search for frames in sequence
{
flag=0;
for(j=0; j < no_frames-1-i; j++) //(based on seq no.) and display
if(shuf_ary[j].seq_no > shuf_ary[j+1].seq_no)
{
hold = shuf_ary[j];
shuf_ary[j] = shuf_ary[j+1];
shuf_ary[j+1] = hold;
flag=1;
}
}
}
int main()
{
int no_frames,i;
printf("Enter the message: ");
gets(str);
no_frames = assign_seq_no();
shuffle(no_frames);
sort(no_frames);
```

```
printf("\n\nAFTER SORTING\n");
for(i=0;i<no_frames;i++)
printf("%s",shuf_ary[i].text);
printf("\n\n");
}
```

Output:

EXPERIMENT-10

Title:

Wireshark

Aim:

To work with Wireshark Tool

Theory

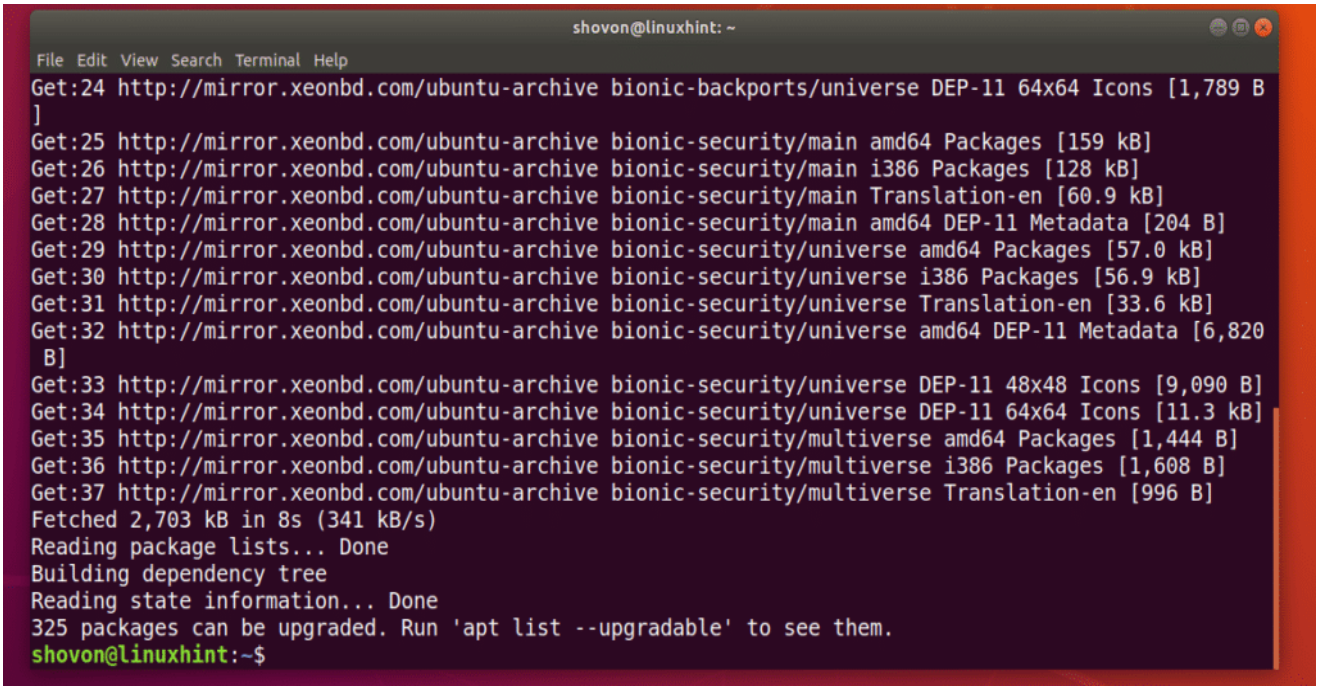
- Wireshark is a powerful open source network analyser which can be used to sniff the data on a network, as an aide to troubleshooting network traffic analysis, but equally as an educational tool to help understand the principles of networks and communication protocols.

Installing Wireshark:

- It is readily available for just about any Linux distribution and for Ubuntu, it can be installed via the Ubuntu Software Center or the terminal:
- First update the APT package repository cache with the following command:

  $ sudo apt update

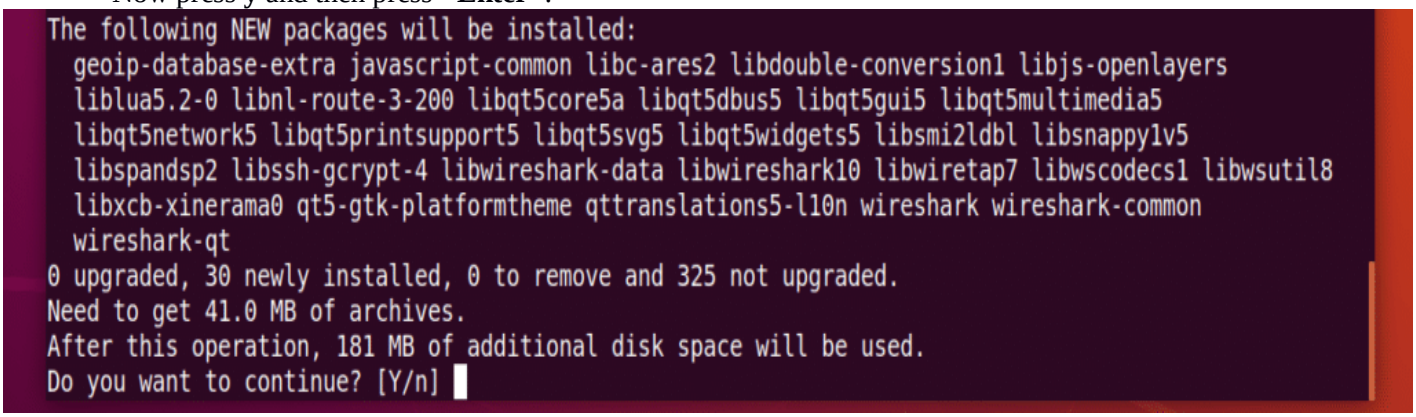
  The APT package repository cache should be updated.



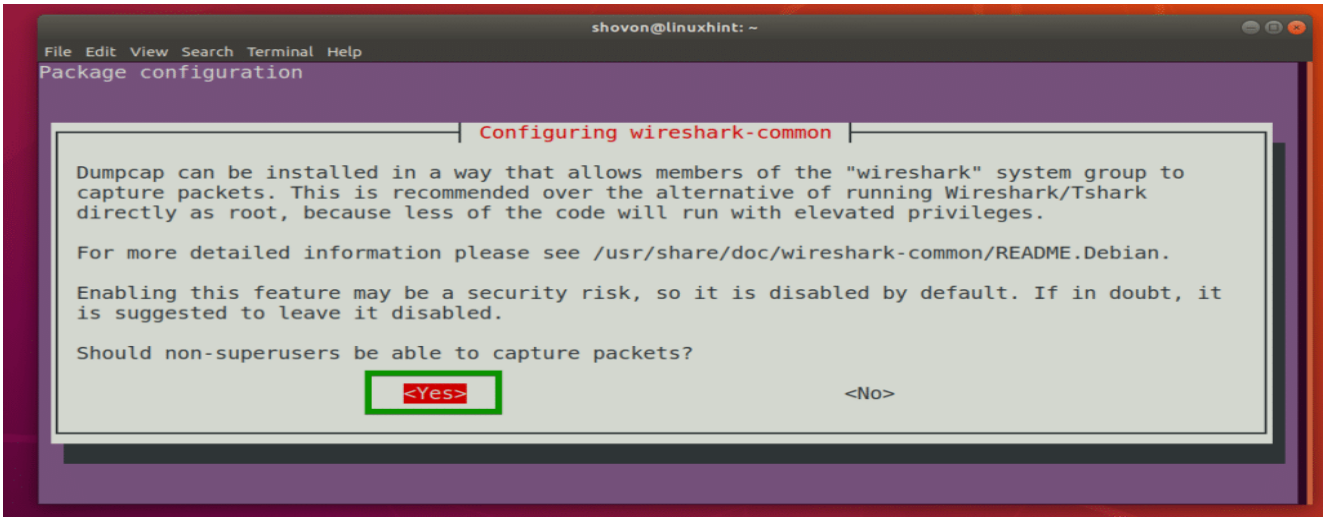
Now, Run the following command to install Wireshark on your Ubuntu machine:

$ sudo apt install wireshark (or **sudoapt-get install** wireshark)

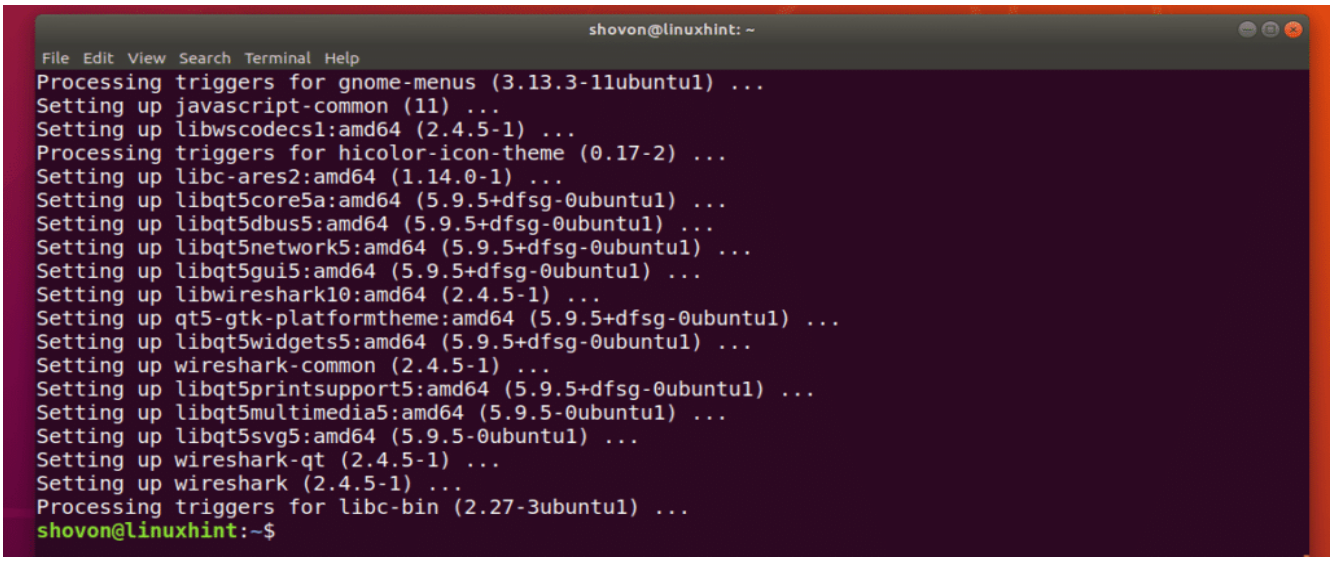
Now press **y** and then press **<Enter>**.



By default, Wireshark must be started as **root** (can also be done with **sudo**) privileges in order to work. If you want to run Wireshark without **root** privileges or without **sudo**, then select **<Yes>** and press **<Enter>**.



Wireshark should be installed.



Now if you selected <**Yes**> in the earlier section to run Wireshark without root access, then run the following command to add your user to the **wireshark** group:

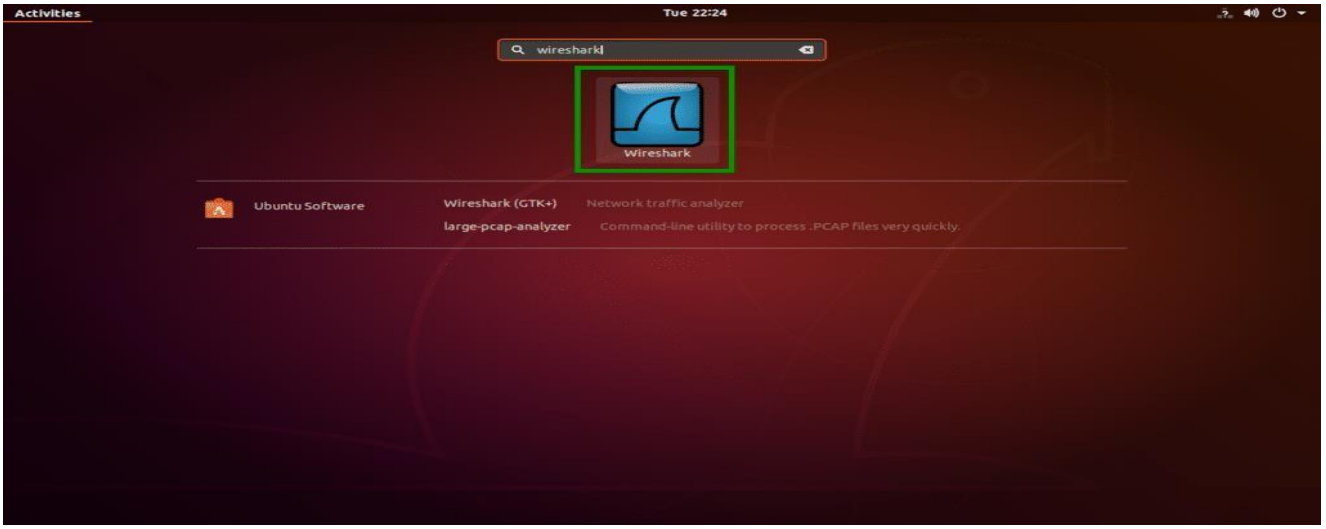
```
$ sudo usermod -aG wireshark $(whoami)
```

Finally, reboot your computer with the following command

```
$ sudo reboot
```

Starting Wireshark:

Now that Wireshark is installed, you can start Wireshark from the Application Menu **of Ubuntu**



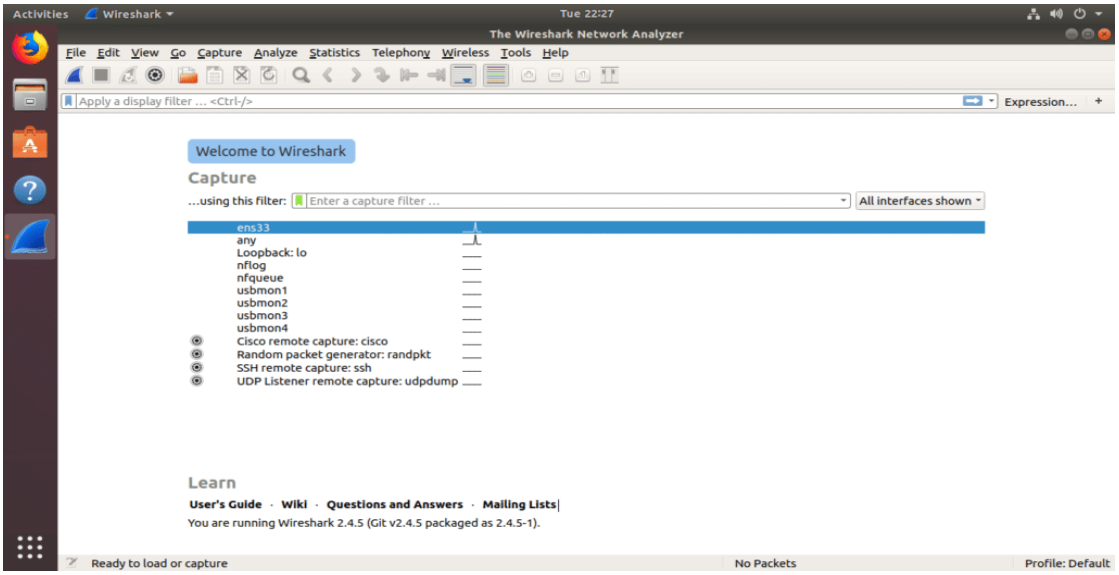
You can also run the following command to start Wireshark from the Terminal:

$ wireshark

If you did not enable Wireshark to run without **root** privileges or **sudo**, then the command should be

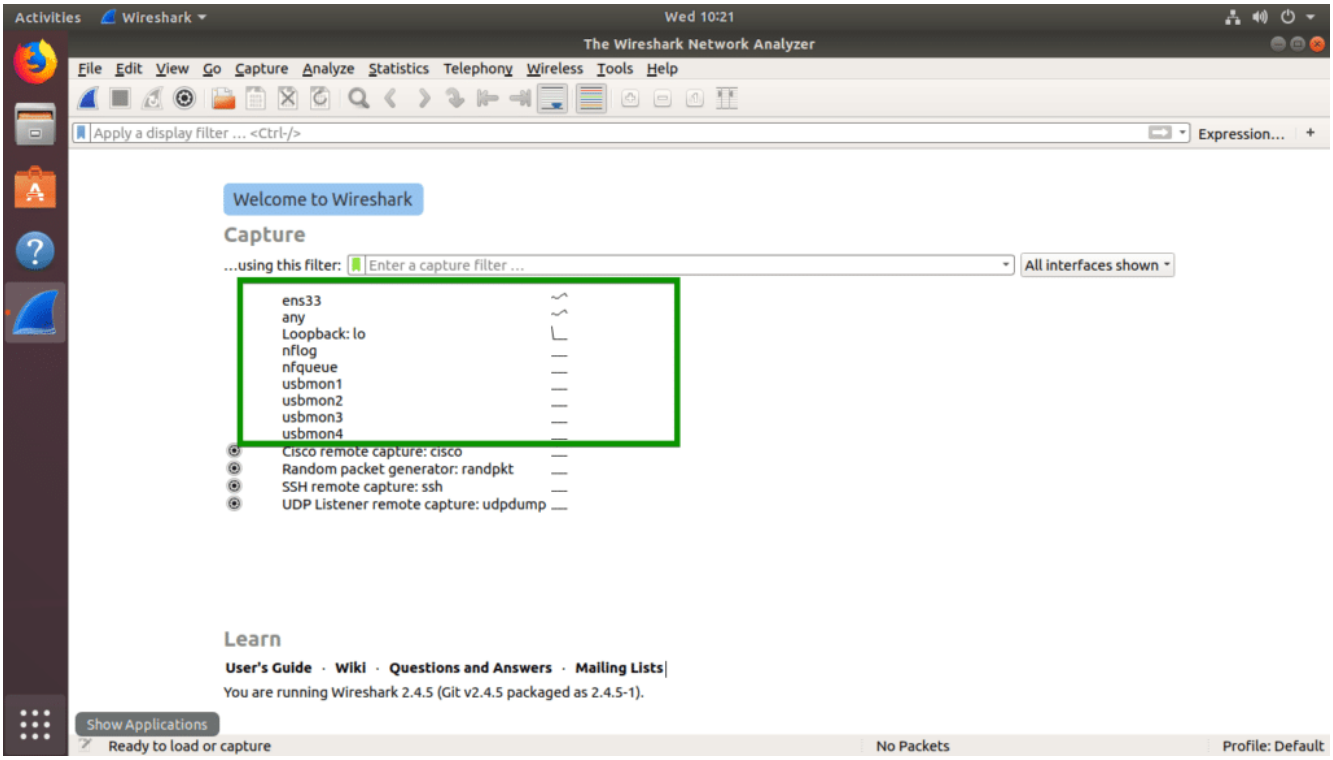
$ sudo wireshark

Wireshark should start.

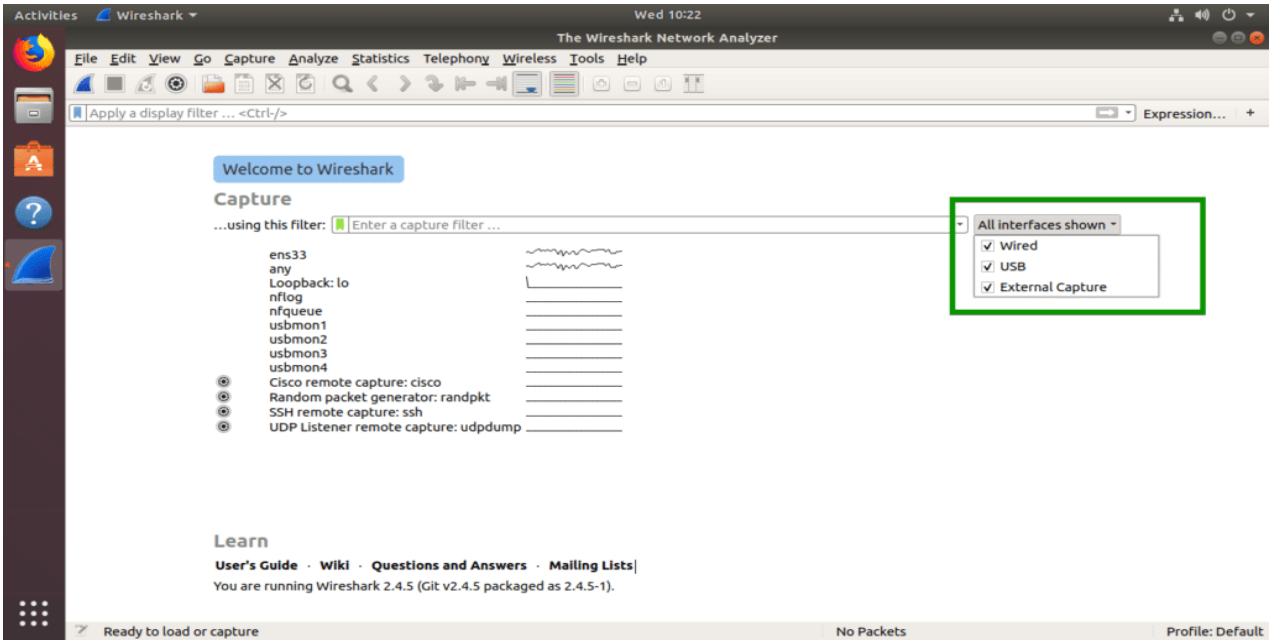


Capturing Packets Using Wireshark:

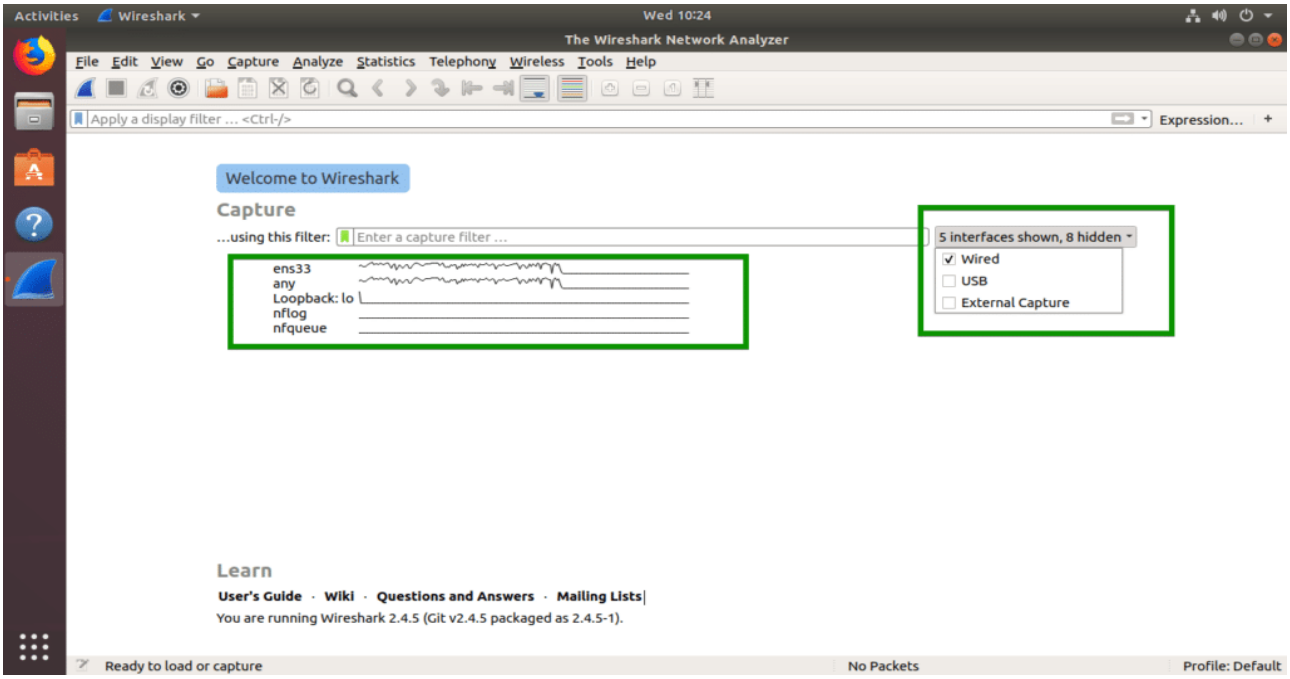
When you start Wireshark, you will see a list of interfaces that you can capture packets to and from.



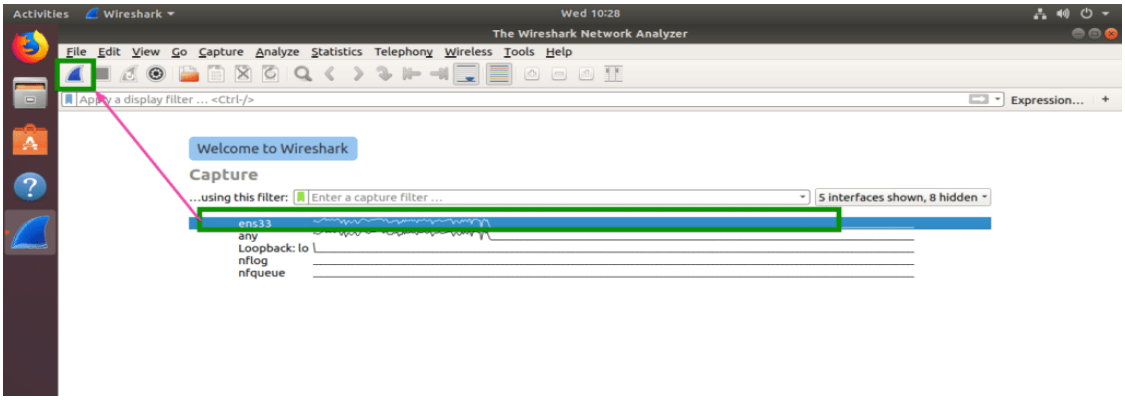
There are many types of interfaces you can monitor using Wireshark, for example, **Wired**, **Wireless**, USB and many external devices. You can choose to show specific types of interfaces in the welcome screen from the marked section of the screenshot below.



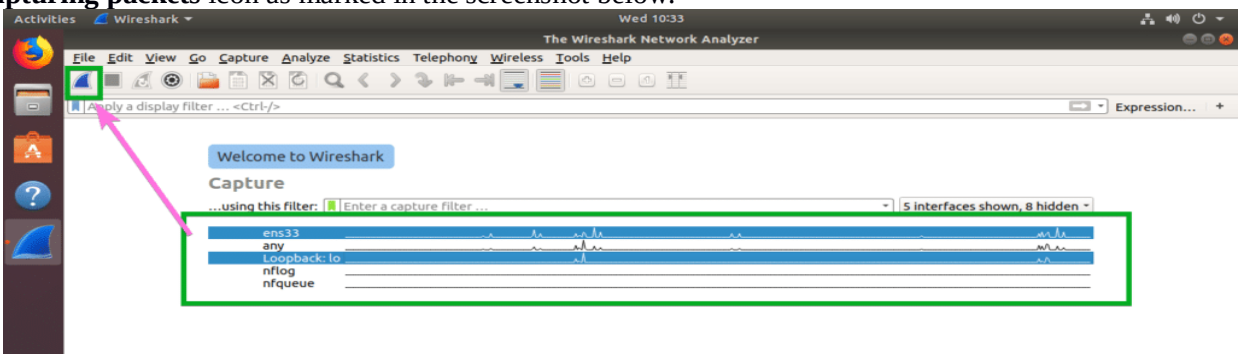
Here, I listed only the **Wired** network interfaces.



Now to start capturing packets, just select the interface (in my case interface **ens33**) and click on the **Start capturing packets** icon as marked in the screenshot below. You can also double click on the interface that you want to capture packets to and from to start capturing packets on that particular interface

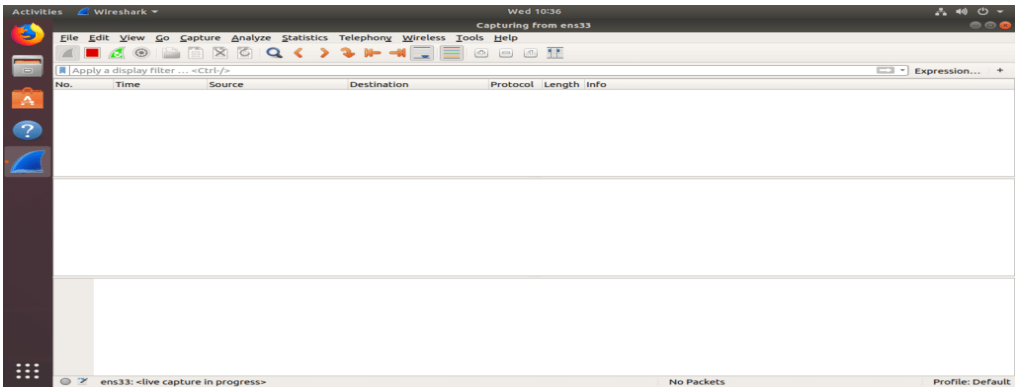


You can also capture packets to and from multiple interfaces at the same time. Just press and hold **<Ctrl>** and click on the interfaces that you want to capture packets to and from and then click on the **Start capturing packets** icon as marked in the screenshot below.

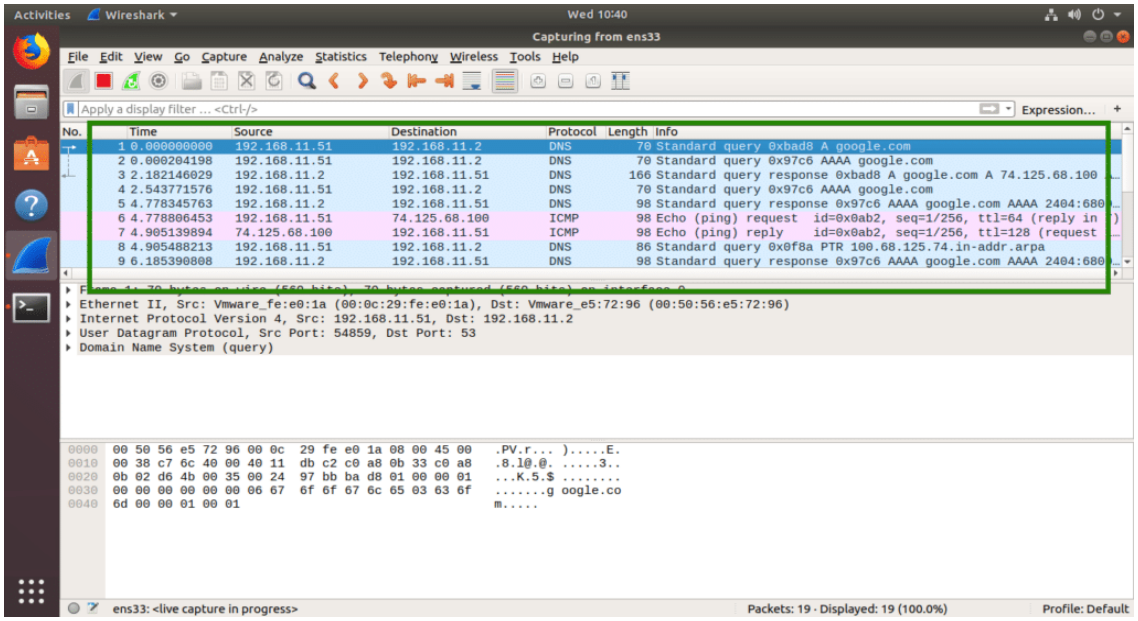


Using Wireshark on Ubuntu:

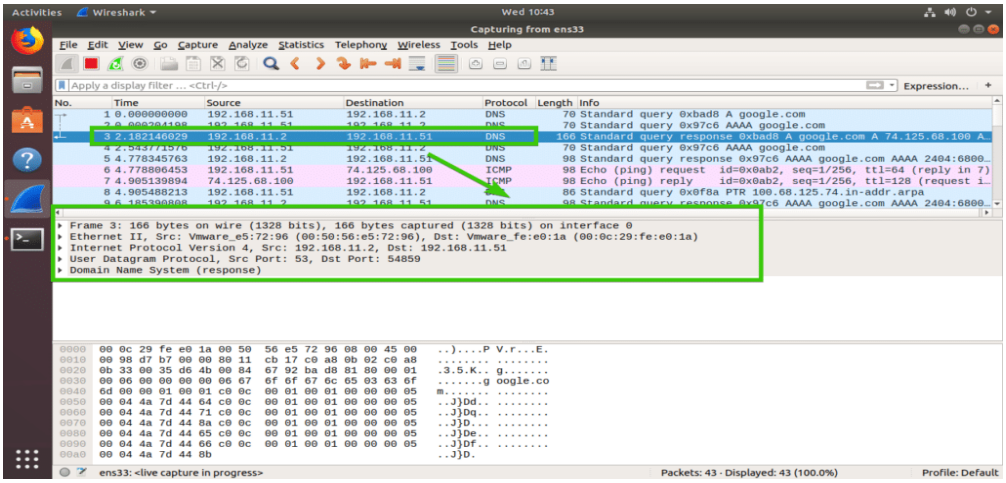
capturing packets on the **ens33** wired network interface as you can see in the screenshot below. Right now, no captured packets.



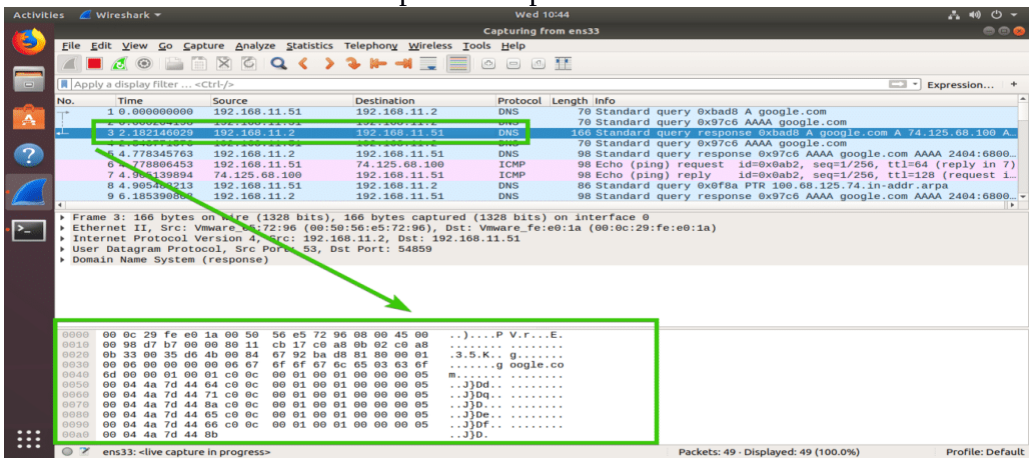
Just ping google.com from the terminal and as you can see, many packets were captured.



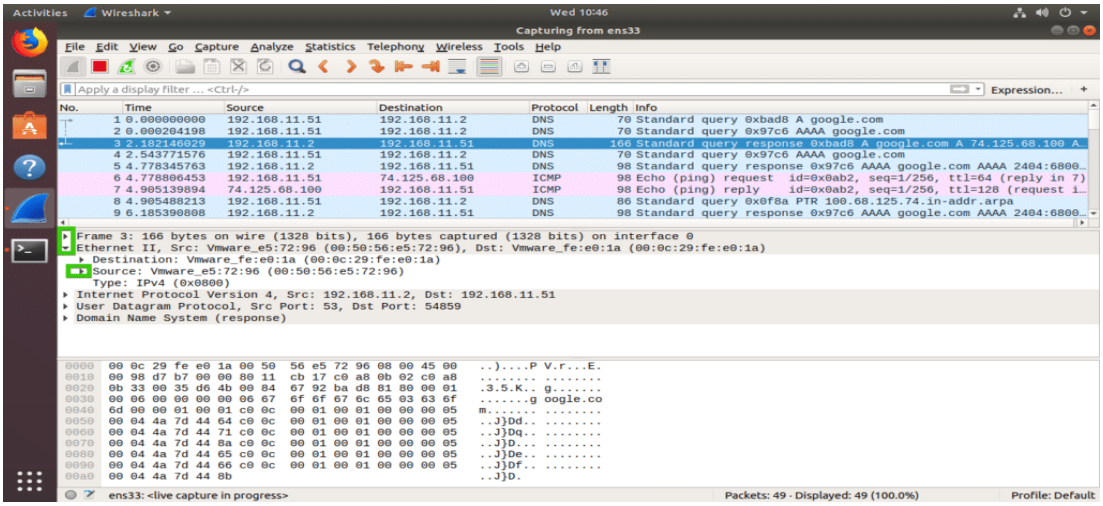
Now you can click on a packet to select it. Selecting a packet would show many information about that packet. As you can see, information about different layers of TCP/IP Protocol is listed.



You can also see the RAW data of that particular packet.



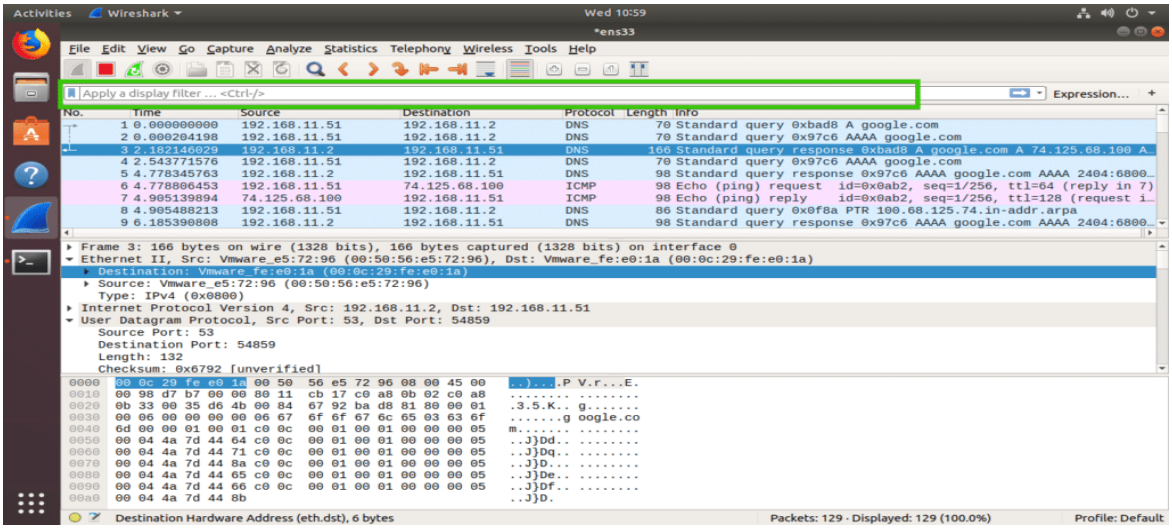
You can also click on the arrows to expand packet data for a particular TCP/IP Protocol Layer.



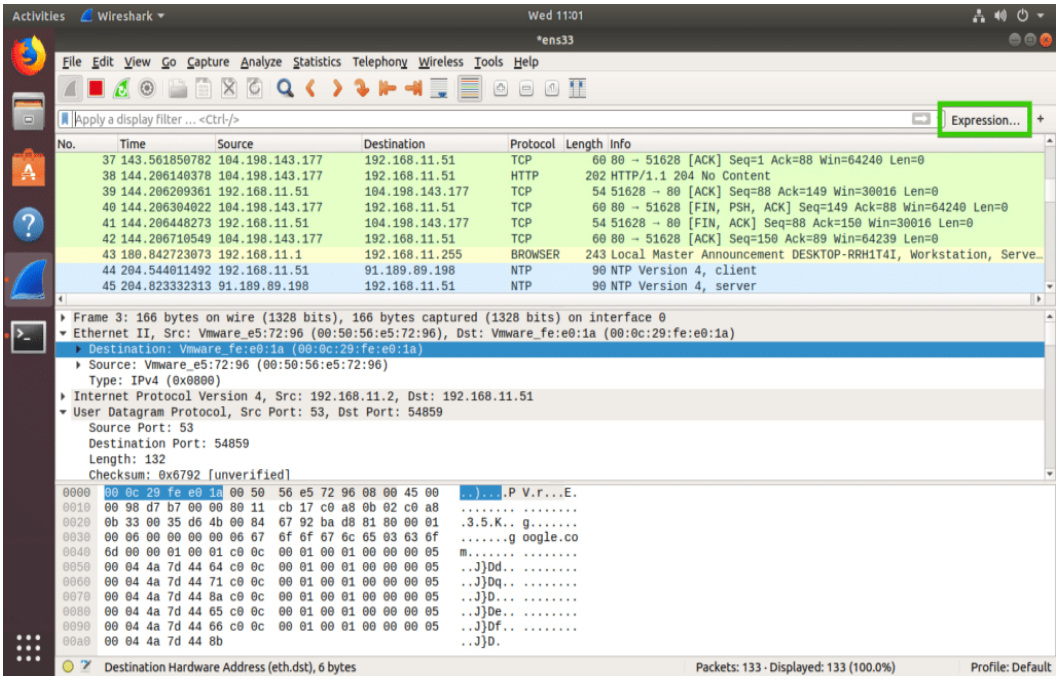
Filtering Packets Using Wireshark:

- On a busy network thousands or millions of packets will be captured each second. So the list will be so long that it will be nearly impossible to scroll through the list and search for certain type of packet.
- The good thing is, in Wireshark, you can filter the packets and see only the packets that you need.

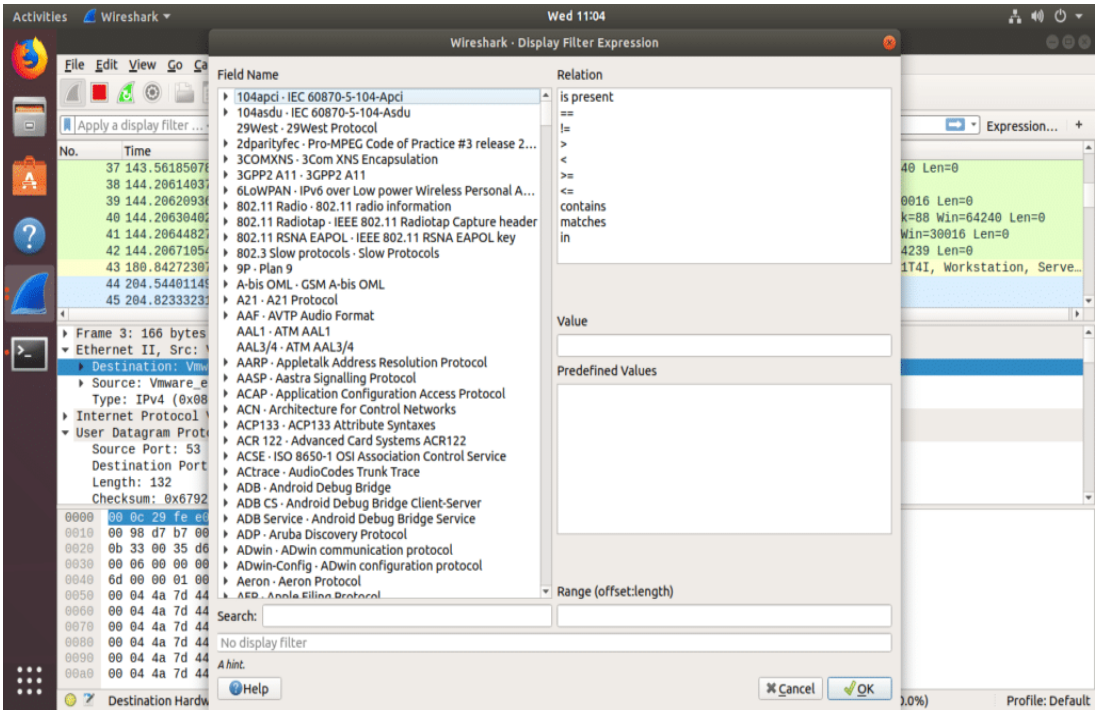
- To filter packets, you can directly type in the filter expression in the textbox as marked in the screenshot below.



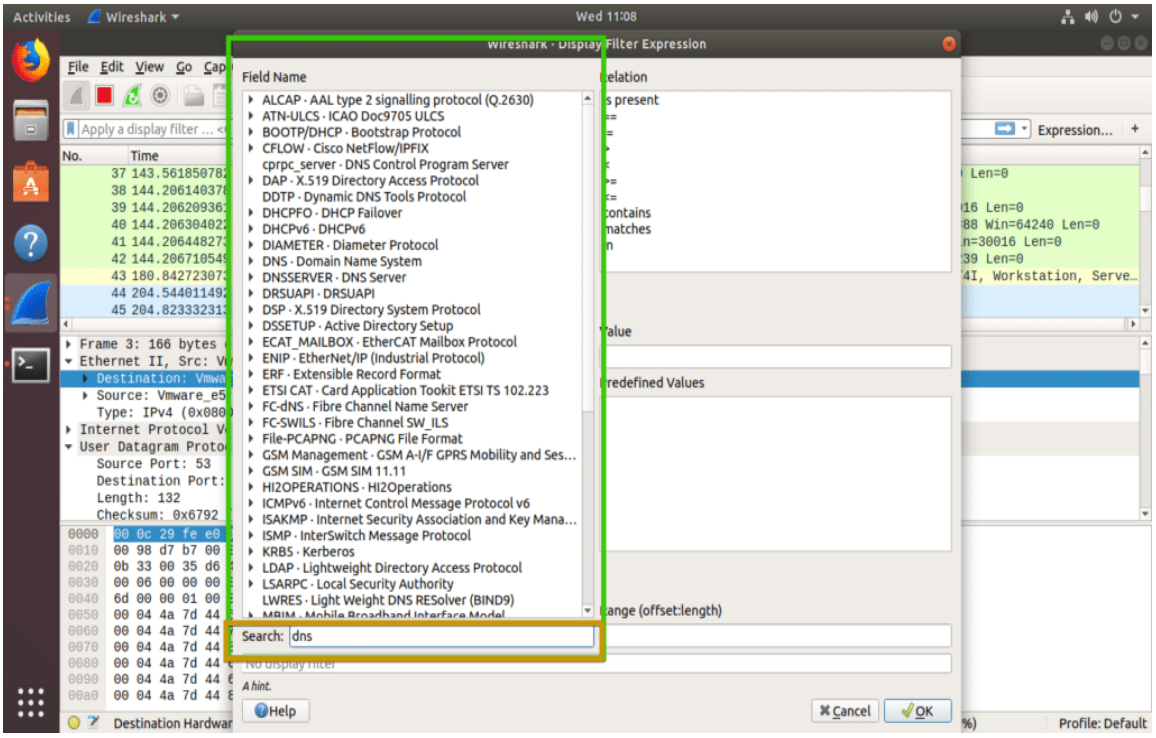
You can also filter packets captured by Wireshark graphically. To do that, click on the **Expression…** button as marked in the screenshot below.



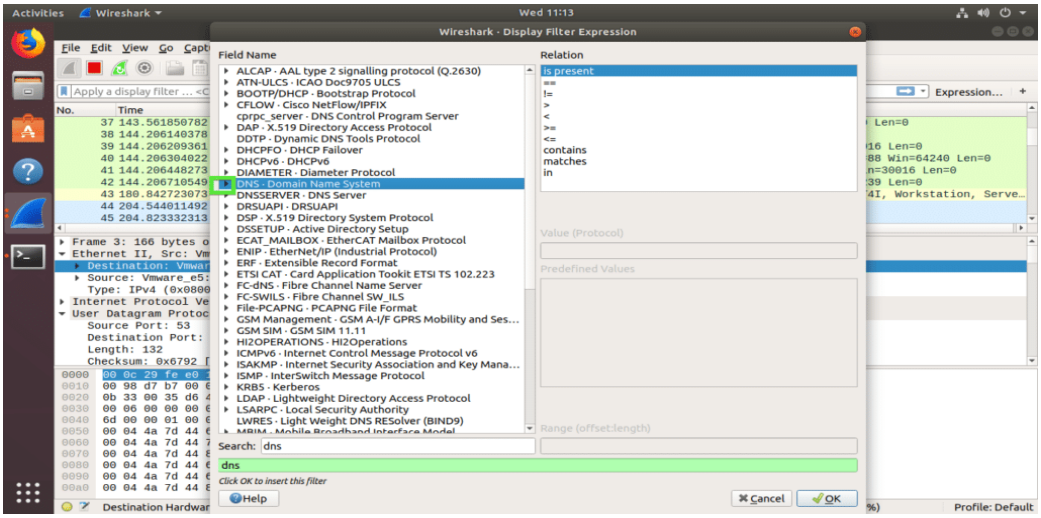
A new window should open as shown in the screenshot below. From here you can create filter expression to search packets very specifically.



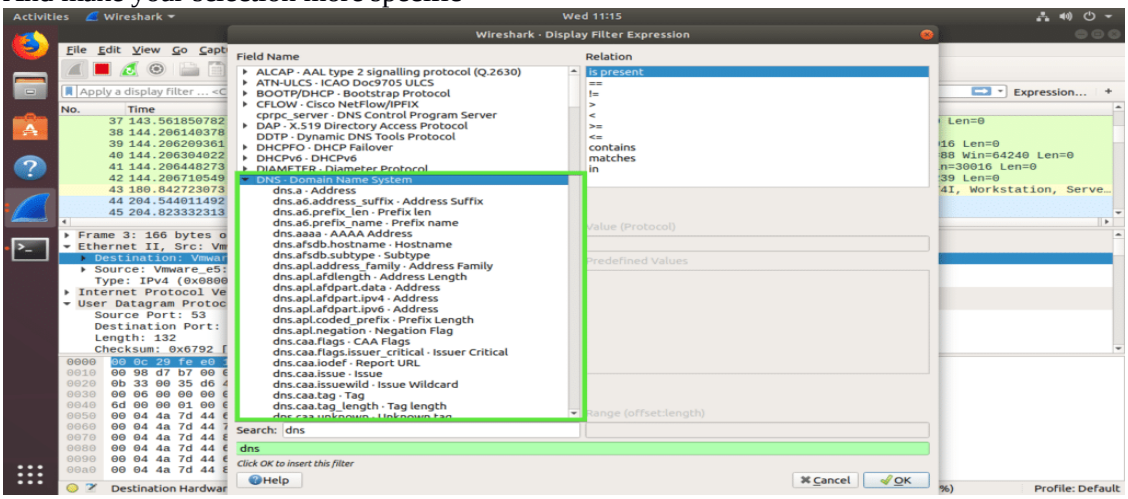
In the **Field Name** section almost all the networking protocols are listed. The list is huge. You can type in what protocol you're looking for in the **Search** textbox and the **Field Name** section would show the ones that matched.



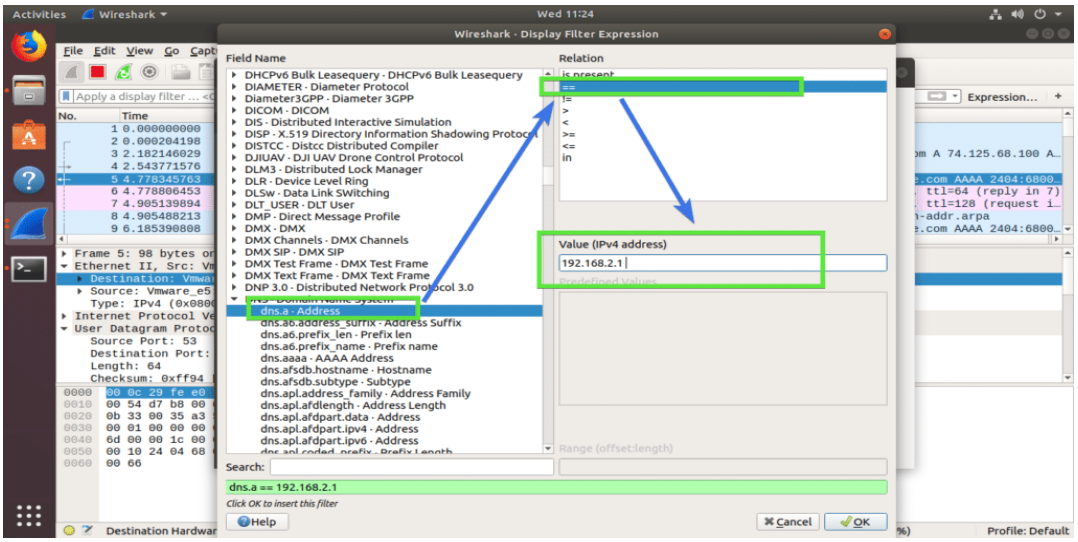
In this article, I am going to filter out all the DNS packets. So I selected **DNSDomain Name System** from the **Field Name** list. You can also click on the **arrow** on any protocol



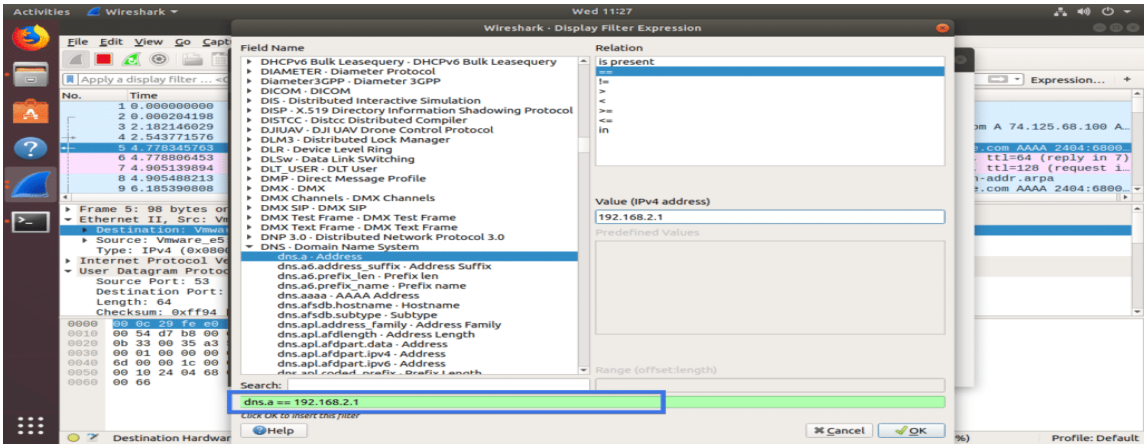
And make your selection more specific



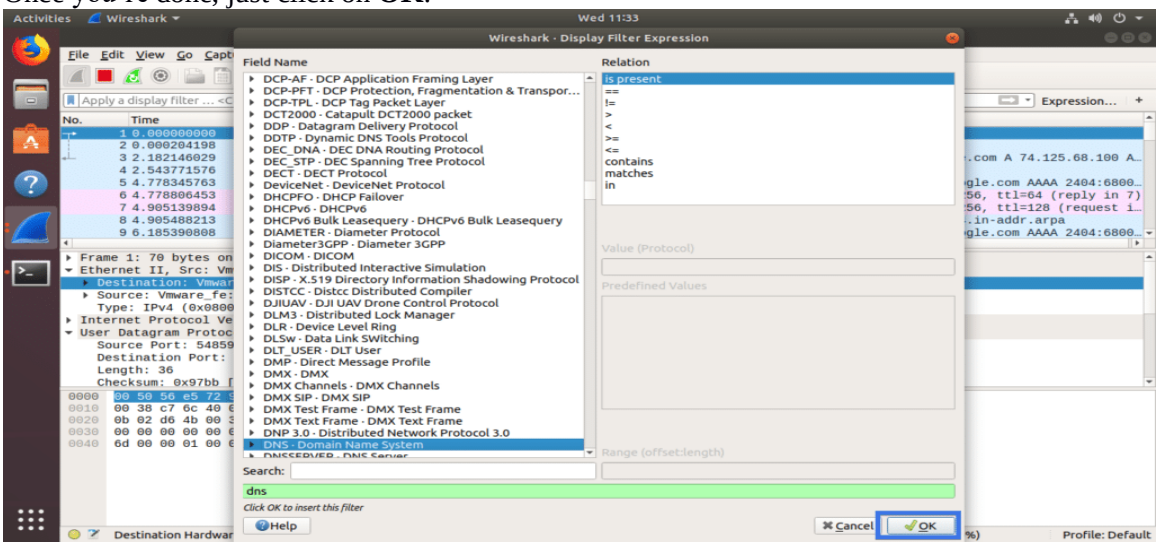
You can also use relational operators to test whether some field is equal to, not equal to, great than or less than some value. I searched for all the **DNS IPv4** address which is equal to **192.168.2.1** as you can see in the screenshot below



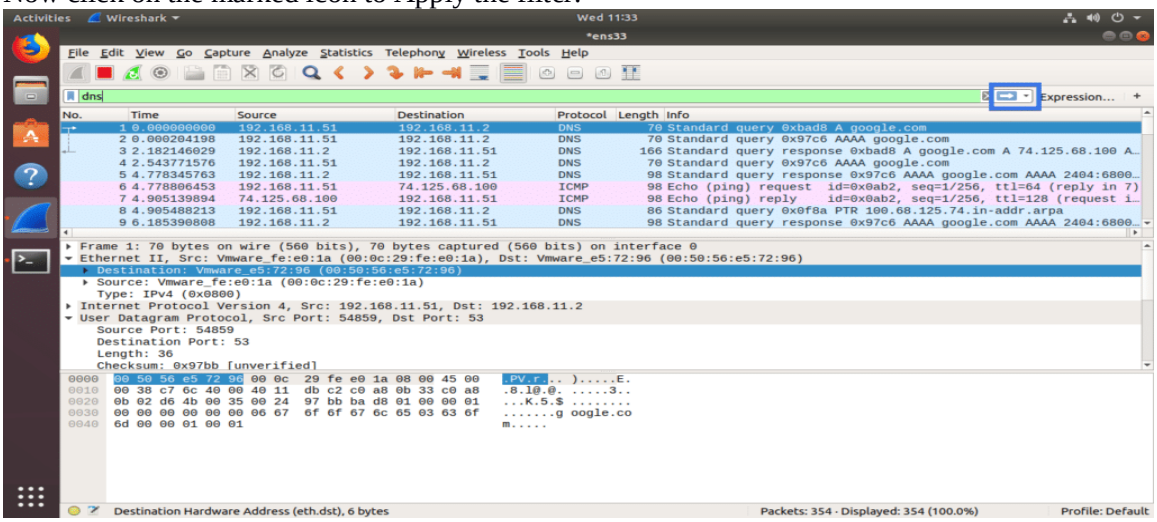
The filter expression is also shown in the marked section of the screenshot below. This is a great way to learn how to write filter expression in Wireshark.



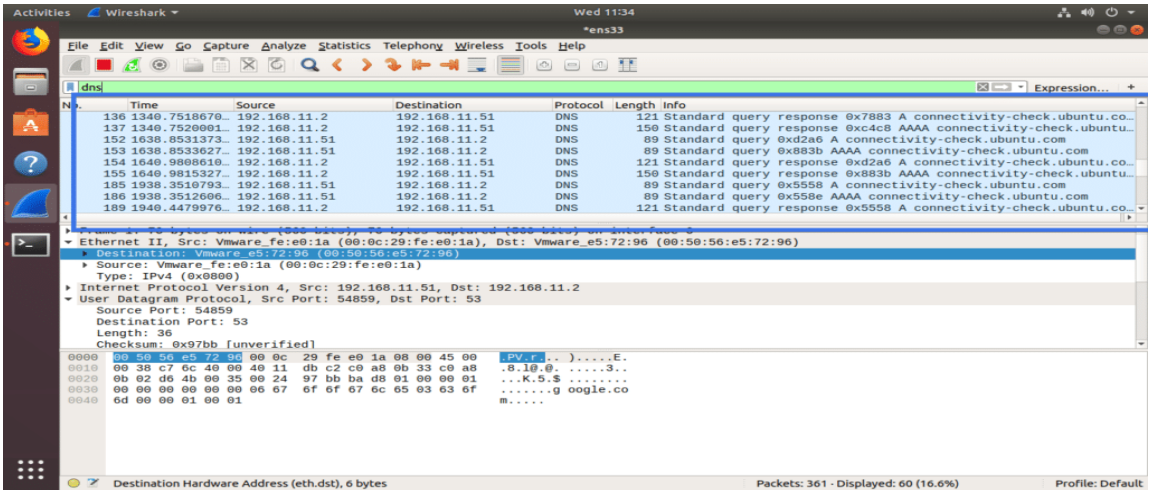
Once you're done, just click on **OK**.



Now click on the marked icon to Apply the filter.

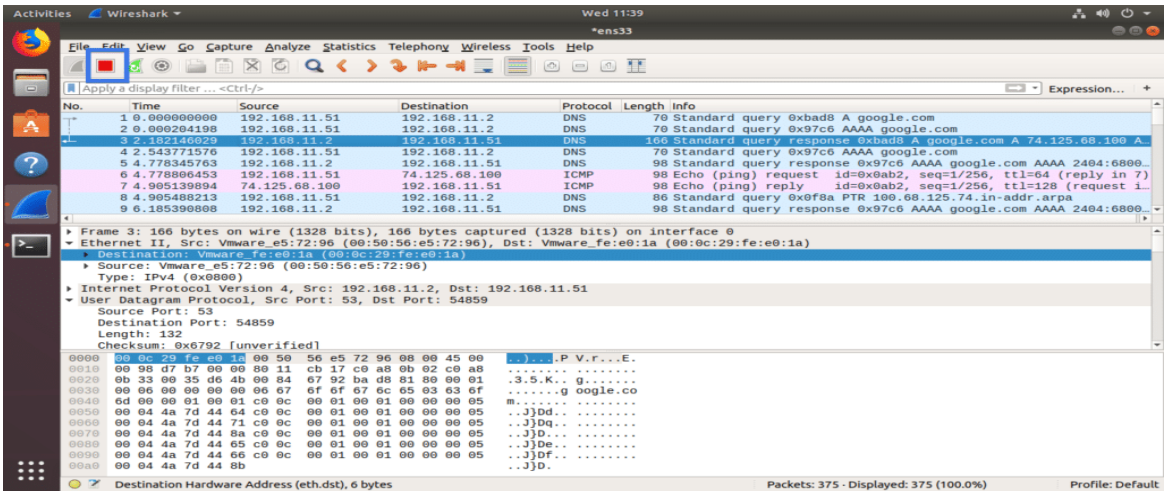


As you can see, only the DNS protocol packets are shown.



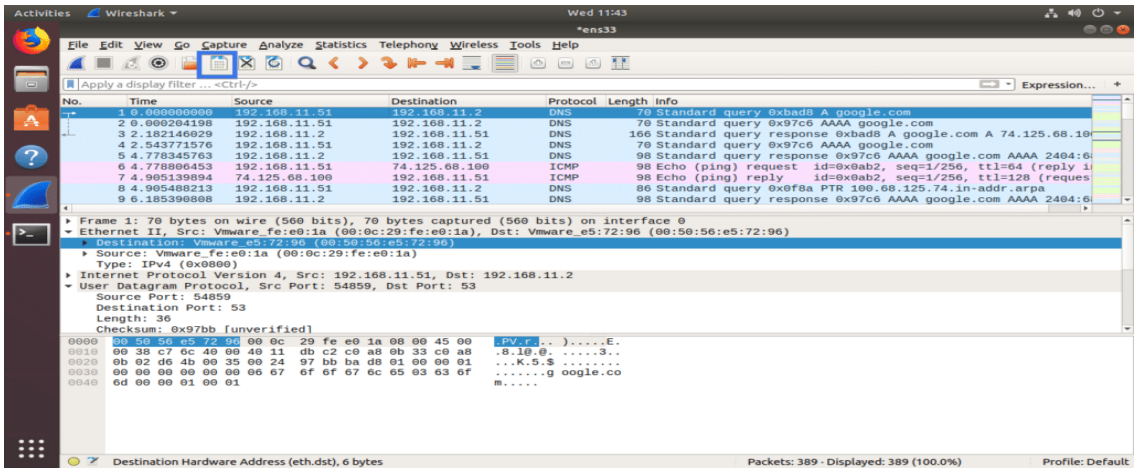
Stopping Packet Capture in Wireshark:

You can click on the red icon as marked in the screenshot below to stop capturing Wireshark packets.

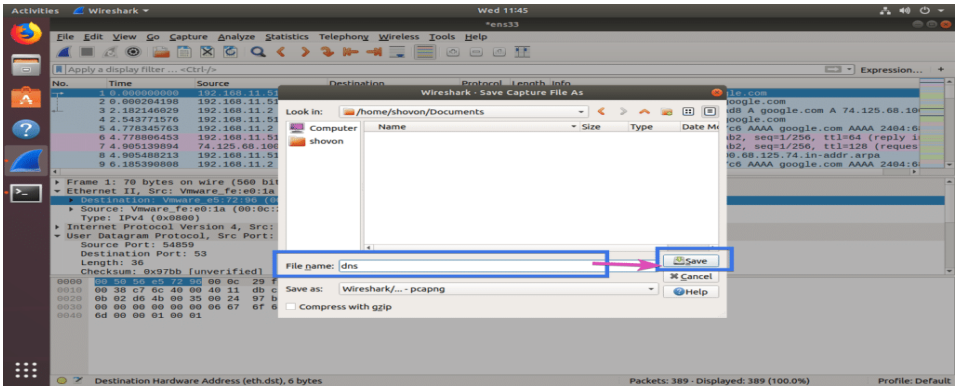


Saving Captured Packets to a File:

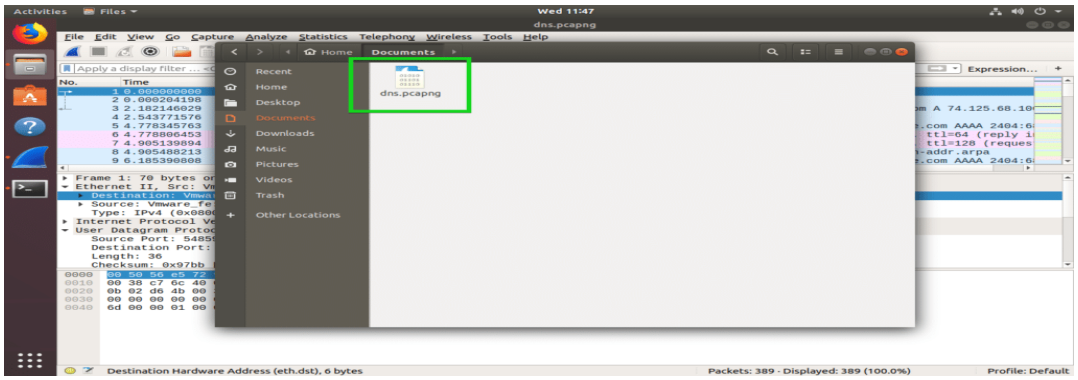
You can click on the marked icon to save captured packets to a file for future use.



Now select a destination folder, type in the file name and click on **Save**.

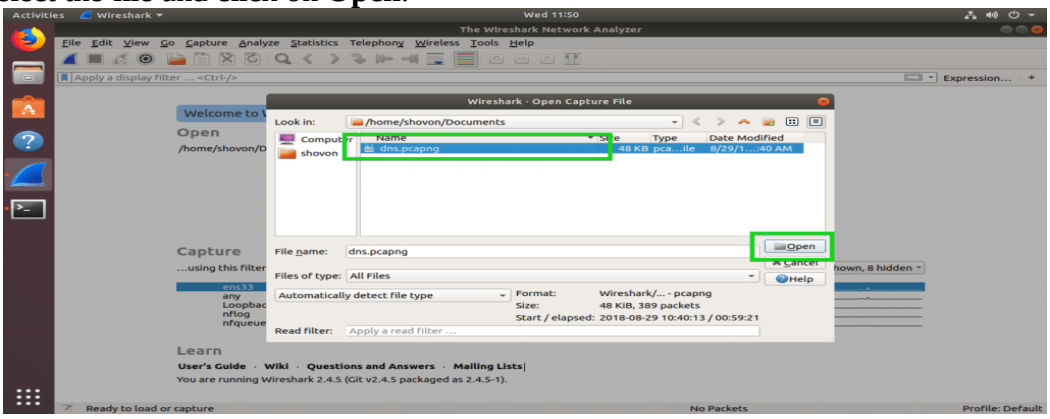


The file should be saved.

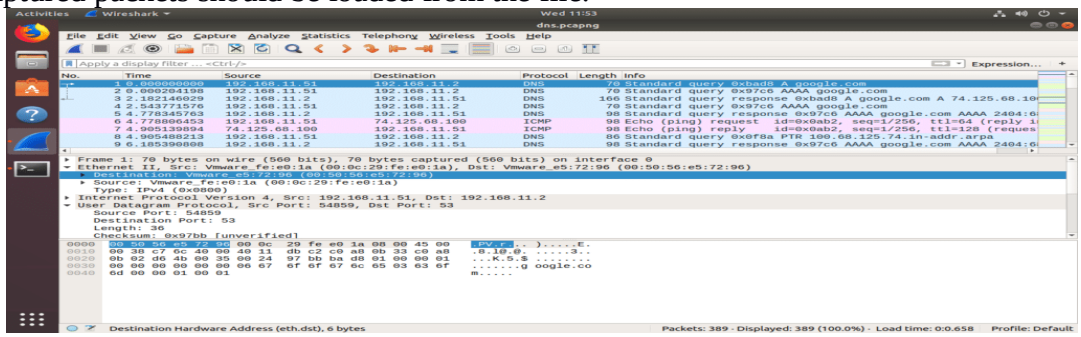


Now you can open and analyze the saved packets anytime. To open the file, go to **File**>**Open** from Wireshark or press **<Ctrl>** + **o**

Then select the file and click on **Open**.



The captured packets should be loaded from the file.



EXPERIMENT-11

Title:

NMAP

Aim:

How to run NMAP SCAN

Theory:

- **Nmap** (*Network Mapper*) is a free and open-sourcenetwork scanner. Nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses.
- Nmap provides a number of features for probing computer networks, including host discovery and service and operating system detection. These features are extensible by scripts that provide more advanced service detection, vulnerability detection,and other features. Nmap can adapt to network conditions including latency and congestion during a scan.

USING NMAP TOOL:

Step 1: Open the Ubuntu command line

We will be using the Ubuntu command line, the Terminal, in order to view the devices connected to our network. Open the Terminal either through the system Dash or the Ctrl+Alt+T shortcut.

Step 2: Install the network scanning tool nmap

When it comes to reliable network scanning, nmap is a tool that you can totally depend on.

Enter the following command as sudo in the Terminal application in order to install the tool.

```
sana@linux:~$ sudo apt install nmap
[sudo] password for sana:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libblas3 liblinear3
Suggested packages:
  liblinear-tools liblinear-dev ndiff
The following NEW packages will be installed:
  libblas3 liblinear3 nmap
0 upgraded, 3 newly installed, 0 to remove and 3 not upgraded.
Need to get 5,353 kB of archives.
After this operation, 24.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

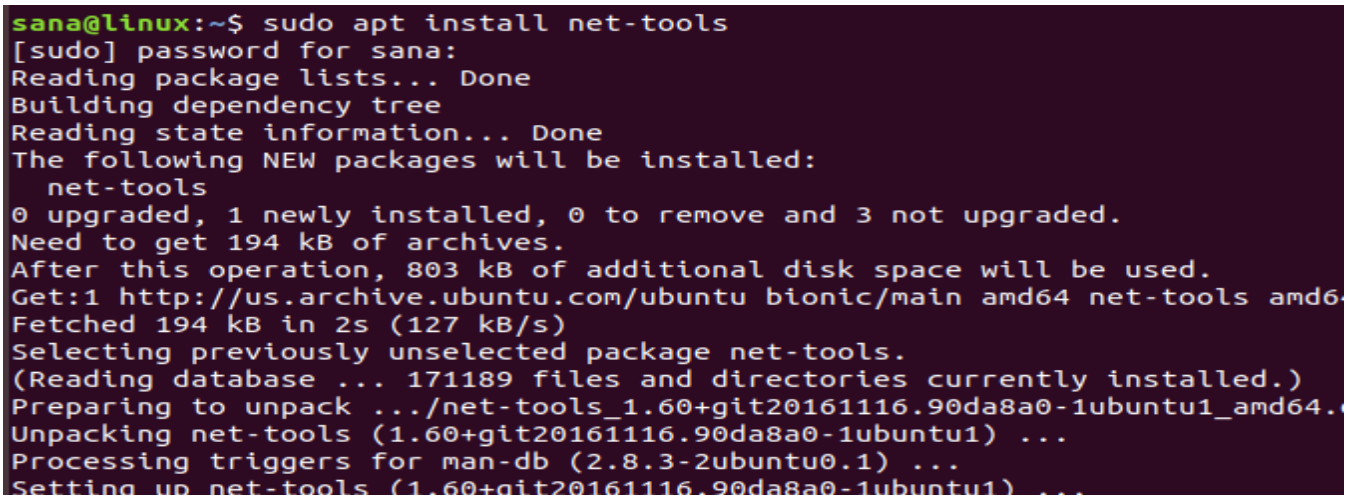
The system will ask you the password for sudo as only an authorized user can install/uninstall and configure software on Ubuntu.

The system will also prompt you with a y/n option to confirm the installation. Please enter Y and hit enter to begin the installation process.

Step 3: Get the IP range/subnet mask of your network

In order to know which devices are connected to your network, you first need to get the IP range or the subnet mask of your network. We will be using the ifconfig command in order to get this IP. In order to run the ifconfig command, we need to have net-tools installed on our Ubuntu. Use the following command in order to install net-tools if you already do not have it installed on your system:

```
$ sudo apt install net-tools
```



The system will prompt you with a y/n option to confirm the installation. Please enter Y and hit enter to begin the installation process.

Once you have the net-tools utility available, run the following command in order to get information about the network(s) your system is connected to:

```
$ ifconfig
```

```
sana@linux:~$ ifconfig
enp37s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.4  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::f23d:232e:4503:9837  prefixlen 64  scopeid 0x20<link>
        ether 10:1f:74:ec:69:c0  txqueuelen 1000  (Ethernet)
        RX packets 113876  bytes 69459909 (69.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 93575  bytes 15186664 (15.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 19802  bytes 1705780 (1.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19802  bytes 1705780 (1.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The highlighted IP from the output indicates that our system is using 192.168.100.0 subnet mask and the range is 255. Thus our network IP range is from 192.168.100.0 to 192.168.100.255

Step 4: Scan network for connected device(s) with nmap

Through the nmap tool, you can scan the report of all devices connected to a network by providing the subnet mask IP as follows:

```
$ nmap -sP 192.168.100.0/24
```

```
sana@linux:~$ nmap -sP 192.168.100.0/24

Starting Nmap 7.60 ( https://nmap.org ) at 2018-11-29 10:32 PKT
Nmap scan report for _gateway (192.168.100.1)
Host is up (0.00063s latency).
Nmap scan report for 192.168.100.2
Host is up (0.086s latency).
Nmap scan report for linux (192.168.100.4)
Host is up (0.00024s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.93 seconds
sana@linux:~$
```

The output shows that there are 3 devices connected on the network; one is the router itself, one is the Linux system I am using on my laptop, and the third one is my phone.

Step 5: Exit the Terminal

Use the following command in order to exit the Terminal application after you are done with extracting the required information:

```
$ exit
```

EXPERIMENT-12

Title:

Operating System Detection

Aim:

Operating System Detection using Nmap

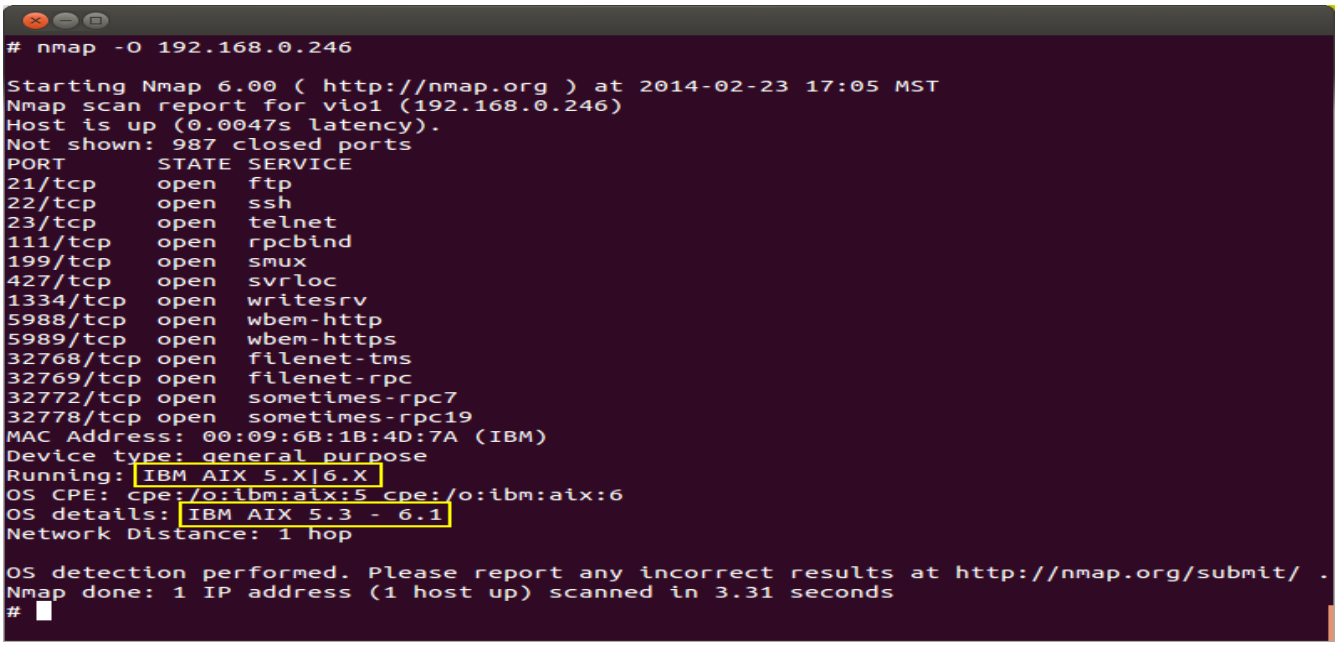
Theory:

- Nmap is one of the most popular tools used for the enumeration of a targeted host. Nmap can use scans that provide the OS, version, and service detection for individual or multiple devices. Detection scans are critical to the enumeration process when conducting penetration testing of a network. It is important to know where vulnerable machines are located on the network so they can be fixed or replaced before they are attacked. Many attackers will use these scans to figure out what payloads would be most effective on a victim's device. The OS scan works by using the TCP/IP stack fingerprinting method. The services scan works by using the Nmap-service-probes database to enumerate details of services running on a targeted host.

Detect OS and services

This is the command to scan and search for the OS (and the OS version) on a host. This command will provide valuable information for the enumeration phase of your network security assessment (if you only want to detect the operating system, type nmap -O 192.168.0.246):

nmap -O 192.168.0.246



EXPERIMENT-13

Title: Network Simulator 2 (NS-2)

Aim: Working with NS2 Simulator introduction

Theory:

1. What is NS2

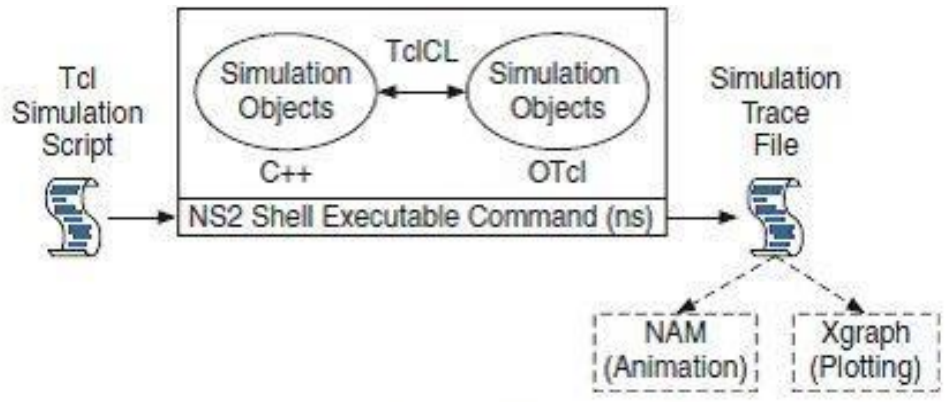
NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks.

2. Features of NS2

1. It is a discrete event simulator for networking research.
2. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.
3. It simulates wired and wireless network.
4. It is primarily Unix based.
5. Uses TCL as its scripting language.
6. Otcl: Object oriented support
7. Tclcl: C++ and otcl linkage
8. Discrete event scheduler

3. Basic Architecture

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL



Basic architecture of NS.

ii) find the number of packets dropped.

STEPS:

Step1: Select the hub icon on the toolbar and drag it onto the working window.

Step2: Select the host icon on the toolbar and drag it onto the working window. Repeat this for another host icon.

Step3: Select the link icon on the toolbar and drag it on the screen from host(node 1) to the hub and again from host(node 2) to the hub. Here the hub acts as node 3 in the point-to-point network. This leads to the creation of the 3-node point-to-point network topology. Save this topology as a .tpl file.

Step4: Double-click on host(node 1), a host dialog box will open up. Click on Node editor and you can see the different layers-interface, ARP, FIFO, MAC, TCPDUMP, Physical layers. Select MAC and then select full-duplex for switches and routers and half duplex for hubs, and in log Statistics, select Number of Drop Packets, Number of Collisions, Throughput of incoming packets and Throughput of outgoing packets. Select FIFO and set the queue size to 50 and press OK. Then click on Add. Another dialog box pops up. Click on the Command box and type the Command according to the following syntax:stg [-t duration(sec)] [-p port number]HostIPaddrand click OK.

Step 5: Double-click on host (node 2), and follow the same step as above with only change in command according to the following syntax:rtg [-t] [-w log] [-p port number]and click OK.

Step 6: Double click on the link between node 1 and the hub to set the bandwidth to some initialvalue say, 10 Mbps. Repeat the same for the other node.

Step 7: Click on the E button (Edit Property) present on the toolbar in order to save the changes made to the topology. Now click on the R button (Run Simulation).
By doing so a user can run/pause/continue/stop/abort/disconnect/reconnect/submit a simulation. No simulation settings can be changed in this mode.

Step 8: Now go to Menu->Simulation->Run. Executing this command will submit he current simulation job to one available simulation server managed by the dispatcher. When the simulation

server is executing, the user will see the time knot at the bottom of the screen move. The time knot reflects the current virtual time (progress) of the simulation case.

Step 9:To start the playback, the user can left-click the start icon(|>) of the time bar located at the bottom. The animation player will then start playing the recorded packet animation

.Step 10: Change the bandwidth say, 9 Mbps, and run the simulationandcomparethetworesults.

Step 11: To view the results, go to the filename. results folder

iii) Find the Number of Packets Dropped due to Congestion

STEPS:

Step1: Click on the subnet icon on the tool bar and then click on the screen of the working window.

Step2:Select the required number of host s and a suitable radius between the host and the switch.

Step 3: In the edit mode, get the IP address of one of the hosts say, host 1 and then for the other host say, host2 set the drop packet and no: of collisions statistics as described in the earlier experiments.

Step 4: Now run the simulation.

Step 5: Now click on any one of the hosts and click on command console and ping the destination node.

iv : the performance with respect to transmission of packets.

STEPS:

Step 1: Connect a host and two WLAN access points to a router.

Step 2: Setup multiple mobile nodes around the two WLAN access points and set the path for each mobile node.

Step 3: Setup a ttcp connection between the mobile nodes and host using the following command:Mobile Host
1ttcp –t –u –s –p 3000

IPAddr Of Receiver Mobile Host 1ttcp –t –u –s –p 4000 IP AddrOf Receiver Host(Receiver)

ttcp –r –u –s –p 3000 ttcp –r –u –s –p 4000

Step 4: Setup the input throughput log at the destination host.

Step 5: To set the transmission range go toMenu->Settings->WLANmobilenode->Showtransmission range.

Step 6: View the results in the filename. results.