Department of CSE (AI & ML)

AM506PC: Computer Networks Lab

Laboratory Manual

R22 - B.Tech. CSE (AI & ML)

III YEAR – I SEM

MALLA REDDY COLLEGE OF ENGINEERING

(Formerly CM Engineering College)

Approved by AICTE & Permanently Affiliated to JNTUH: ISO 9001:2015 Certified Institution

Vision of the Institute

To emerge as a Center of Excellence for producing professionals who shall be the leaders in technology innovation, entrepreneurship, management and in turn contribute for advancement of society and human kind.

Mission of the Institute

- To provide an environment of learning in emerging technologies.
- To nurture a state of art teaching learning process and R&D culture.
- To foster networking with Alumni, Industry, Institutes of repute and other stakeholders for effective interaction.
- To practice and promote high standards of ethical values through societal commitment.

AM506PC: COMPUTER NETWORKS LAB

B.Tech. III Year I Sem. L T P C
0 0 2 1

Course Objectives

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

Course Outcomes

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

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 techniques used in buffers.
10. Wireshark
    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. NS2 Simulator-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

TEXT BOOK:

1. Computer Networks, Andrew S Tanenbaum, David. j. Wetherall, 5th Edition. Pearson Education/PHI.

REFERENCE BOOKS:

1. An Engineering Approach to Computer Networks, S. Keshav, 2nd Edition, Pearson Education.
2. Data Communications and Networking – Behrouz A. Forouzan. 3rd Edition, TMH.

Table of Contents

1.(a) IMPLEMENTATION OF DATA LINK LAYER FRAMING METHOD

BIT STUFFING

AIM

To implement **BIT STUFFING** - the data link layer framing method.

ALGORITHM

1. Read the string (0's and 1's).
2. Add 0 if consecutive five bits containing 1's.
3. Add the flag bits 01111110 in starting and end of the input string.
4. Print the stuffing string.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{
 char ch, arr[50], fs[50]="", t[50]="";
 int count=0, i=0, j=0;
 clrscr();
 printf("Enter the bit srting : \t");
 scanf("%s",arr);
 strcat(fs,"01111110-");
 if(strlen(arr)<5)
 {
  strcat(fs,arr);
  strcat(fs,"-01111110");
  printf("The bits after stuffing are \n %s",fs);
  }
 else
 {
  while((ch=arr[j]) !='\0')
  {
   if(ch=='1')
          ++count;
   else
          count=0;
   t[i++]=ch;
   if(count==5)
   {
    t[i++]='0';
    count=0;
```

```
  }
 j++;
 }
 t[i]='\0';
 strcat(t,"-01111110");
 strcat(fs,t);
 printf("The bits after bit stuffing are\n %s", fs);
 }
getch();
return 0;
}
```

SAMPLE OUTPUT -1

Enter the bit srting : 111111
The bits after bit stuffing are
01111110-1111101-01111110

SAMPLE OUTPUT -2

Enter the bit srting : 111111111111111111111
The bits after bit stuffing are
01111110-11111011111011111011111011-01111110

SAMPLE OUTPUT -3

Enter the bit srting : 111
The bits after stuffing are
01111110-111-01111110

RESULT

Thus the program was executed and verified successfully for **BIT STUFFING**.

1.(b) IMPLEMENTATION OF DATA LINK LAYER FRAMING METHOD

CHARACTER (OR) BYTE STUFFING

AIM

To implement **BYTE STUFFING** or **CHARACTER STUFFING** - the data link layer framing method.

ALGORITHM

1. Read the string (Character string).
2. Convert the input string into capital letters by using ASCII.
3. Add the flag DLESTX (Start of Text) at starting and DLEETX (End of Text) at end of the input string.
4. Add DLE if the input string contains DLE.
5. Print the string after stuffing.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{
 char sdel[]="DLESTX", data[50]="", sdata[50]="";
 int i=0, j=0, k;
 clrscr();
 printf("Enter the message \t\t: ");
 scanf("%s",data);
 for ( k=0; data [k]!='\0'; k++)
   if(data[k]>='a' && data[k]<='z')
      data[k]=data[k]-32;

 printf("Original Message is \t: %s \n",data);
 if(strlen(data)<3)
 {
  strcat(sdel, data);
  strcat(sdel,"DLEETX");
  printf("Message after character stuffing is \t: %s",sdel);
  }
 else
 {
  while(data[i] != '\0')
  {
   if(data[i] == 'D' && data[i+1] == 'L' && data[i+2] == 'E')
   {
    strcat(sdata, "DLEDLE");
    i=i+3;
```

```
  j=j+6;
  continue;
  }
 sdata[j]=data[i];
 j++;
 i++;
}
strcat(sdel, sdata);
strcat(sdel,"DLEETX");
printf("Message after stuffing is \t : %s",sdel);
}
getch();
return 0;
}
```

SAMPLE OUTPUT -1

Enter the message : hai
Original Message is : HAI
Message after stuffing is : DLESTXHAIDLEETX

SAMPLE OUTPUT -2

Enter the message : adle
Original Message is : ADLE
Message after stuffing is : DLESTXADLEDLEDLEETX

SAMPLE OUTPUT -3

Enter the message : adlestxadleetx
Original Message is : ADLESTXADLEETX
Message after stuffing is : DLESTXADLEDLESTXADLEDLEETXDLEETX

RESULT

Thus the program was executed and verified successfully for **CHARACTER STUFFING**.

2. COMPUTATION OF CRC CODE FOR THE POLYNOMIALS CRC-12, CRC-16 and CRC CCIP

AIM

To write a program to compute **CRC code for the polynomials** – CRC-12, CRC-16, CRC-CCIP.

DESCRIPTION

1. **Calculation of CRC at Sender Side**
    - A string of n 0's is appended to the data unit to be transmitted.
    - Here, n is one less than the number of bits in CRC generator.
    - Binary division is performed of the resultant string with the CRC generator.
    - After division, the remainder so obtained is called as **CRC**.
    - It may be noted that CRC also consists of n bits.
2. **Appending CRC to Data Unit**
    - The CRC is obtained after the binary division.
    - The string of n 0's appended to the data unit earlier is replaced by the CRC remainder.
3. **Transmission to Receiver**
    - The newly formed code word (Original data + CRC) is transmitted to the receiver.
4. **Checking at Receiver Side**
    - The transmitted code word is received.
    - The received code word is divided with the same CRC generator.
    - On division, the remainder so obtained is checked.

The following two cases are possible now.

Case-1: Remainder = 0

- Receiver assumes that no error occurred in the data during the transmission.
- Receiver accepts the data.

Case-02: Remainder ≠ 0

- Receiver assumes that some error occurred in the data during the transmission.
- Receiver rejects the data and asks the sender for retransmission.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main(void)
{
 int data[50],div[16],rem[16];
 int datalen, divlen, i,j,k;
 int ch;
 clrscr();
 printf("Enter the data: ");
 i = 0;
 while((ch = fgetc(stdin)) != '\n')
 {
  if(ch == '1')
    data[i] = 1;
  else
    data[i] = 0;
  i++;
  }
 datalen = i;
 printf("\nEnter the divisor: ");
 i = 0;
 while((ch = fgetc(stdin)) != '\n')
 {
  if(ch == '1')
    div[i] = 1;
  else
    div[i] = 0;
  i++;
  }
 divlen = i;
 for(i = datalen ; i < datalen + divlen - 1 ; i++)
    data[i] = 0;
 datalen = datalen + divlen - 1;
 for(i = 0 ; i < divlen ; i++)
    rem[i] = data[i];
 k = divlen-1;
 while(k < datalen)
    if(rem[0] == 1)
    {
     for(i = 0 ; i < divlen ; i++)
        rem[i] = rem[i] ^ div[i];
     }
    else
```

```
  {
   if(k == datalen-1)
      break;
   for(i = 0 ; i < divlen-1 ; i++)
   {
    rem[i] = rem[i+1];
    printf("%d",rem[i]);
   }
  rem[i] = data[++k];
  printf("%d\n",rem[i]);
 }

j=1;
for(i = datalen - divlen + 1 ; i < datalen ; i++)
     data[i] = rem[j++];
printf("\nThe data to be sent is\n");
for(i = 0 ; i < datalen ; i++)
     printf("%d",data[i]);
getch();
return 0;
}
```

SAMPLE OUTPUT

Enter the data: 10101011

Enter the divisor: 101

000

001

010

101

001

010

100

The data to be sent is

1010101101

RESULT

Thus the program was executed and verified successfully for **CRC code for the polynomials**.

3.(a) FLOW CONTROL USING THE SLIDING WINDOW PROTOCOL

AIM

To develop a simple date link layer that performs the **flow control using the sliding window protocol**.

DESCRIPTION

1. Sliding window protocol is applied on the Data Link Layer of OSI model. At data link layer data is in the form of frames.
2. Both sender and receiver agrees on some window size. If window size=w then after sending w frames sender waits for the acknowledgement (ack) of the first frame.
3. As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender.
4. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frames are properly received.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int w,i,f,frames[50];
   clrscr();
   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");
  getch();
  return 0;
 }
```

SAMPLE OUTPUT

Enter window size: 3
Enter number of frames to transmit: 5
Enter 5 frames: 21
25
45
65
29

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

21 25 45
Acknowledgement of above frames sent is received by sender

65 29
Acknowledgement of above frames sent is received by sender

RESULT

Thus the program was executed and verified successfully for **flow control using the sliding window protocol**.

3.(b) LOSS RECOVERY USING THE GO-BACK-N MECHANISM

AIM

To develop a simple date link layer that performs the **loss recovery using the Go-Back-N Mechanism**.

DESCRIPTION

1. The sender can transmit N frames before receiving the ACK frame.
2. The size of the send sliding window is N.
3. The copy of sent data is maintained in the sent buffer of the sender until all the sent packets are acknowledged.
4. If the timeout timer runs out then the sender will resend all the packets.
5. Once the data get acknowledged by the receiver then that particular data will be removed from the buffer.
6. Whenever a valid acknowledgement arrives then the send window can slide one or more slots.

PROGRAM

```
#include<stdio.h>
#include<contio.h>
int main()
{
 int windowsize, sent=0, ack, i;
 clrscr();
 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 \t");
        scanf("%d",&ack);




if(ack == windowsize)
        {
```

```
                printf("All the frames transmitted successfully ");
                break;
            }
        else
                sent = ack;
        }
 getch();
 return 0;
 }
```

SAMPLE OUTPUT

```
enter window size
5
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.

Please enter the last Acknowledgement received      4
Frame 4 has been transmitted.

Please enter the last Acknowledgement received      5
All the frames transmitted successfully
```

RESULT

Thus the program was executed and verified successfully for **Loss recovery using the go-back-n mechanism**.

4. DIJKSTRA'S ALGORITHM TO COMPUTE SHORTEST PATH THROUGH A NETWORK

AIM

To implement Dijkstra's algorithm to compute shortest path through a network.

DESCRIPTION

1. Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
2. The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
3. Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
4. The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

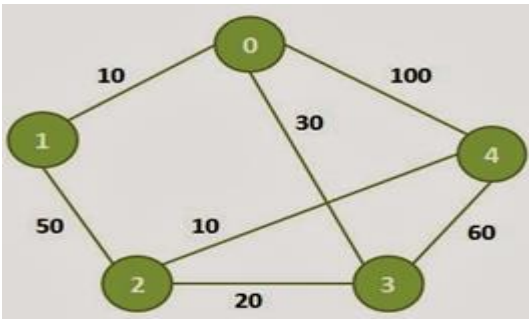
PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra (int G[MAX][MAX], int n, int startnode);

int main()
{
 int G[MAX][MAX], i, j, n, u;
 clrscr();
 printf("Enter no. of vertices :\t");
 scanf("%d",&n);
 printf("\nEnter the adjacency matrix:\n");
 for(i=0; i<n; i++)
 for(j=0; j<n; j++)
   scanf("%d",&G[i][j]);
 printf("\nEnter the starting node :\t");
 scanf("%d",&u);
 dijkstra(G,n,u);
 getch();
 return 0;
 }
void dijkstra(int G[MAX][MAX], int n, int startnode)
{
```

```
int cost[MAX][MAX], distance[MAX], pred[MAX];
int visited[MAX], count, mindistance, nextnode, i, j;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0; i<n; i++)
for(j=0; j<n; j++)
  if(G[i][j] == 0)
     cost[i][j] = INFINITY;
  else
     cost[i][j] = G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0; i<n; i++)
{
  distance[i] = cost[startnode][i];
  pred[i] = startnode;
  visited[i] = 0;
 }
distance[startnode] = 0;
visited[startnode] = 1;
count = 1;
while(count < n-1)
{
 mindistance = INFINITY;
 //nextnode gives the node at minimum distance
 for(i=0; i<n; i++)
  if(distance[i] < mindistance && !visited[i])
  {
   mindistance = distance[i];
   nextnode = i;
  }

//check if a better path exists through nextnode
 visited[nextnode] = 1;
 for(i=0; i<n; i++)
  if(!visited[i])
     if(mindistance + cost[nextnode][i] < distance[i])
     {
      distance[i] = mindistance + cost[nextnode][i];
      pred[i] = nextnode;
     }
 count++;
}
```

```
//print the path and distance of each node
for(i=0; i<n; i++)
  if(i != startnode)
  {
   printf("\n Distance of node%d = %d", i, distance[i]);
   printf("\n\t Path = %d",i);
   j = i;
   do
   {
     j = pred[j];
     printf(" <-- %d",j);
    }while(j != startnode);
  }
}
```

SAMPLE OUTPUT




```
Enter no. of vertices : 5
Enter the adjacency matrix:
0   10  0   30  100
10  0   50  0   0
0   50  0   20  10
30  0   20  0   60
100 0   10  60  0
Enter the starting node :        0
 Distance of node1 = 10
        Path = 1 <-- 0
 Distance of node2 = 50
        Path = 2 <-- 3 <-- 0
 Distance of node3 = 30
        Path = 3 <-- 0
 Distance of node4 = 60
        Path = 4 <-- 2 <-- 3 <-- 0
```

RESULT

Thus the program was executed and verified successfully for shortest path through a network using Dijkstra's algorithm.

5. BROADCAST TREE FOR THE SUBNET

AIM

To obtain broadcast tree for the subnet of host.

DESCRIPTION

- A router creates a data packet and then sends it to each host one by one. In this case, the router creates multiple copies of single data packet with different destination addresses.
- All packets are sent as unicast but because they are sent to all, it simulates as if router is broadcasting.
- This method consumes lots of bandwidth and router must destination address of each node.
- Secondly, when router receives a packet that is to be broadcasted, it simply floods those packets out of all interfaces. All routers are configured in the same way.
- This method is easy on router's CPU but may cause the problem of duplicate packets received from peer routers.

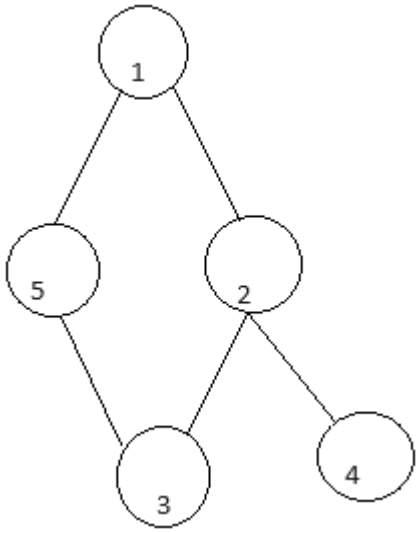
PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a[10][10],n;

void main()
{
 int i,j,root;
 clrscr();
 printf("Enter no.of nodes : ");
 scanf("%d",&n);
 printf("Enter adjacent matrix \n");
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
{
        printf("Enter connecting of %d --> %d :: ",i,j);
        scanf("%d",&a[i][j]);
 }

 printf("Enter root node : ");
 scanf("%d",&root);
 adj(root);
 getch();
 }
```

```
adj(int k)
{
 int i,j;
 printf("Adjacent node of root node ::\n");
 printf("%d\n",k);
 for(j=1;j<=n;j++)
{
  if(a[k][j]==1 || a[j][k]==1)
          printf("%d \t",j);
 }
printf("\n");
for(i=1;i<=n;i++)
{
  if((a[k][j]==0) && (a[i][k]==0) && (i!=k))
          printf("%d \t",i);
 }
}
```

SAMPLE OUTPUT

```
Enter no.of nodes : 5
Enter adjacent matrix
Enter connecting of 1 --> 1 :: 0
Enter connecting of 1 --> 2 :: 1
Enter connecting of 1 --> 3 :: 0
Enter connecting of 1 --> 4 :: 0
Enter connecting of 1 --> 5 :: 1
Enter connecting of 2 --> 1 :: 1
Enter connecting of 2 --> 2 :: 0
Enter connecting of 2 --> 3 :: 1
Enter connecting of 2 --> 4 :: 1
Enter connecting of 2 --> 5 :: 0
Enter connecting of 3 --> 1 :: 0
Enter connecting of 3 --> 2 :: 1
Enter connecting of 3 --> 3 :: 0
Enter connecting of 3 --> 4 :: 0
Enter connecting of 3 --> 5 :: 1
Enter connecting of 4 --> 1 :: 0
Enter connecting of 4 --> 2 :: 1
Enter connecting of 4 --> 3 :: 0
Enter connecting of 4 --> 4 :: 0
Enter connecting of 4 --> 5 :: 0
Enter connecting of 5 --> 1 :: 1
Enter connecting of 5 --> 2 :: 0
Enter connecting of 5 --> 3 :: 1
Enter connecting of 5 --> 4 :: 0
Enter connecting of 5 --> 5 :: 0
Enter root node : 2
Adjacent node of root node ::
2
1       3       4
5
```



RESULT

Thus the program was executed and verified successfully for obtaining broadcast tree for the subnet of host.

6. IMPLEMENTATION OF DISTANCE VECTOR ROUTING ALGORITHM FOR OBTAINING ROUTING TABLES AT EACH NODE

AIM

To implement the distance vector routing algorithm for obtaining routing tables at each node.

DESCRIPTION

Let $d_x(y)$ be the cost of the least-cost path from node x to node y. The least costs are related by Bellman-Ford equation,

$$d_x(y) = \min_v\{c(x,v) + d_v(y)\}$$

Where the minv is the equation taken for all x neighbors. After traveling from x to v, if we consider the least-cost path from v to y, the path cost will be $c(x,v)+d_v(y)$. The least cost from x to y is the minimum of $c(x,v)+d_v(y)$ taken over all neighbors.

With the Distance Vector Routing algorithm, the node x contains the following routing information:

- For each neighbor v, the cost c(x,v) is the path cost from x to directly attached neighbor, v.
- The distance vector x, i.e., $D_x = [D_x(y) : y \text{ in } N]$, containing its cost to all destinations, y, in N.
- The distance vector of each of its neighbors, i.e., $D_v = [D_v(y) : y \text{ in } N]$ for each neighbor v of x.

Distance vector routing is an asynchronous algorithm in which node x sends the copy of its distance vector to all its neighbors. When node x receives the new distance vector from one of its neighboring vector, v, it saves the distance vector of v and uses the Bellman-Ford equation to update its own distance vector. The equation is given below:

$$d_x(y) = \min_v\{ c(x,v) + d_v(y)\} \quad \text{for each node y in N}$$

The node x has updated its own distance vector table by using the above equation and sends its updated table to all its neighbors so that they can update their own distance vectors.

PROGRAM

```
#include<stdio.h>
#include<conio.h>

struct node
{
 unsigned dist[20];
 unsigned from[20];
 }rt[10];

int main()
{
 int dmat[20][20];
 int n,i,j,k,count=0;
 clrscr();
 printf("\nEnter the number of nodes : ");
 scanf("%d",&n);
 printf("\nEnter 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++)
{
 printf("\n\nState value for router %d is \n",i+1);
 for(j=0;j<n;j++)
 {
  printf("\t\nnode %d via %d Distance is %d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
  }
 }
printf("\n\n");
getch();
return 1;
}
```

SAMPLE OUTPUT

```
Enter the number of nodes : 3
Enter the cost matrix :
0   2   4
2   0   5
4   5   0
State value for router 1 is

node 1 via 1 Distance is 0
node 2 via 2 Distance is 2
node 3 via 3 Distance is 4

State value for router 2 is

node 1 via 1 Distance is 2
node 2 via 2 Distance is 0
node 3 via 3 Distance is 5

State value for router 3 is

node 1 via 1 Distance is 4
node 2 via 2 Distance is 5
node 3 via 3 Distance is 0
```

RESULT

Thus the program was executed and verified successfully for distance vector routing algorithm for obtaining routing tables at each node.

7. IMPLEMENTATION OF DATA ENCRYPTION AND DATA DECRYPTION

AIM

To implement encryption of data and decryption of data.

ALGORITHM

1. Start the program.
2. Read the input string to encrypt.
3. The key for encryption is 3 that is added to ASCII value and the encrypted message displayed.
4. The key for encryption is 3 that is subtracted to ASCII value for the decryption and the decrypted message displayed.
5. Stop the program.

PROGRAM

```
#include <stdio.h>
#include<conio.h>
int main()
{
  int i, x;
  char str[100];
  clrscr();
  printf("\nPlease enter a string:\t");
  gets(str);

  do
  {
   printf("\nPlease choose following options:\n");
   printf("1 = Encrypt the string.\n");
   printf("2 = Decrypt the string.\n");
   scanf("%d", &x);

   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;

  case 3:
    exit(0);

  default:
    printf("\nError\n");
  }
 } while(x==1 || x==2);
 getch();
return 0;
}
```

SAMPLE OUTPUT

```
Please enter a string:   welcome
Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
1
Encrypted string: zhofrph

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

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
6
Error
```

RESULT

Thus the program was executed and verified successfully for encryption and decryption.

8. LEAKY BUCKET ALGORITHM FOR CONGESTION CONTROL

AIM

To implement Leaky Bucket Algorithm for Congestion Control.

DESCRIPTION

Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue.

If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded.

This arrangement can be built into the hardware interface or simulated by the host operating system.

The host is allowed to put one packet per clock tick onto the network.

This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define MAX 25
#define min(x,y) ((x)<(y)? (x):(y))

int main()
{
   int cap, oprt,nsec,cont,i=0,inp[MAX],ch;
   clrscr();
   printf("Enter the bucket size:\n");
   scanf("%d",&cap);
   printf("Enter the accepted rate:\n");
   scanf("%d",&oprt);
   do
   {
      printf("Enter the number of packets entering at %d seconds: ",i+1);
      scanf("%d",&inp[i]);
      i++;
      printf("Enter 1 to insert packets or 0 to quit\n");
      scanf("%d",&ch);
    }while(ch);


   nsec=i;
   printf("\n seconds  packets received    packets sent    packets left in bucket\n");
   cont=0;

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

```
for( ; cont != 0; i++)
  {

    if(cont > cap)
       cont = cap;
    printf("  %d ",(i+1));
    printf("\t\t 0 ");
    printf("\t\t\t\t\t %d ",min(cont,oprt));
    cont=cont - min(cont,oprt);
    printf("\t\t\t\t\t %d \n",cont);
   }
  getch();
  return(0);
}
```

SAMPLE OUTPUT

```
Enter the bucket size:
20
Enter the accepted rate:
5
Enter the number of packets entering at 1 seconds: 25
Enter 1 to insert packets or 0 to quit
1
Enter the number of packets entering at 2 seconds: 5
Enter 1 to insert packets or 0 to quit
0
seconds        packets received    packets sent    packets left in bucket
 1              25                  5               15
 2              5                   5               15
 3              0                   5               10
 4              0                   5               5
 5              0                   5               0
```

RESULT

Thus the program was executed and verified successfully for congestion control using leaky bucket algorithm.

9. FRAME SORTING TECHNIQUE USED IN BUFFERS

AIM

To implement Frame sorting technique used in buffers.

DESCRIPTION

A datagram may be fragmented several times before it reached the final destination and also, the datagrams referred to as (frames in Data link layer) may arrives out of order at destination.

Hence sorting of frames need to be done at the destination to recover the original data.

PROGRAM

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
#include <string.h>
struct frame
{
   int sno;
   char msg[15];
   int flag;
};

int main()
{
   int i,j,n,r,k;
   clrscr();
   printf("enter no of frames\n");
   scanf("%d",&n);
   struct frame fr[n];
   int s[n];
   for(i=0;i<n;i++)
   {
      s[i]=-1;
      fr[i].sno=-1;
    }
   printf("enter the message \n");
   for(i=0;i<n;i++)
   {
      scanf("%s",fr[i].msg);
      fr[i].sno=i;
    }
   for(j=0;j<n;j++)
   {
      r=rand()%n;
```

```
  if(s[r]==-1)
  {
     fr[j].flag=r;
     s[r]=1;
   }
  else if(s[r]==1)
  {
     for(k=0;k<n;k++)
     {
      r=k;
      if(s[r]==-1)
      {
       fr[j].flag=r;
       s[r]=1;
       break;
      }
     }
  }
}

printf("\narrived frame are:");
printf("\n sno \t msg  \n");
for(i=0;i<n;i++)
{
 for(j=0;j<n;j++)
   if(fr[j].flag==i)
   {
     printf("%d\t%s",fr[j].sno,fr[j].msg);
     printf("\n");
     }
 }

for(i=0;i<n;i++)
{
 for(j=0;j<n-1;j++)
 {
    if(fr[j].sno>fr[j+1].sno)
    {
      struct frame temp;
      temp=fr[j];
      fr[j]=fr[j+1];
      fr[j+1]=temp;
      }
  }
```

```
    }
    printf("\nafter sorting arrived frames are:");
    printf("\n sno \t msg \n");
    for(i=0;i<n;i++)
    {
     printf("%d\t%s",fr[i].sno,fr[i].msg);
     printf("\n");
     }
    getch();
    return 0;
   }
```

SAMPLE OUTPUT

```
enter no of frames
5
enter the message
hai
hello
dear
welcome
morning

arrived frame are:
 sno      msg
3         welcome
1         hello
2         dear
0         hai
4         morning

after sorting arrived frames are
 sno      msg
0         hai
1         hello
2         dear
3         welcome
4         morning
```

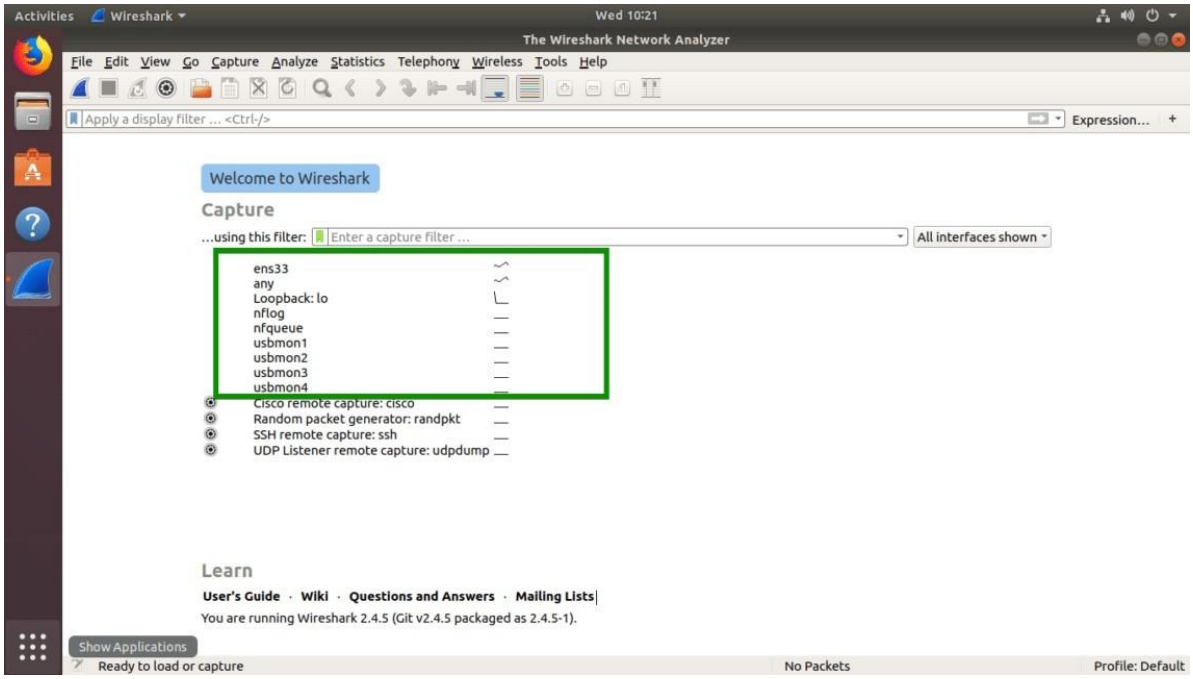
RESULT

Thus the program was executed and verified successfully for frame sorting technique.

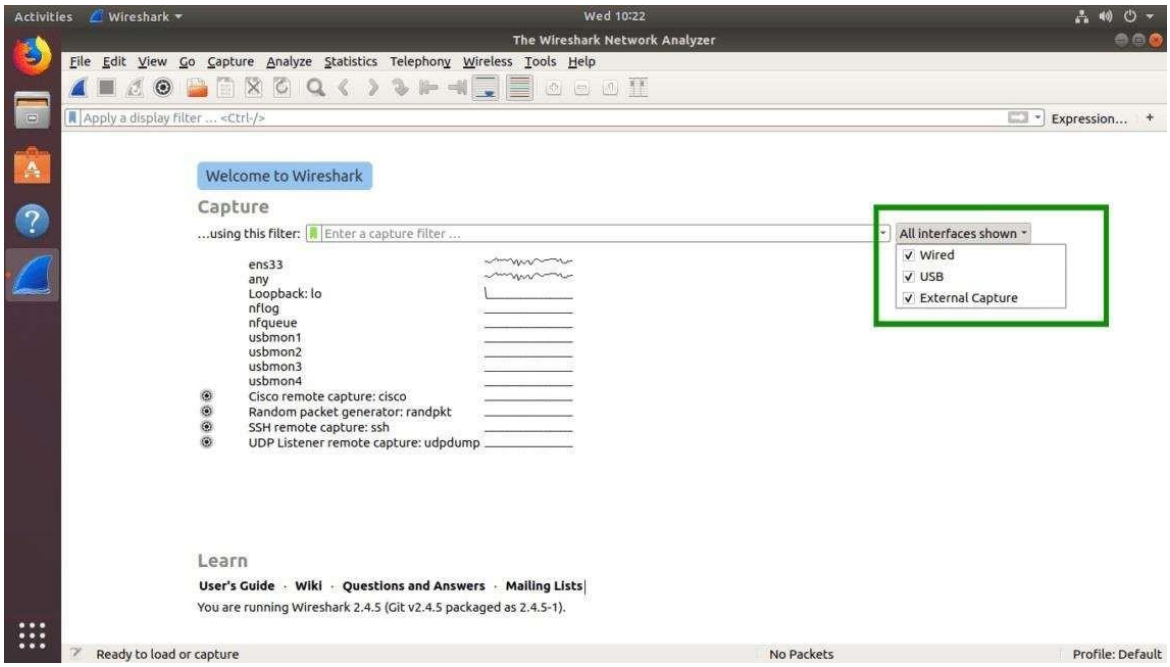
10. Wire shark

Packet Capture Using Wire shark.

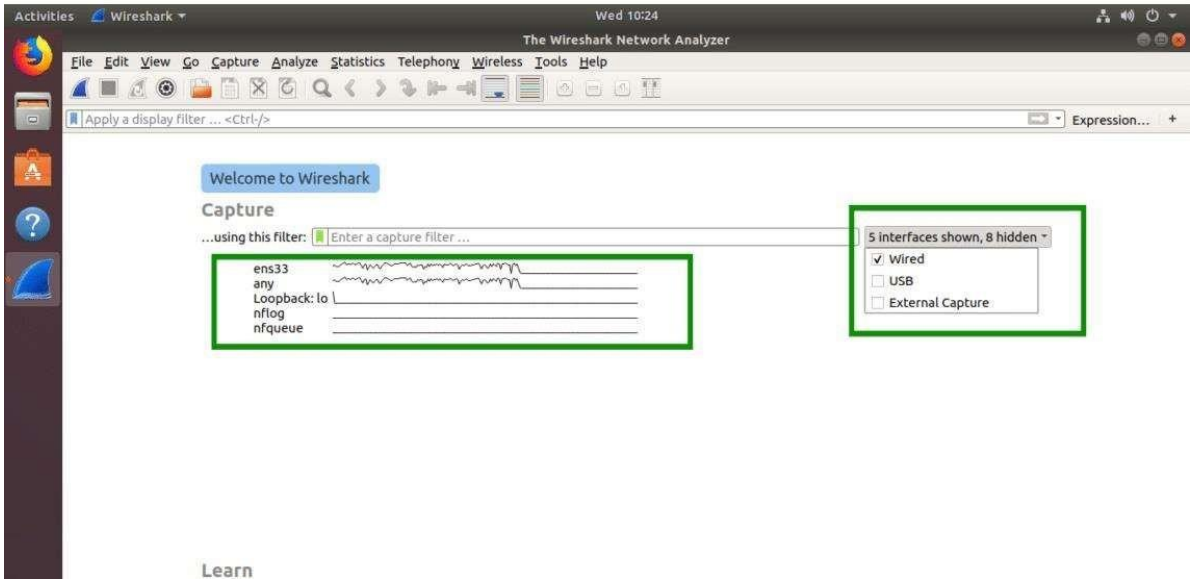
When you start Wireshark, you will see a list of interfaces that you can capture packets to and from.



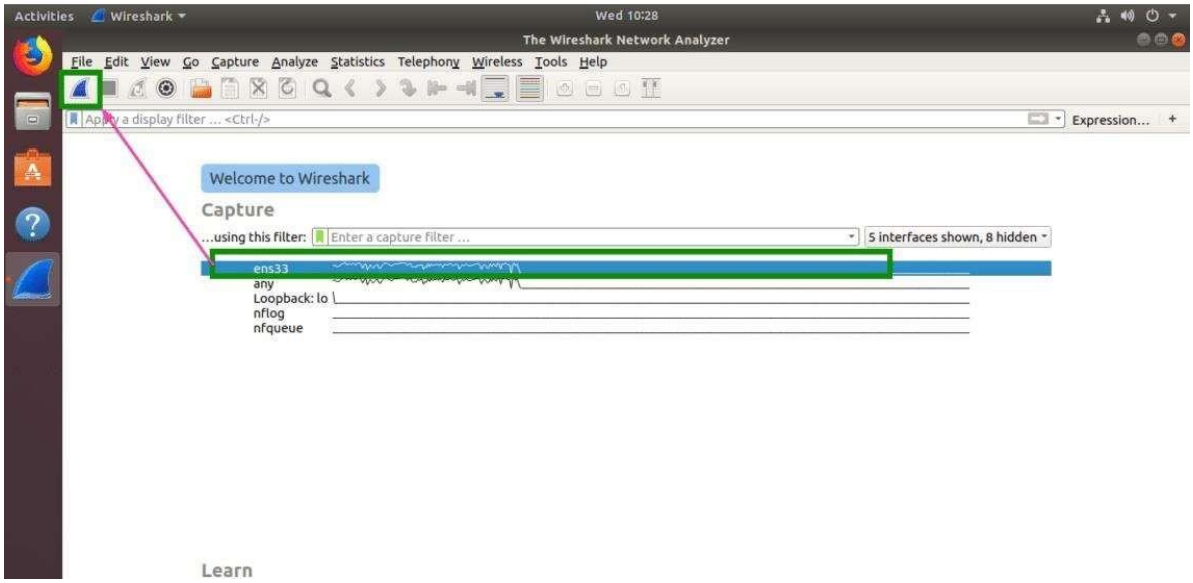
There are many types of interfaces you can monitor using Wireshark, for example, Wired, Wireless, USB and many external devices. You can choose to show specific types of interfaces in the welcome screen from the marked section of the screenshot below.



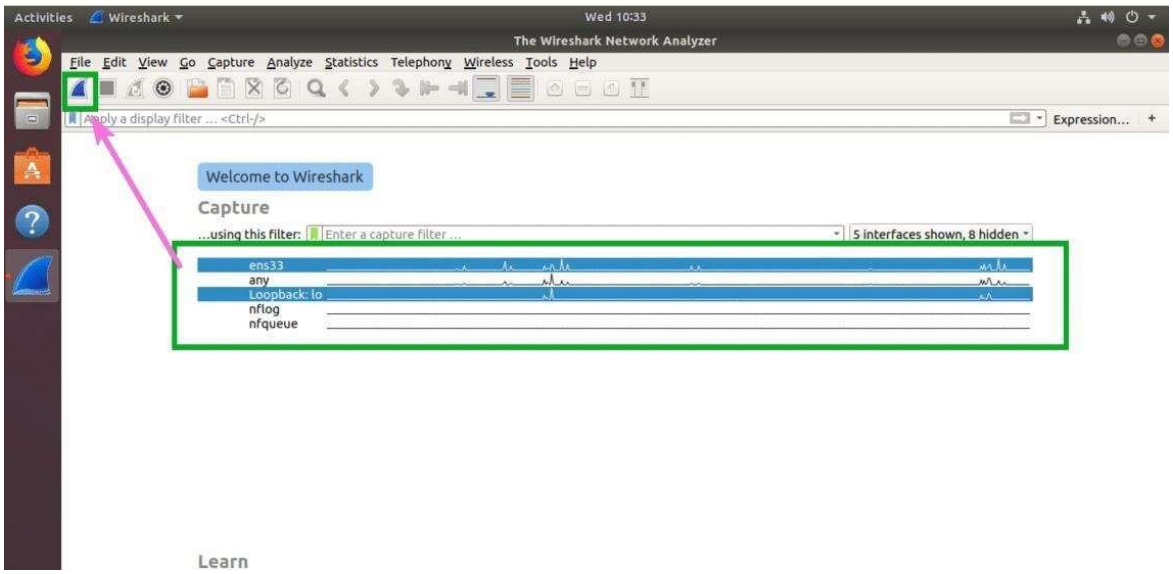
Here, I listed only the Wired network interfaces.



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.

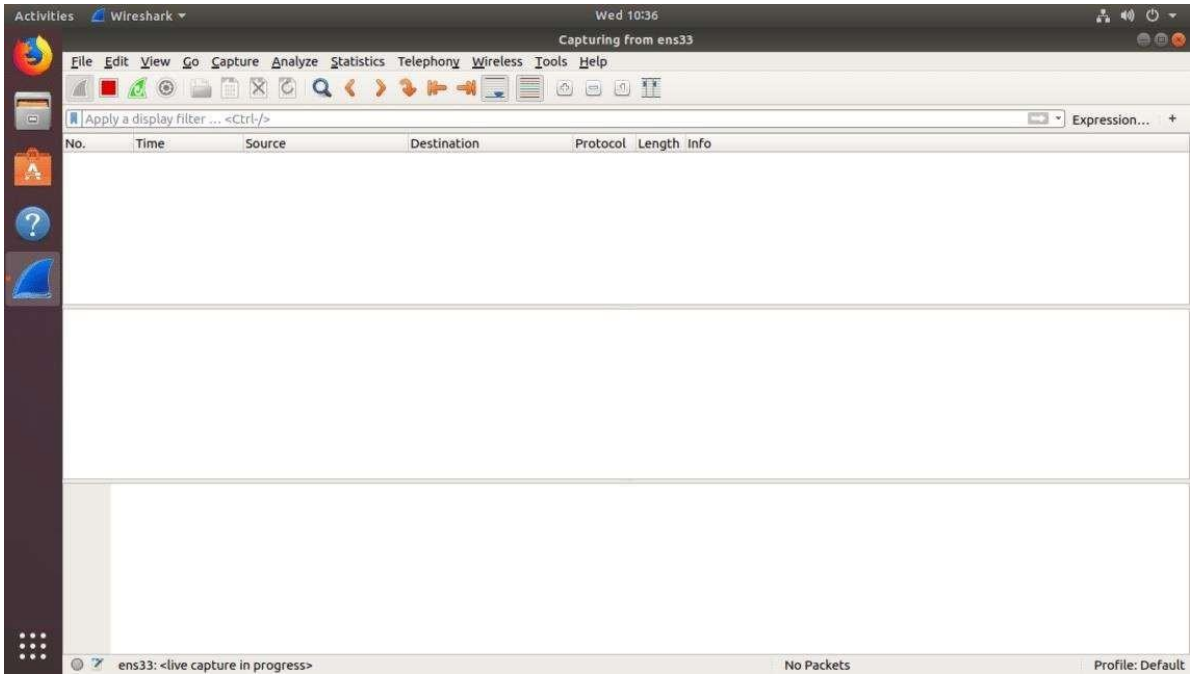


You can also capture packets to and from multiple interfaces at the same time. Just press and hold <Ctrl> and click on the interfaces that you want to capture packets to and from and then click on the Start capturing packets icon as marked in the screenshot below.

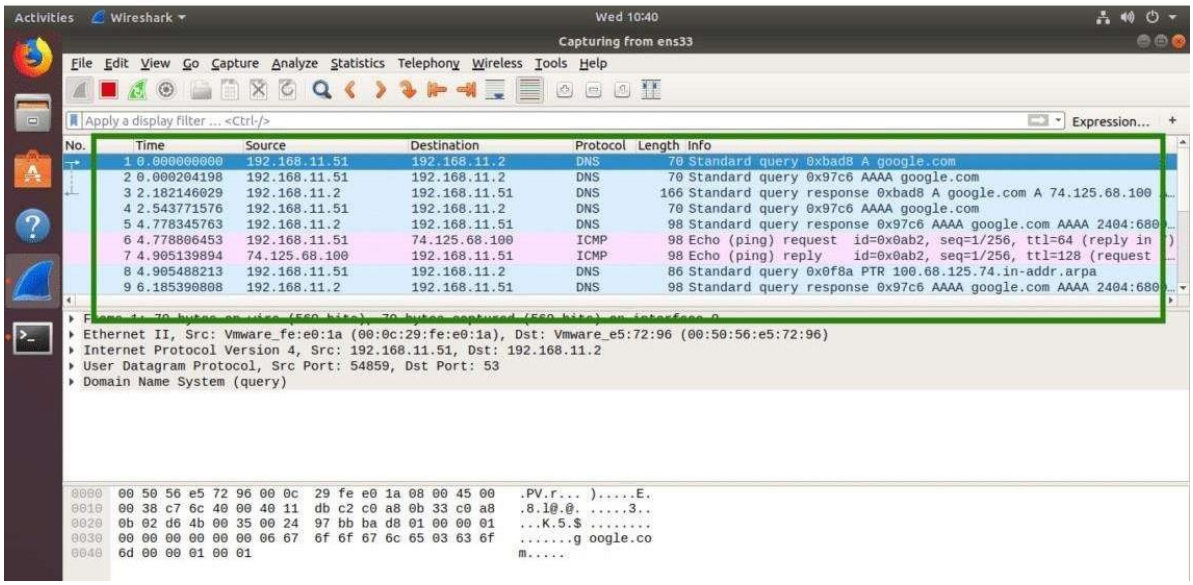


Using Wireshark on Ubuntu:

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



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



Starting Wireshark:

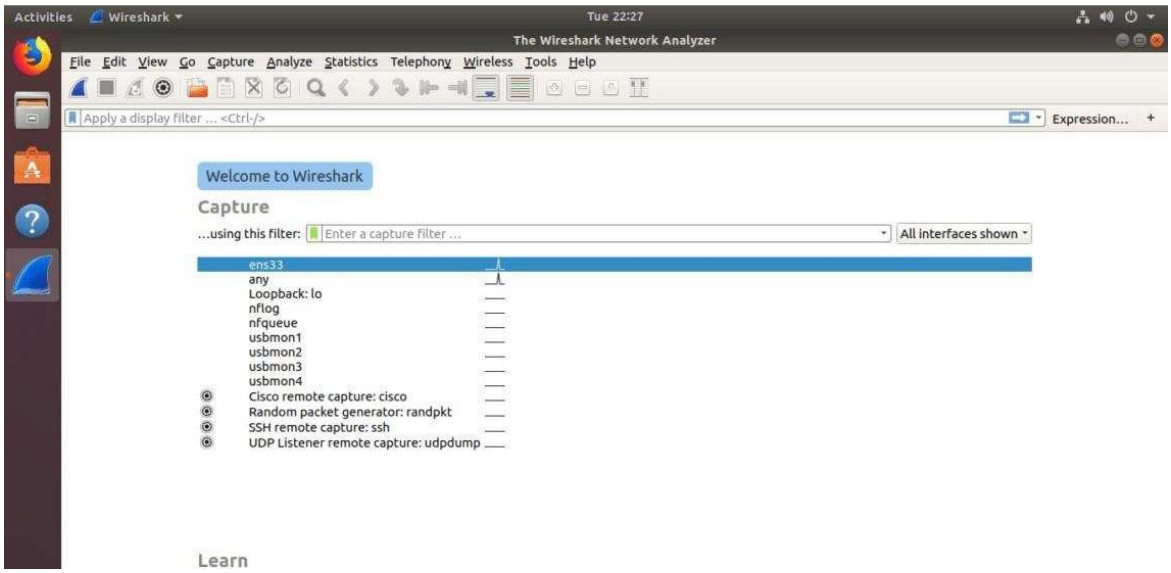
You can also run the following command to start Wireshark from the Terminal:

$ wireshark

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

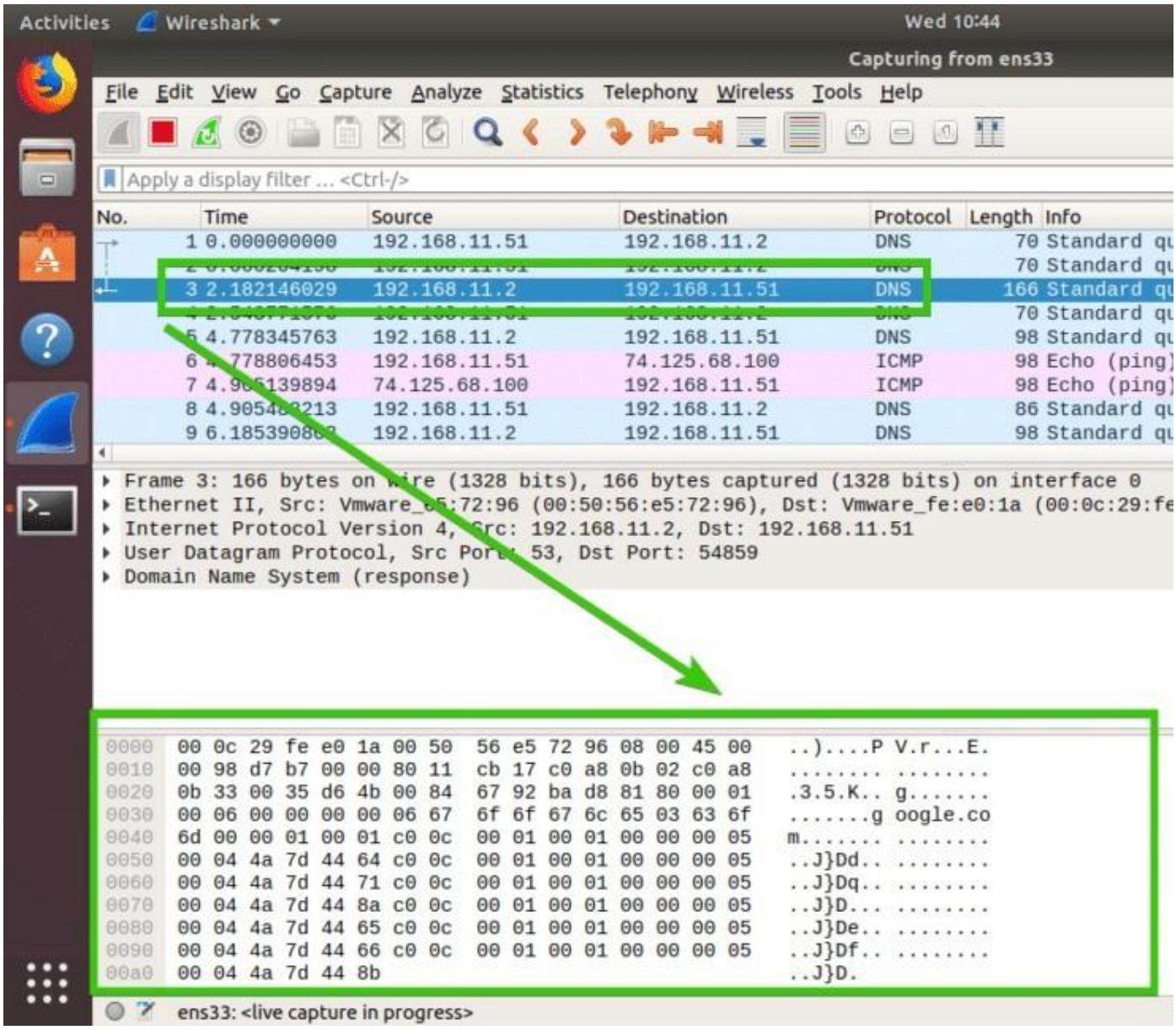
$ sudo wireshark

Wireshark should start.

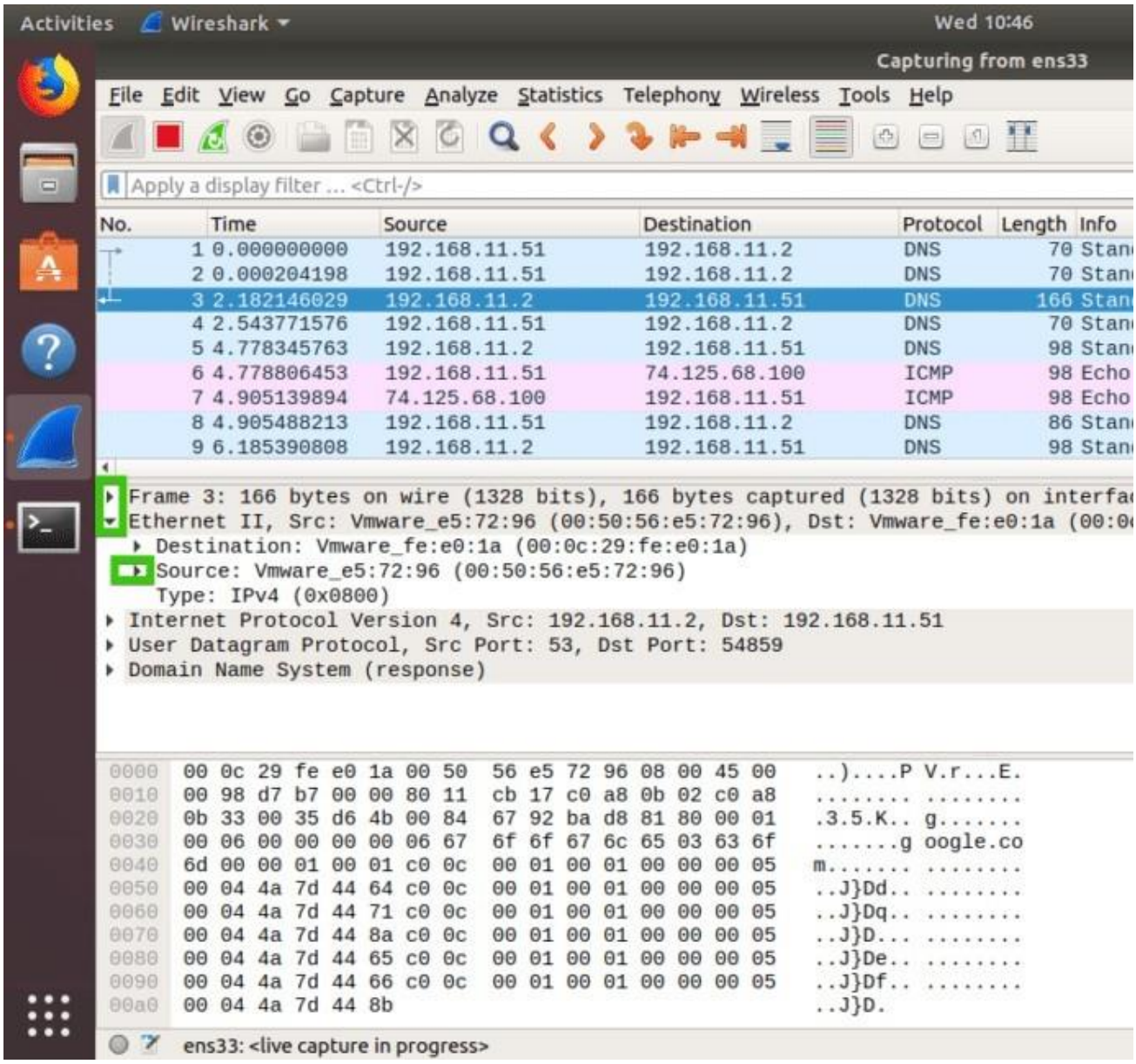


Viewing Captured Traffic

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

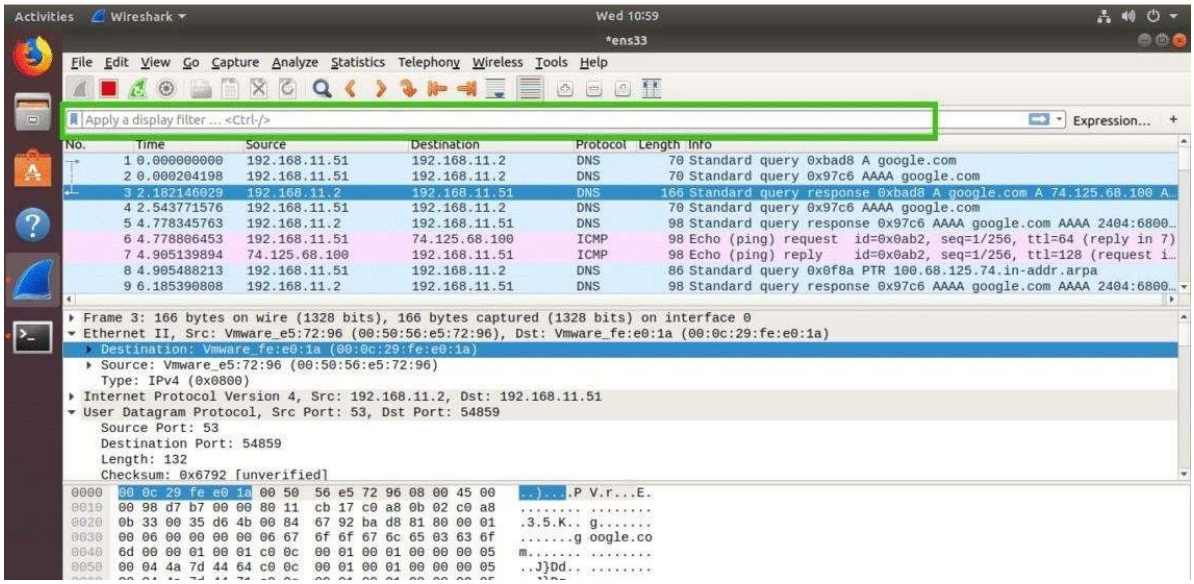


you can also click on the arrows to expand packet data for a particular TCP/IP Protocol Layer.



Analysis and Statistics & Filters.

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



11. How to run Nmap scan

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
```

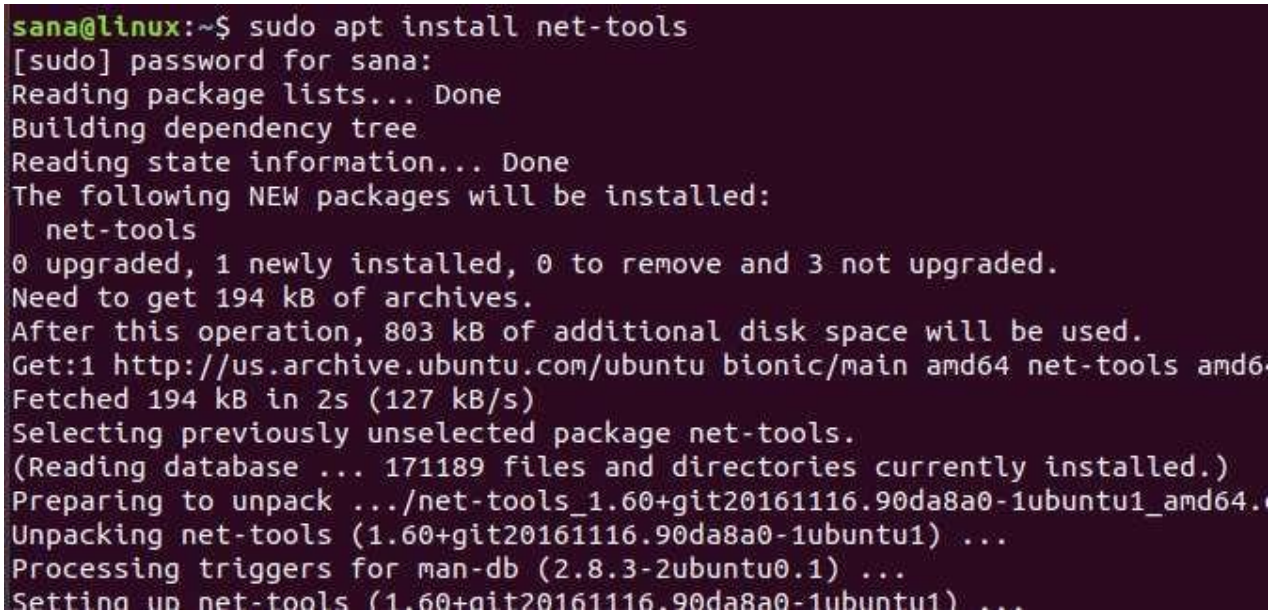
The system will ask you the password for sudo as only an authorized user can install/uninstall and configure software on Ubuntu.

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
```

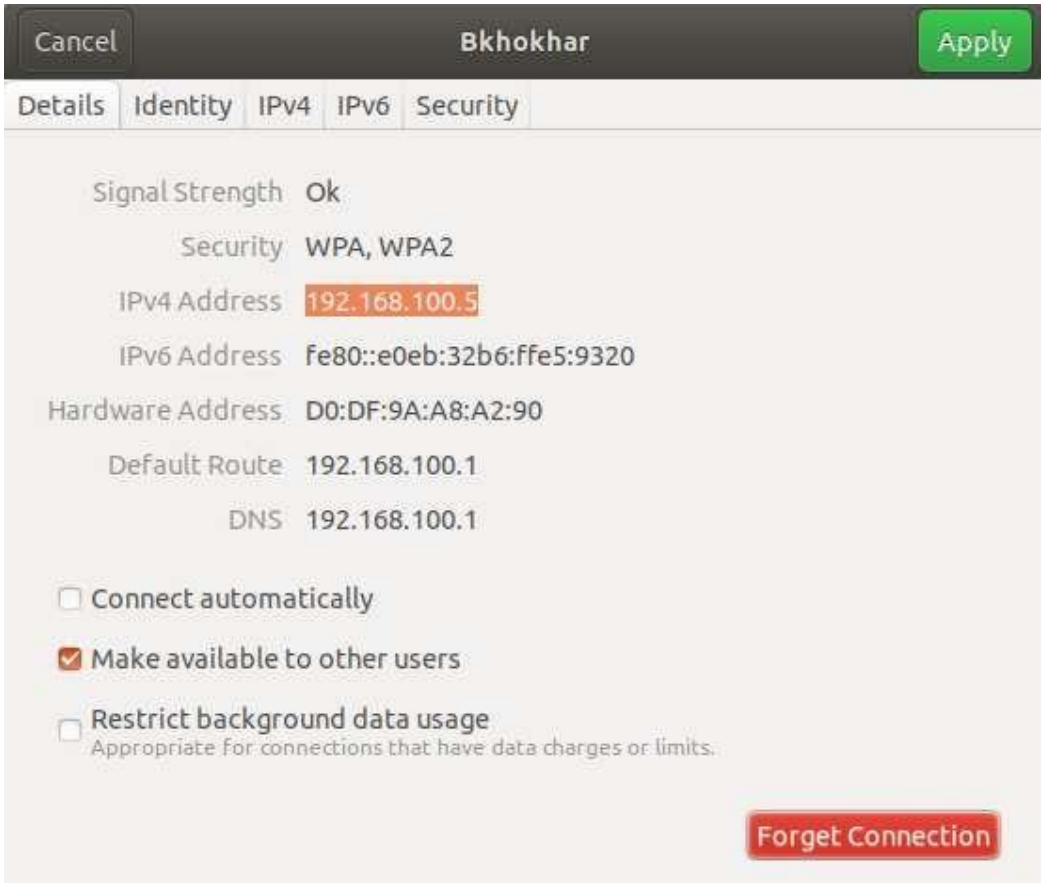
The highlighted IP from the output indicates that our system is using 192.168.100.0 subnet mask and the range is 255. Thus our network IP range is from 192.168.100.0 to 192.168.100.255.

Alternative (UI)

Instead of using the ifconfig tool, you can get the subnet mask IP through the Ubuntu GUI as well.

Access the Settings utility from the system Dash and check the details of your network either by clicking the settings icon against the wifi or ethernet network you are connected to.

In this example, we have checked the settings of a wi-fi network we are currently connected to.



Operating System Detection using Nmap

NMAP has a database which is installed when you install NMAP. The database is used when doing OS detection, but it is not automatically updated.

The database is located at '/usr/share/nmap/nmap-os-db'. The easiest way to manage an update is first to look at the database version number. Open the file in a text editor and the version number is usually listed on the second line. The second line of my database is '# $Id: nmap-os-db 35407 2015-11-10 04:26:26Z dmiller $'. The database version for this file is 35407.

OS Detection Process

Before we get into the actual command and performing an OS Detection we should cover some details about what is happening during this scan.

There are five separate probes being performed. Each probe may consist of one or more packets. The response to each packet by the target system helps to determine the OS type.

The five different probes are:

1. Sequence Generation
2. ICMP Echo
3. TCP Explicit Congestion Notification
4. TCP

5. UDP

Now we can look at these individually to see what they are doing.

Sequence Generation

The Sequence Generation Probe consists of six packets. The six packets are sent 100 ms apart and are all TCP SYN packets.

The result of each TCP SYN packet will help NMAP determine the OS type.

ICMP Echo

Two ICMP Request packets are sent to the target system with varying settings in the packet.

The resulting responses will help verify the OS type by NMAP.

TCP Explicit Congestion Notification

When a lot of packets are being generated and passing through a router causing it to be burdened is known as congestion. The result is that systems slow down to reduce congestion so the router is not dropping packets.

The packet being sent is only to get a response from the target system. Specific values returned are used to determine the specific OS since each OS handles the packets in different ways.

TCP

Six packets are sent during this probe.

Some packets are sent to open or closed ports with specific packet settings. Again, the results will vary depending on the target OS.

The TCP Packets are all sent with varying flags as follows:

1. no flags
2. SYN, FIN, URG and PSH
3. ACK
4. SYN
5. ACK
6. FIN, PSH, and URG

UDP

This probe consists of a single packet sent to a closed port.

If the port used on the target system is closed and an ICMP Port Unreachable message is returned then there

is no Firewall.

NMAP OS Detection Command

Now we need to run the actual command to perform an OS Detection. If you have read any of the other of my NMAP articles then it is best not to perform a PING. To skip the PING we use the parameter '-Pn'. To see the extra information we may require you should use the '-v' parameter for adding verbosity. Specifically to get the OS Detection the parameter '-O' is needed.

For the command to complete properly and perform the TCP SYN Scan you need to perform the command as ROOT. In my case, I will perform the scan on one system only and not the whole network so the command would be:

sudo nmap -v -Pn -O 192.168.0.63

The results of the scan are shown in Figure 2. The scan shows that there are seven open ports using a SYN Stealth Scan.

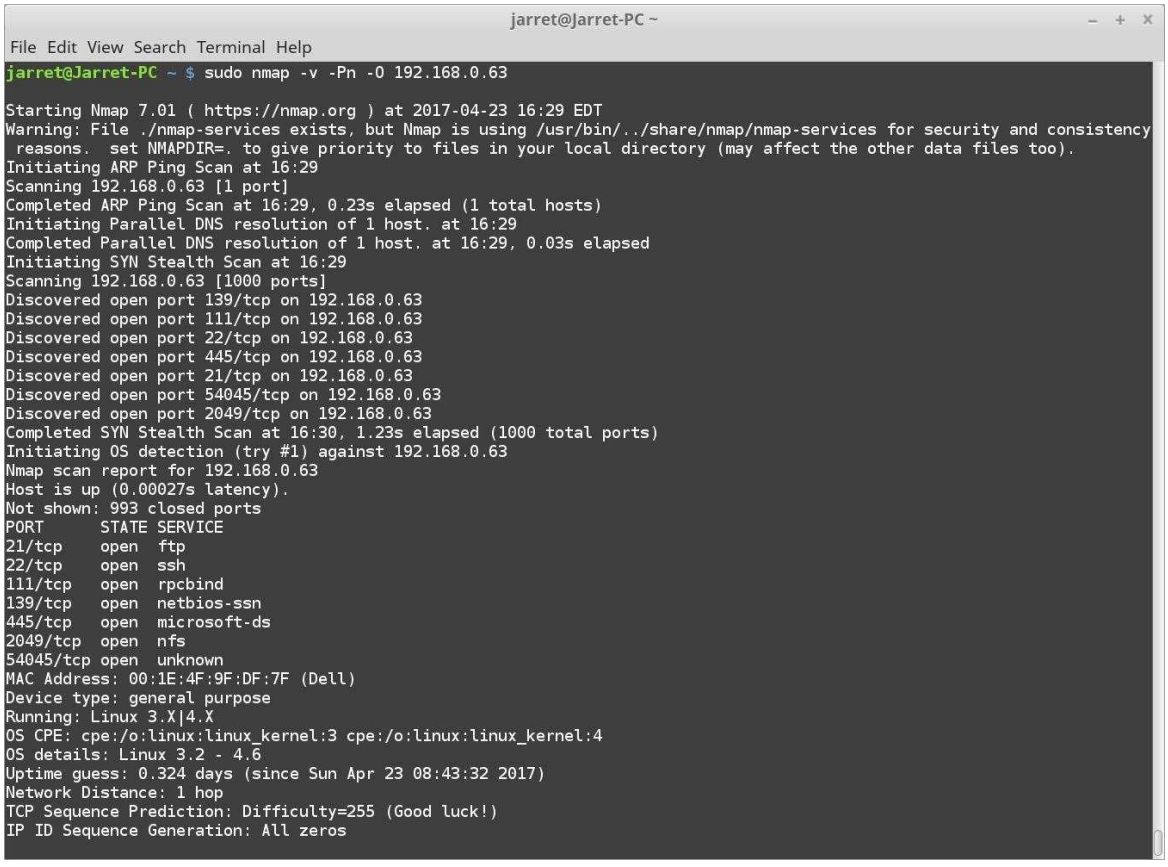


FIGURE 2

The open ports are:

1. 21/tcp ftp
2. 22/tcp ssh

3. 111/tcp rpcbind
4. 139/tcp netbios-ssn
5. 445/tcp microsoft-ds
6. 2049/tcp nfs
7. 54045/tcp unknown

The MAC Address of the system is 00:1E:4F:9F F:7F.

The final portion shows the OS type:

Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.6
Uptime guess: 0.324 days (since Sun Apr 23 08:43:32 2017)

The system is running Ubuntu Server 16.10 and the Linux Kernel is 4.8. The uptime is guessed correctly.

I performed a second test against another system. The results are shown in Figure 3.

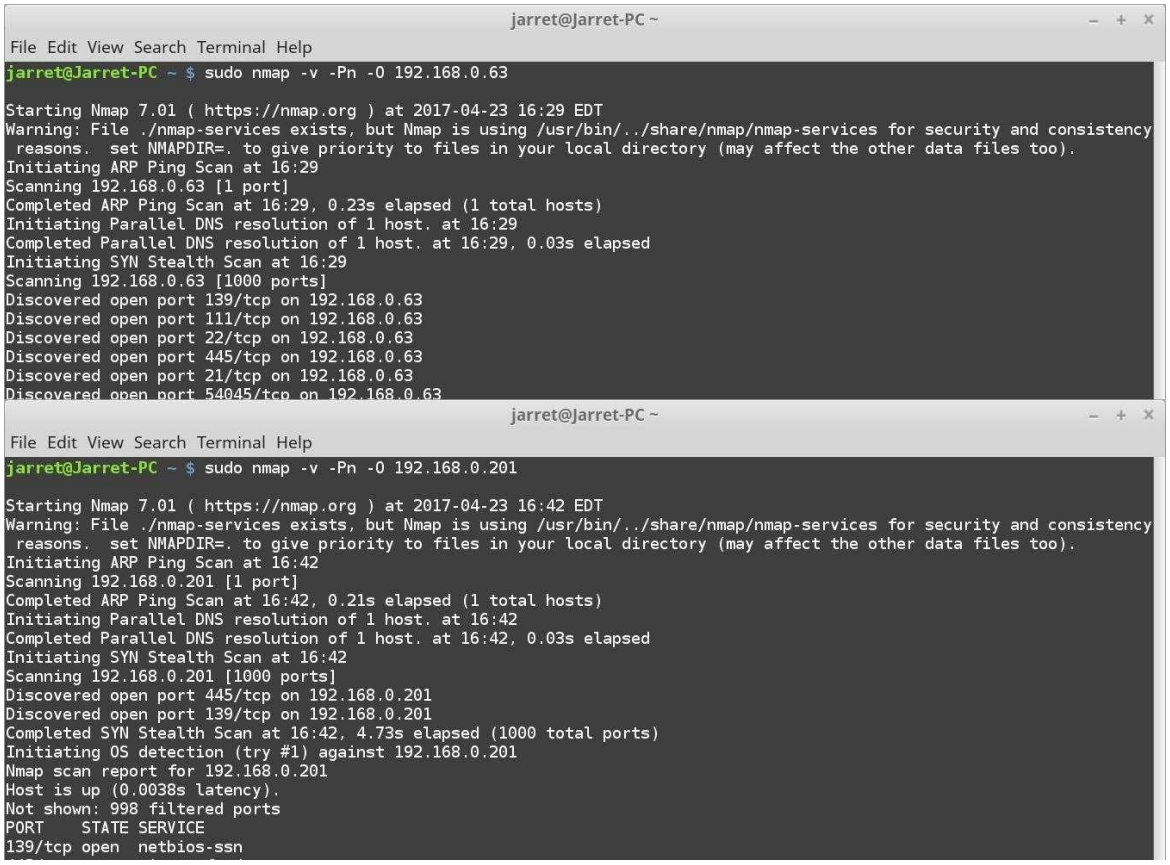


FIGURE 3

In this scan different ports are open. The system is guessed to be 'Microsoft Windows 2000|XP'. It is Windows XP running SP3.

Port Sniffer Results.

Let's look at what is going on in the background of the first scan shown in Figure 2.

Initially NMAP is performing a TCP Stealth Scan. In this instance the OS Detection Probes start at Packet 2032 in Figure 4.

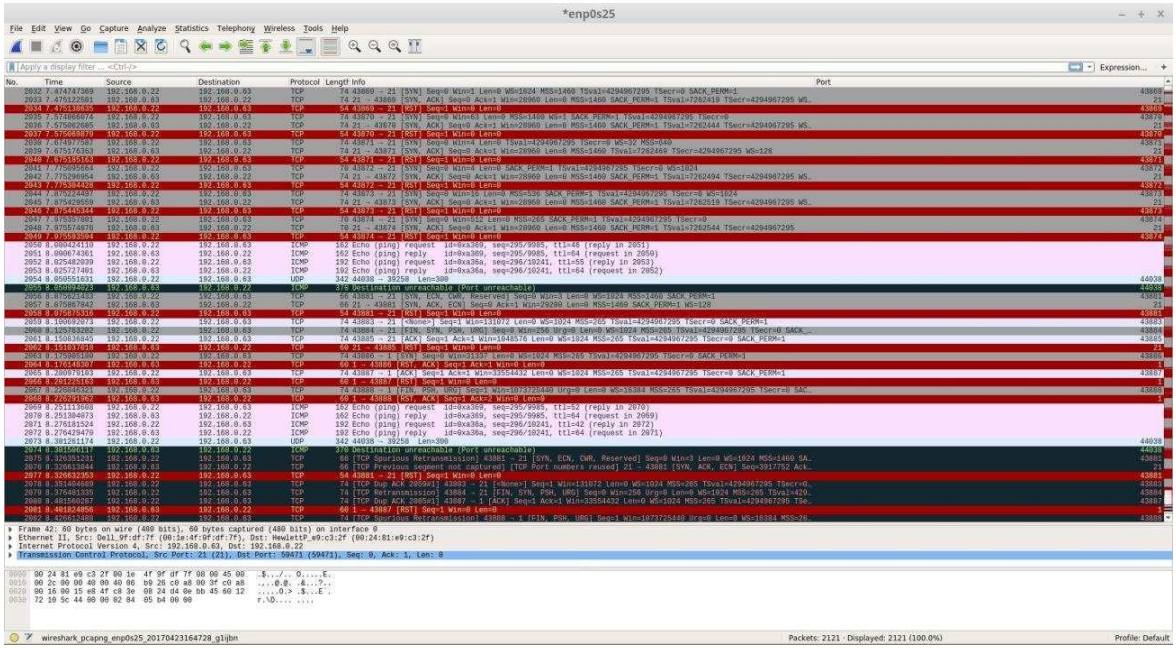


FIGURE 4

The Sequence Generation starts at Packet 2032/2033 and the sixth one is at Packet 2047/2048. Notice that each one is sent twice and the next packet in the probe is sent 100 ms later.

The ICMP Packets are sent at 2050-2053. The two packets are duplicated making a total of four.

Packets 2056-2057 are the TCP Explicit Congestion Notification packets.

The six probes for the TCP are on lines 2059, 2060, 2061, 2063, 2065 and 2067.

The last probe for UDP is line 2073.

These are the probes that are used to determine the OS of the target system.

I hope this helps you to understand how to update the NMAP OS Detection Database and perform a scan on systems. Be aware that you can run the scan on systems which are not on your network, so the scan can be performed on systems on the Internet.

13. Do the following using NS2 Simulator

i. NS2 Simulator-Introduction

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

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.

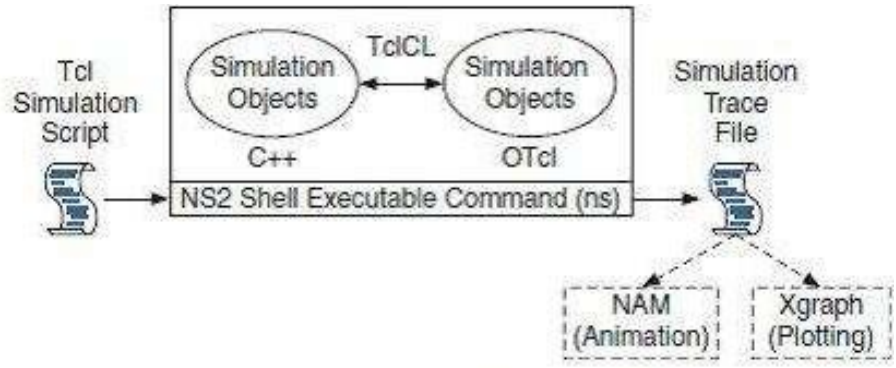
6. Otcl: Object oriented support

7. Tclcl: C++ and otcl linkage

8. Discrete event scheduler

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. Simulate to Find the Number of Packets Dropped

```
set ns [new Simulator] set nf [open lab1.nam w]
$ns namtrace-all $nf

/* Letter S is capital */
/* open a nam trace file in write mode */
/* nf – nam file */


set tf [open lab1.tr w] /* tf- trace file */
$ns trace-all $tf

proc finish { } { /* provide space b/w proc and finish and all are in small case */ global ns nf tf
$ns flush-trace                                    /* clears trace file contents */ close $nf
close $tf
exec nam lab1.nam & exit 0
}

set n0 [$ns node] /* creates 4 nodes */ set n1 [$ns node]
set n2 [$ns node] set n3 [$ns node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail /*Letter M is capital Mb*/
$ns duplex-link $n1 $n2 100Mb 5ms DropTail /*D and T are capital*/ $ns duplex-link $n2 $n31Mb 1000ms
DropTail

$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10
```

```
set udp0 [new Agent/UDP] /* Letters A,U,D and P are capital */
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR] /* A,T,C,B and R are capital*/
$cbr0 set packetSize_ 500 /*S is capital, space after underscore*/
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2

set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2

set null0 [new Agent/Null] /* A and N are capital */
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0

$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 1.0 "finish"
```

$ns run

AWK file (Open a new editor using "vi command" and write awk file and save with ".awk" extension)

/*immediately after BEGIN should open braces „{„

```
BEGIN { c=0;

}

{

If ($1= ="d")

{

c++;

printf("%s\t%s\n",$5,$11);

}

}
```

/*immediately after END should open braces „{„

```
END{

printf("The number of packets dropped =%d\n",c);

}
```

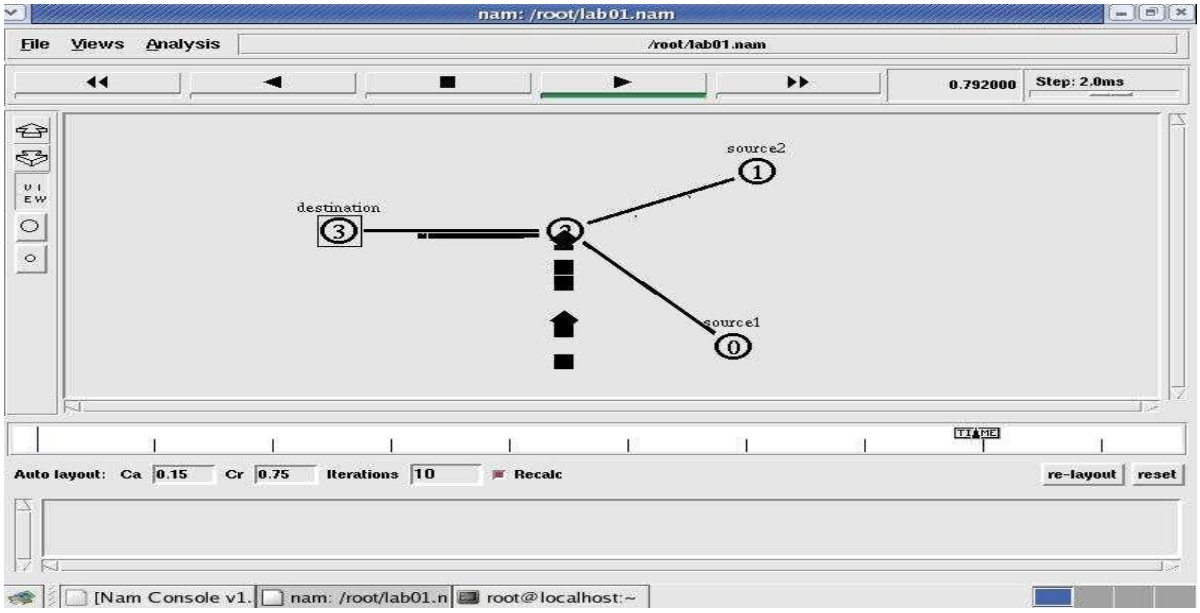
Steps for execution

1. Open vi editor and type program. Program name should have the extension **— .tcl‖**
   [root@localhost ~]# vi lab1.tcl
2. Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3. Open vi editor and type **awk** program. Program name should have the extension **-.awk ||**

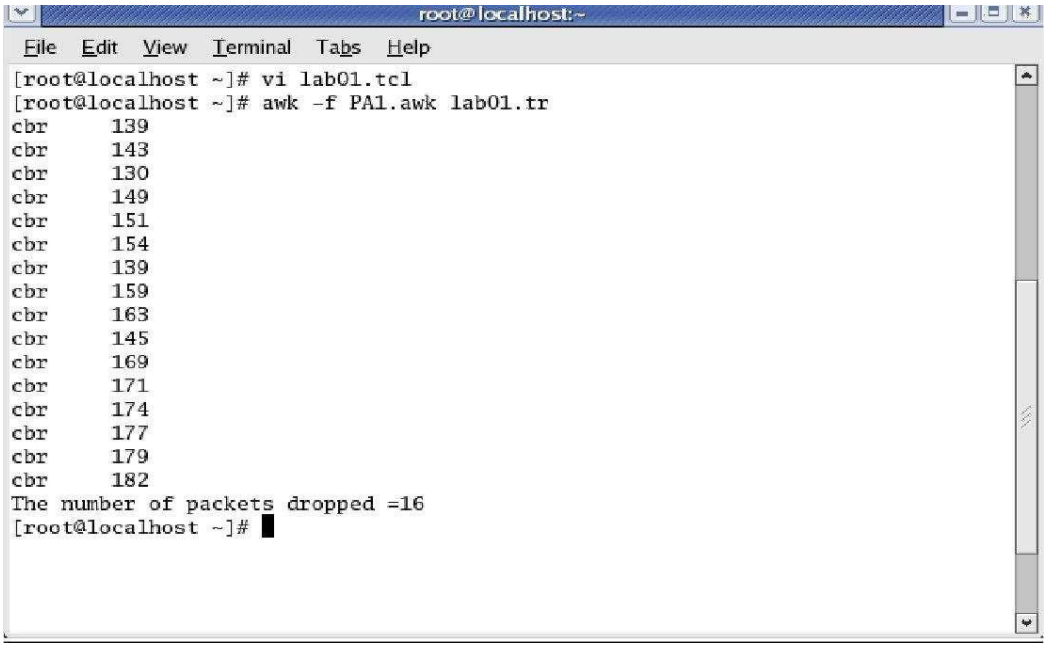
   [root@localhost ~]# vi lab1.awk
4. Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
5. Run the simulation program
   [root@localhost~]# ns lab1.tcl
   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
   ii) Now press the play button in the simulation window and the simulation will begins.
6. After simulation is completed run **awk file** to see the output ,
   [root@localhost~]# awk –f lab1.awk lab1.tr
7. To see the trace file contents open the file as ,
   [root@localhost~]# vi lab1.tr

Trace file contains 12 columns:-

Event type, Event time, From Node, Source Node, Packet Type, Packet Size, Flags (indicated by ---------), Flow ID, Source address, Destination address, Sequence ID, Packet ID



Output



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

set ns [new Simulator]

set nf [open lab2.nam w]

$ns namtrace-all $nf

set tf [open lab2.tr w]

$ns trace-all

$tf set n0 [$ns node]

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n4 shape box
$ns duplex-link $n0 $n4 1005Mb 1ms DropTail
$ns duplex-link $n1 $n4 50Mb 1ms DropTail
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail
$ns duplex-link $n3 $n4 200Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set p1 [new Agent/Ping]
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001
set p2 [new Agent/Ping]
$ns attach-agent $n1 $p2
set p3 [new Agent/Ping]
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001
set p4 [new Agent/Ping]
$ns attach-agent $n3 $p4 set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5
$ns queue-limit $n0 $n4 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n4 $n5 2 Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received answer from $from with round trip time $rtt msec"
```

```
}

# please provide space between $node_ and id. No space between $ and from. No #space between and $ and
rtt */

$ns connect $p1 $p5

$ns connect $p3 $p4 proc finish { } { global ns nf tf

$ns flush-trace close  $nf close Stf

exec nam lab2.nam & exit 0

}

$ns at 0.1 "$p1 send"

$ns at 0.2 "$p1 send"

$ns at 0.3 "$p1 send"

$ns at 0.4 "$p1 send"

$ns at 0.5 "$p1 send"

$ns at 0.6 "$p1 send"

$ns at 0.7 "$p1 send"

$ns at 0.8 "$p1 send"

$ns at 0.9 "$p1 send"

$ns at 1.0 "$p1 send"

$ns at 1.1 "$p1 send"

$ns at 1.2 "$p1 send"

$ns at 1.3 "$p1 send"

$ns at 1.4 "$p1 send"

$ns at 1.5 "$p1 send"

$ns at 1.6 "$p1 send"

$ns at 1.7 "$p1 send"

$ns at 1.8 "$p1 send"

$ns at 1.9 "$p1 send"

$ns at 2.0 "$p1 send"

$ns at 2.1 "$p1 send"

$ns at 2.2 "$p1 send"
```

```
$ns at 2.3 "$p1 send"
$ns at 2.4 "$p1 send"
$ns at 2.5 "$p1 send"
$ns at 2.6 "$p1 send"
$ns at 2.7 "$p1 send"
$ns at 2.8 "$p1 send"
$ns at 2.9 "$p1 send"
$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
$ns at 1.1 "$p3 send"
$ns at 1.2 "$p3 send"
$ns at 1.3 "$p3 send"
$ns at 1.4 "$p3 send"
$ns at 1.5 "$p3 send"
$ns at 1.6 "$p3 send"
$ns at 1.7 "$p3 send"
$ns at 1.8 "$p3 send"
$ns at 1.9 "$p3 send"
$ns at 2.0 "$p3 send"
$ns at 2.1 "$p3 send"
$ns at 2.2 "$p3 send"
```

```
$ns at 2.3 "$p3 send"

$ns at 2.4 "$p3 send"

$ns at 2.5 "$p3 send"

$ns at 2.6 "$p3 send"

$ns at 2.7 "$p3 send"

$ns at 2.8 "$p3 send"

$ns at 2.9 "$p3 send"

$ns at 3.0 "finish"

$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with ".awk" extension)

```
BEGIN{

drop=0;

}

{

if($1= ="d" )

{

drop++;

}

} END{

printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);

}
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension **— .tcl**
   [root@localhost ~]# vi lab2.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3) Open vi editor and type **awk** program. Program name should have the extension **–.awk**
   [root@localhost ~]# vi lab2.awk
4) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
5) Run the simulation program
   [root@localhost~]# ns lab2.tcl
   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.
   ii) Now press the play button in the simulation window and the simulation will

begins.

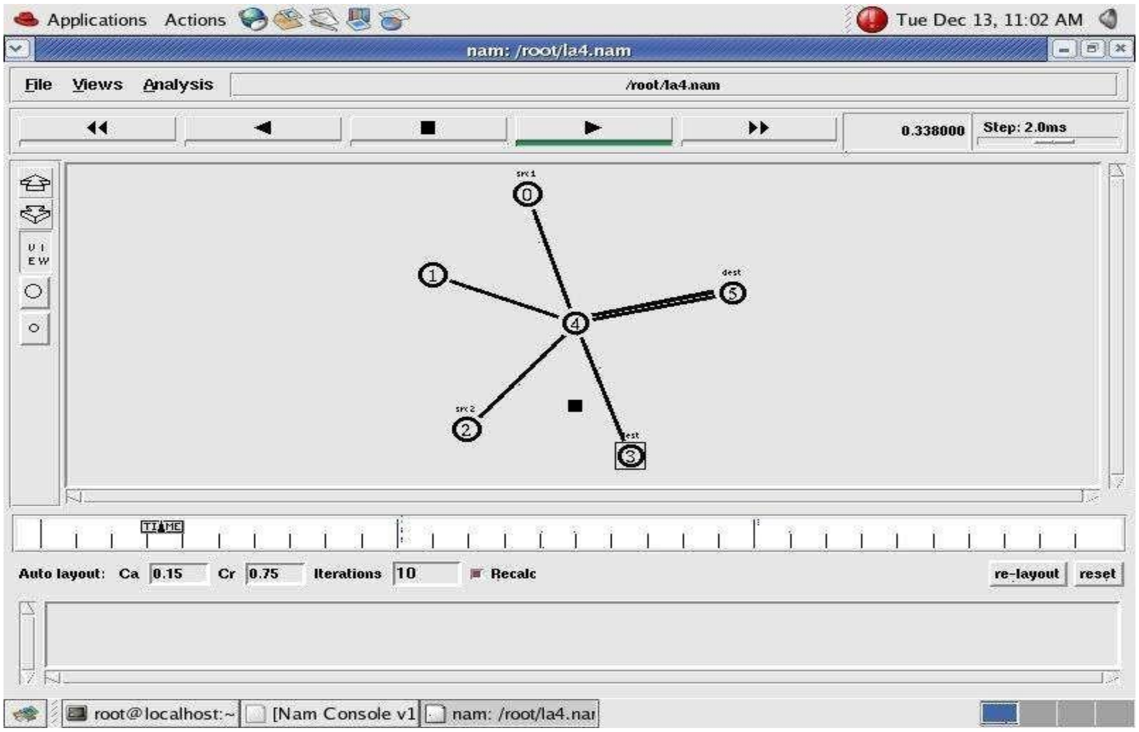
- After simulation is completed run **awk file** to see the output ,

  [root@localhost~]# awk –f lab2.awk lab2.tr

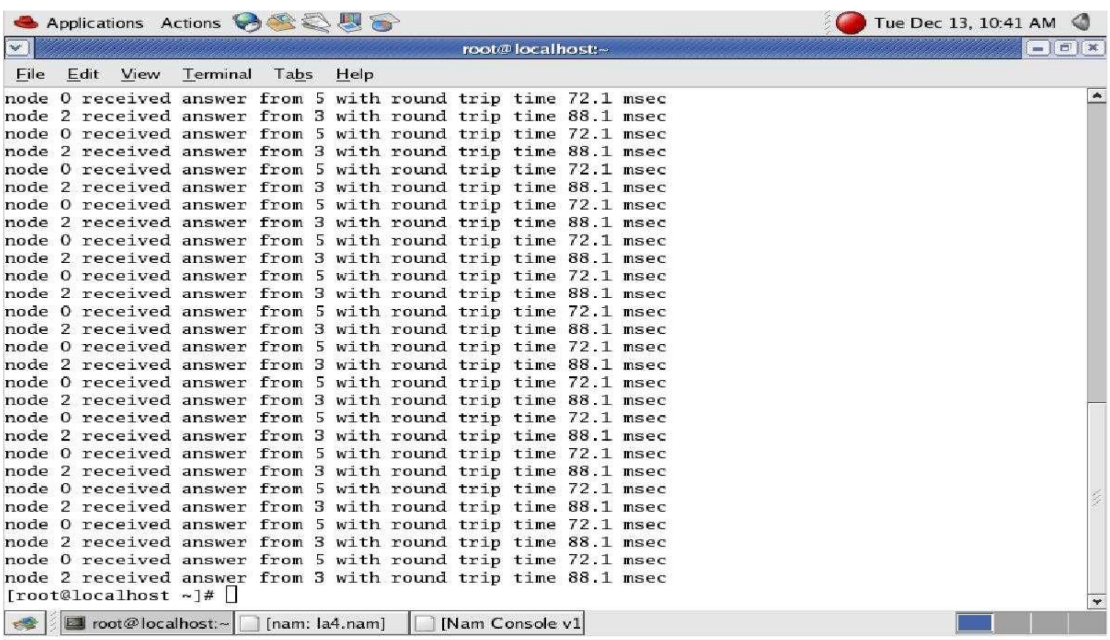
- To see the trace file contents open the file as ,

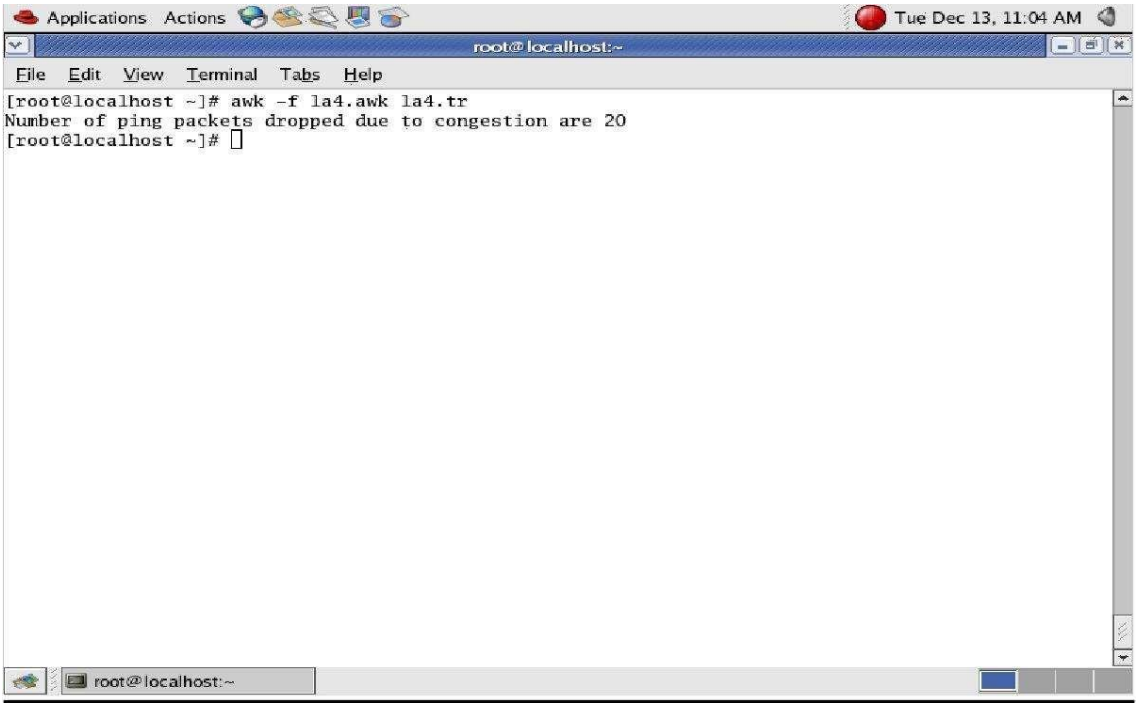
  [root@localhost~]# vi lab2.tr

Topology



OUTPUT





Note:

Vary the bandwidth and queue size between the nodes n0-n2 , n2-n4. n6-n2 and n2- n5 and see the number of packets dropped at the nodes.

!V.Simulate to Plot Congestion for Different Source/Destination

set ns [new Simulator] set tf [open lab3.tr w]

$ns trace-all $tf

set nf [open lab3.nam w]

$ns namtrace-all $nf

set n0 [$ns node]

$n0 color "magenta"

$n0 label "src1" set n1 [$ns node] set n2 [$ns node]

$n2 color "magenta"

$n2 label "src2" set n3 [$ns node]

$n3 color "blue"

$n3 label "dest2" set n4 [$ns node]

set n5 [$ns node]

$n5 color "blue"

$n5 l

```
abel "dest1"

$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
/* should come in single line */
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5 set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3

$ns connect $tcp2 $sink3
set file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp2 attach $file2
```

```
$tcp0 trace cwnd_ /* must put underscore ( _ ) after cwnd and no space between them*/

$tcp2 trace cwnd_


proc finish { } { global ns nf tf

$ns flush-trace close $tf

close $nf

exec nam lab3.nam & exit 0

}

$ns at 0.1 "$ftp0 start"

$ns at 5 "$ftp0 stop"

$ns at 7 "$ftp0 start"

$ns at 0.2 "$ftp2 start"

$ns at 8 "$ftp2 stop"

$ns at 14 "$ftp0 stop"

$ns at 10 "$ftp2 start"

$ns at 15 "$ftp2 stop"

$ns at 16 "finish"

$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with ".awk" extension)

cwnd:- means congestion window

```
BEGIN {
}
{
if($6= ="cwnd_") /* don't leave space after writing cwnd_ */
printf("%f\t%f\t\n",$1,$7); /* you must put \n in printf */ }

END {
}
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension **—.tcl**
   [root@localhost ~]# vi lab3.tcl
2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.
3) Open vi editor and type **awk** program. Program name should have the extension

-.awk ||

[root@localhost ~]# vi lab3.awk

4 Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

5 Run the simulation program

[root@localhost~]# ns lab3.tcl

8 After simulation is completed run **awk file** to see the output ,

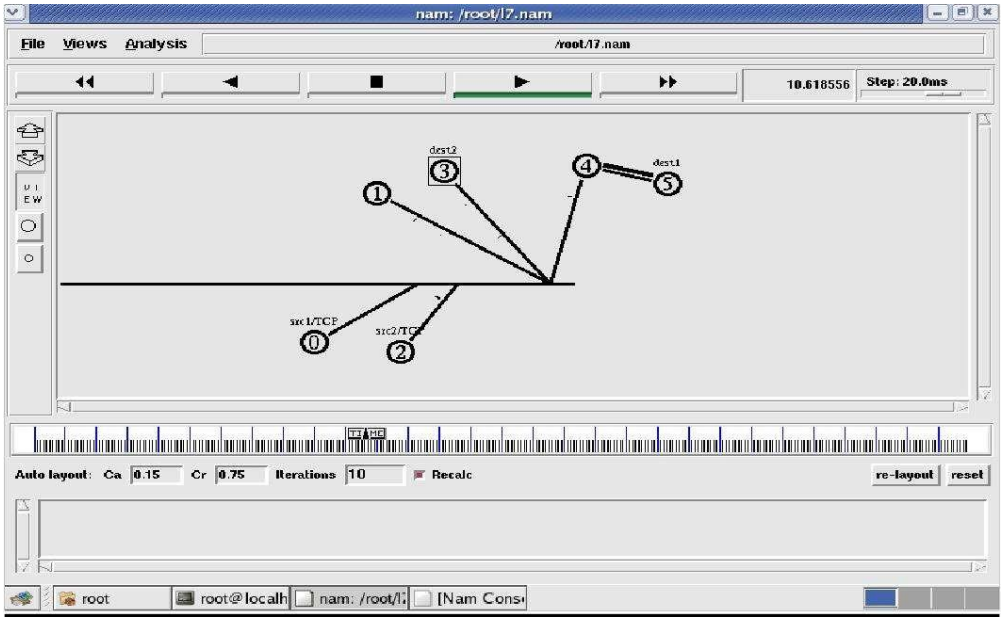
i. **[root@localhost~]# awk –f lab3.awk file1.tr > a1**
ii. **[root@localhost~]# awk –f lab3.awk file2.tr > a2**
iii. **[root@localhost~]# xgraph a1 a2**

9 Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.

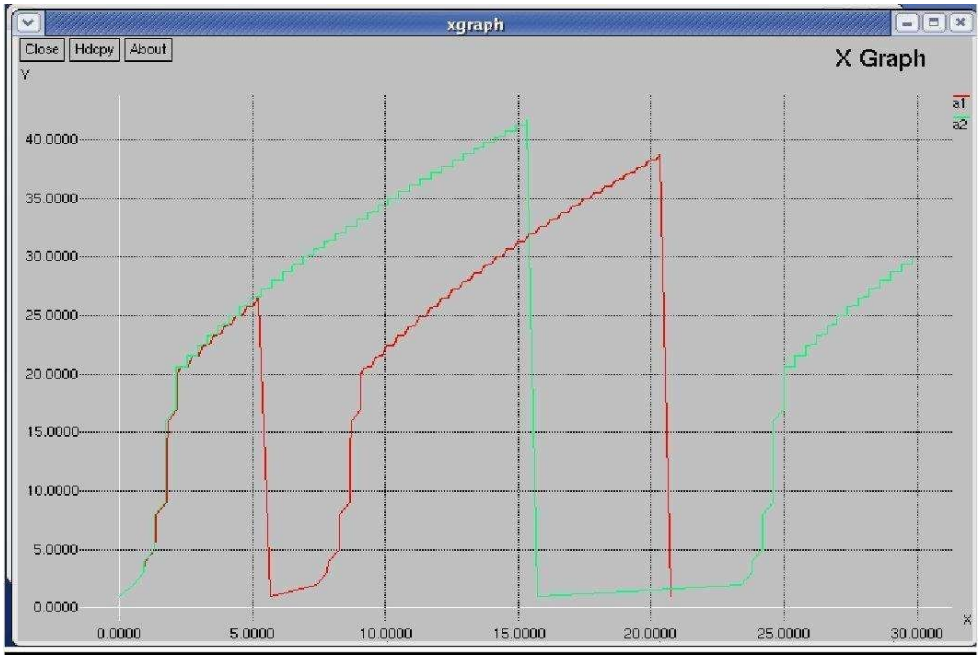
10 To see the trace file contents open the file as ,

[root@localhost~]# vi lab3.tr

Topology



Output



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

set ns [new Simulator] set tf [open lab4.tr w]

$ns trace-all $tf

set topo [new Topography]

$topo load_flatgrid 1000 1000 set nf [open lab4.nam w]

$ns namtrace-all-wireless $nf 1000 1000

$ns node-config -adhocRouting DSDV \

-llType LL \

-macType Mac/802_11 \

-ifqType Queue/DropTail \

-ifqLen 50 \

-phyType Phy/WirelessPhy \

-channelType Channel/WirelessChannel \

-propType Propagation/TwoRayGround \

-antType Antenna/OmniAntenna \

-topoInstance $topo \

-agentTrace ON \

```
-routerTrace ON

create-god 3

set n0 [$ns node] set n1 [$ns node] set n2 [$ns node]

$n0 label "tcp0"

$n1 label "sink1/tcp1"

$n2 label "sink2"


$n0 set X_ 50

$n0 set Y_ 50

$n0 set Z_ 0

$n1 set X_ 100

$n1 set Y_ 100

$n1 set Z_ 0

$n2 set X_ 600

$n2 set Y_ 600

$n2 set Z_ 0

$ns at 0.1 "$n0 setdest 50 50 15"

$ns at 0.1 "$n1 setdest 100 100 25"

$ns at 0.1 "$n2 setdest 600 600 25"


set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

set sink1 [new Agent/TCPSink]

$ns attach-agent $n1 $sink1

$ns connect $tcp0 $sink1 set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set ftp1 [new Application/FTP]
```

```
$ftp1 attach-agent $tcp1

set sink2 [new Agent/TCPSink]

$ns attach-agent $n2 $sink2

$ns connect $tcp1 $sink2

$ns at 5 "$ftp0 start"

$ns at 5 "$ftp1 start"

$ns at 100 "$n1 setdest 550 550 15"

$ns at 190 "$n1 setdest 70 70 15" proc finish { } {

global ns nf tf

$ns flush-trace

exec nam lab4.nam & close $tf

exit 0

}

$ns at 250 "finish"

$ns run
```

AWK file (Open a new editor using "vi command" and write awk file and save with ".awk" extension)

```
BEGIN{
        count1=0
        count2=0
        pack1=0
        pack2=0
        time1=0
}       time2=0
{

        if($1= ="r"&& $3= ="_1_" && $4= ="AGT")
        {
                count1++
                pack1=pack1+$8
                time1=$2
        }
        if($1= ="r" && $3= ="_2_" && $4= ="AGT")
        {
                count2++
                pack2=pack2+$8
```

```
                time2=$2
        }
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n",
((count1*pack1*8)/(time1*1000000))); printf("The Throughput from n1 to
n2: %f Mbps", ((count2*pack2*8)/(time2*1000000))); }
```

Steps for execution

1) Open vi editor and type program. Program name should have the extension **—.tcl**

   [root@localhost ~]# vi lab4.tcl

2) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

3) Open vi editor and type **awk** program. Program name should have the extension **-.awk ||**

   [root@localhost ~]# vi lab4.awk

4) Save the program by pressing **"ESC key"** first, followed by **"Shift and :"** keys simultaneously and type **"wq"** and press **Enter key**.

5) Run the simulation program

   [root@localhost~]# ns lab4.tcl

   i) Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.

   ii) Now press the play button in the simulation window and the simulation will begins.

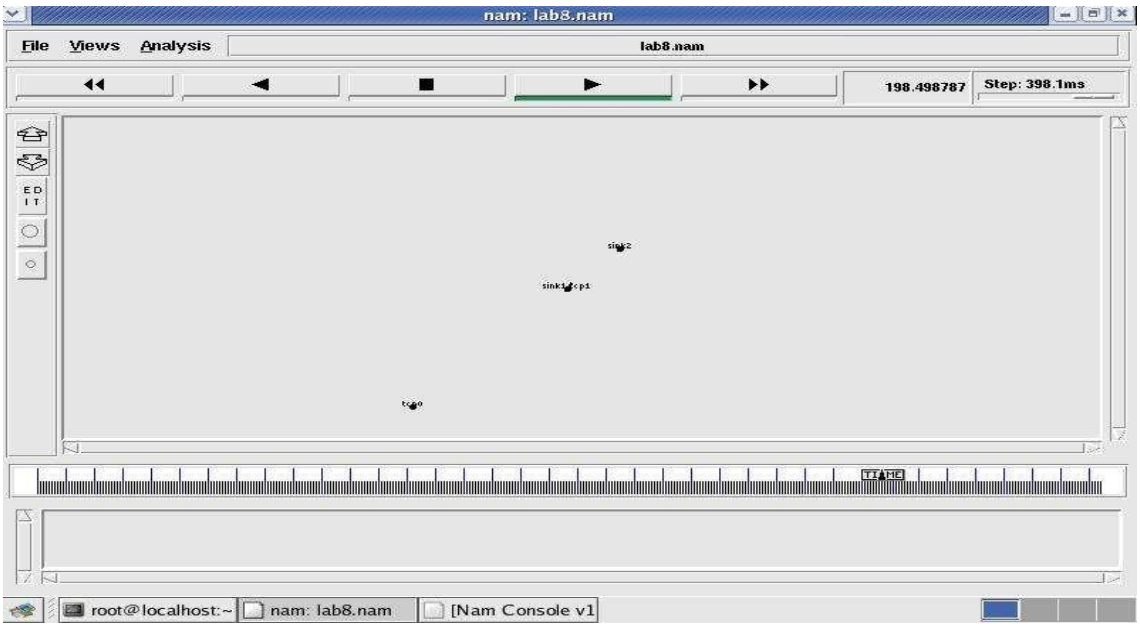
6) After simulation is completed run **awk file** to see the output ,

   [root@localhost~]# awk –f lab4.awk lab4.tr

7) To see the trace file contents open the file as ,

   [root@localhost~]# vi lab4.tr

Topology

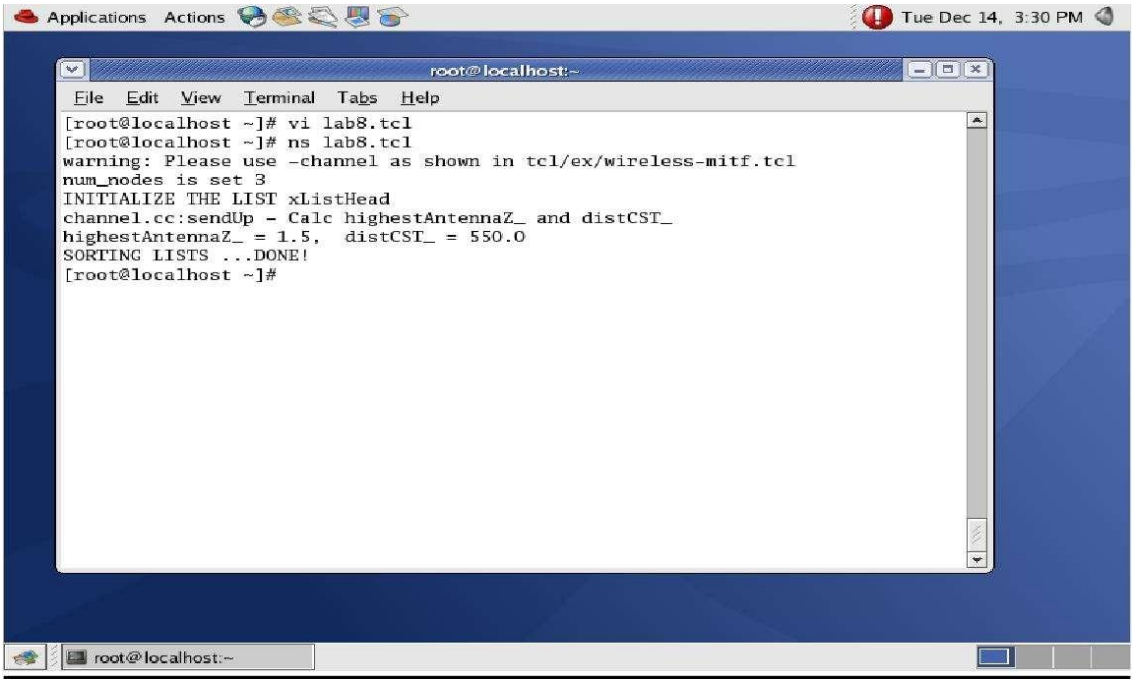


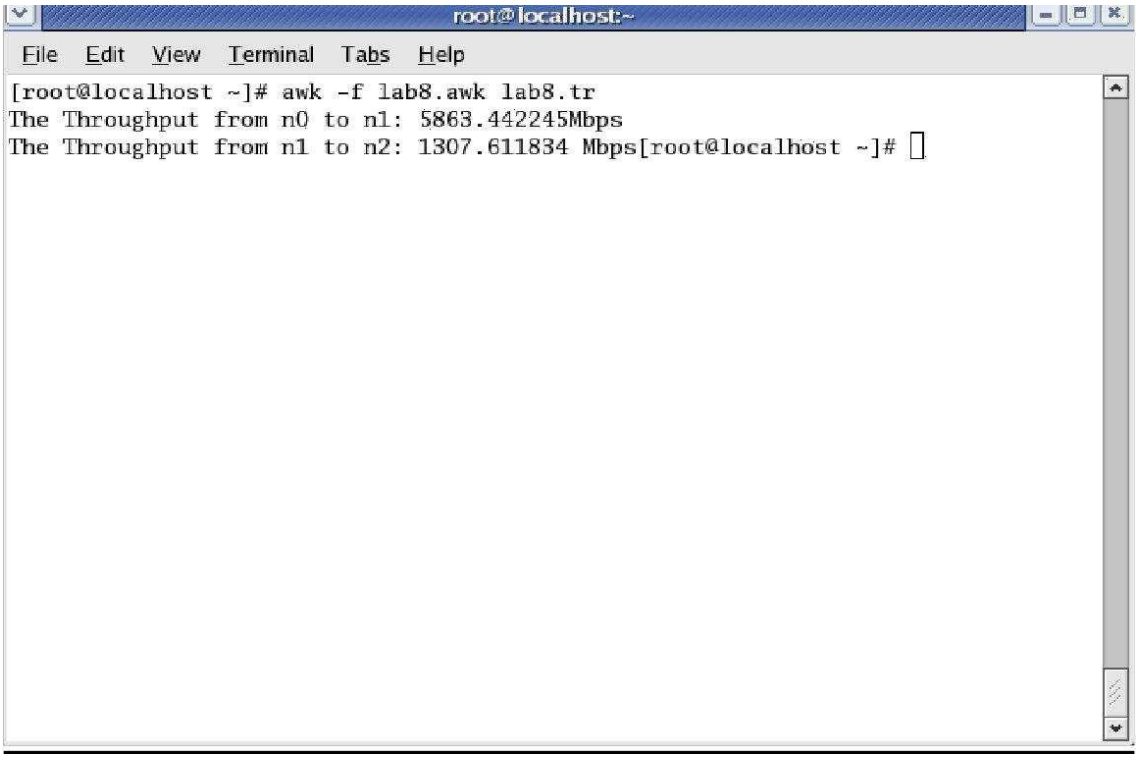
Trace file

```
root@localhost:~
File  Edit  View  Terminal  Tabs  Help
s 0.036400876 _0_ RTR  --- 0 message 32 [0 0 0 0] ------- [0:255 -1:255 32 0]
r 0.037421112 _1_ RTR  --- 0 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255
 32 0]
M 0.10000 0 (50.00, 50.00, 0.00), (50.00, 50.00), 15.00
M 0.10000 1 (100.00, 100.00, 0.00), (100.00, 100.00), 25.00
M 0.10000 2 (600.00, 600.00, 0.00), (600.00, 600.00), 25.00
s 0.182633994 _1_ RTR  --- 1 message 32 [0 0 0 0] ------- [1:255 -1:255 32 0]
r 0.183694230 _0_ RTR  --- 1 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255
 32 0]
s 0.882774710 _2_ RTR  --- 2 message 32 [0 0 0 0] ------- [2:255 -1:255 32 0]
s 5.000000000 _0_ AGT  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
r 5.000000000 _0_ RTR  --- 3 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0
s 5.000000000 _0_ RTR  --- 3 tcp 60 [0 0 0 0] ------- [0:0 1:0 32 1] [0 0] 0 0
s 5.000000000 _1_ AGT  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.000000000 _1_ RTR  --- 4 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
r 5.004812650 _1_ AGT  --- 3 tcp 60 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1
 0
s 5.004812650 _1_ AGT  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.004812650 _1_ RTR  --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
s 5.004812650 _1_ RTR  --- 5 ack 60 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0
r 5.006977357 _0_ AGT  --- 5 ack 60 [13a 0 1 800] ------- [1:0 0:0 32 0] [0 0] 1
 0
s 5.006977357 _0_ AGT  --- 6 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [1 0] 0 0
"lab8.tr" 128664L, 11456314C                                  1,1           Top
```

Here **"M"** indicates mobile nodes, **"AGT"** indicates Agent Trace, **"RTR"** indicates Router Trace

Output


Applications Actions
Tue Dec 14, 3:30 PM
root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# vi lab8.tcl
[root@localhost ~]# ns lab8.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5, distCST_ = 550.0
SORTING LISTS ...DONE!
[root@localhost ~]#
root@localhost:~


root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# awk -f lab8.awk lab8.tr
The Throughput from n0 to n1: 5863.442245Mbps
The Throughput from n1 to n2: 1307.611834 Mbps[root@localhost ~]#