B.M.S. COLLEGE OF ENGINEERING

Department of Electronics and Communication Engineering

Bull Temple Road, Basavanagudi, Bangalore - 560 019

COMUPTER COMMUNICATION NETWORKS

[19EC6PCCCN]

LAB REPORT

Submitted by

Name of the Student	USN
Deviprasad H	1BM19EC044

Faculty In Charge

Dr. K. N. Madhusudan

Assistant Professor

Academic Year 2021-22

List of Experiments

Part A

1. Write a program to demonstrate framing.
2. Write a program to generate CRC code for checking error.
3. Write a program to simulate Shortest Path Routing Algorithm.
4. To study the Basic Networking Commands.
5. To demonstrate the Connection of computers in Local Area Network.
6. Configure Host IP, Subnet Mask and Default Gateway in a System in LAN (TCP/IP Configuration).
7. Write a program to encrypt and decrypt a given message using substitution cipher method.
8. Write a program to implement Diffie-Hellman Algorithm.

Part B

1. Configure network with the following topologies and analyze
i) BUS ii) RING iii) Fully connected mesh topology, disable a node in each of the topologies and find the changes.

2. Simulate Ethernet LAN with 4 nodes , apply relevant TCP and UDP applications and determine
i) the number of data packets sent by UDP and TCP
ii)Average jitter of UDP and TCP
iii)Number of periodic updates sent by the routing algorithm
iv)number of ACK packets sent

3. Simulate a network of N nodes with point to point connection; apply TCP and UDP applications vary the queue size and bandwidth and find
i) Number of packets dropped due to queue overflow
ii) Average hop count for data packets
iii) Average delay and jitter.
iv)Apply FTP and TELNET traffic between the nodes of the above network and analyze the throughput.

4. Simulate Ethernet LAN with N nodes , configure multicast traffic and Determine
i)the total multicast data bytes received
ii) Total multicast data bytes transmitted
iii) Multicast average delay at the transport layer for UDP
iv)Packets sent by DVMRP
v)Neighbors for every node as determined by DVMRP
vi)packets dropped due to expired TTL
vii)Packets dropped due to no route.

5. Apply multiple UDP and TCP applications between any 2 nodes of N (N=4)node Ethernet LAN and compare it with experiment number 4.(compare multiple unicast with multicast)

6. Simulate a wireless ad hoc network apply relevant TCP and UDP applications between any 2 nodes and determine
i) Number of packets dropped due to retransmission limit
ii) Number of CTS packets sent by the node
iii) Number of RTS packets sent and ACK packets sent by the node
iv) Determine the number of RTS retransmission due to timeout
v) Packet retransmission due to ACK timeout
vi) Signals received with error

7. Simulate a network having 2 LANs connected by a switch. Apply relevant TCP and UDP applications between nodes across the LANS (send data from a node in one LAN to a node in another LAN) and determine application layer, transport layer, network layer and MAC layer parameters.

8. Simulate a network with the topology as shown in the figure, apply TCP and UDP applications between nodes shown in the figure. Modify the network to make communication happen between node 1 and 9 and node 6 and 16

9. Configure a network of 5 routers with point to point connection. Apply RIP and OSPF routing algorithms and compare.

10. Simulate a wireless infrastructure network with 6 nodes and analyze.

11. Configure a wired network with 4 nodes and wireless infrastructure network with 4 nodes apply relevant TCP and UDP applications from a node in wired network to a node in wireless network and analyze

12. a. Simulate wireless ad hoc network with 6 nodes give mobility to a node and analyze b. give mobility to all the nodes.

PART A

Experiment 1:

Aim: Write a program to demonstrate framing.

Introduction:
Framing in the data link layer separates a message from one source to a destination, or from other messages to other destinations, by adding a sender address and a destination address. Although the whole message could be packed in one frame, that is not normally done. One the reason is that a frame can be very large, making flow and error control very inefficient. When a message is carried in one very large frame, even a single bit error would require the retransmission of the whole message. When a message is divided into smaller frames, a single bit error affects only that small frame.
There are two types of framing:
a. Fixed Size Framing
b. Variable Size Framing

Theory:
Variable Size Framing involves two approaches: Bit oriented and Character Oriented.

1. Bit-Oriented Protocols
In a bit-oriented protocol, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video and so on. In addition to headers and possible trailers, there is a flag (01111110) to separate one frame from other. If this flag sequence appears in the data, bit stuffing is done to inform the receiver that this is not the end of the frame. Bit stuffing is the process of stuffing one extra bit 0 to the data sequence whenever 0 followed by five 1s regardless of the value of the next bit. This extra stuffed bit is eventually removed from the data at the receiver. This guarantees that the flag field sequence does not inadvertently appear in the frame.

2. Character-Oriented Protocols
In a character-oriented protocol, the data to be carried are 8-bit characters from a coding system such as ASCII. In addition to headers and possible trailers, there is a flag (01111110) to separate one frame from another. If this flag sequence appears in the data, byte stuffing is done to inform the receiver that this is not the end of the frame. Byte stuffing is the process of stuffing one extra byte (escape character - ESC) whenever there is a flag or escape character in the text. Whenever the receiver encounters the ESC character, it removes the ESC character from the data section and treats the next character as data, not a delimiting flag.

1. Bit stuffing and destuffing:

Code:

```
#include <iostream>
using namespace std;

char* stuff(char * data, int sz)
{
   int newsz= (int)sz/6+ sz;
   int count,i,j;
   char* stuffed=(char*)malloc(newsz*sizeof(char));
   for(i=0, j=0;i<newsz;i++, j++)
   {
```

```
        if(data[j]=='0' && data[j+1]=='1')
        {
           stuffed[i]=data[j];
           for(count=1;count<6;count++)
           {
              stuffed[i+count]=data[j+count];
              if(data[j+count]=='1')  continue;
              else break;
           }
           count--;
           i+=count;
           j+=count;
           if(count == 5)
           {
              stuffed[i+1]='0';
              i++;
              count=1;
           }
           else count=1;

        }
        else stuffed[i]=data[j];
    }
    return stuffed;
}

char* destuff(char* stuffed, int sz)
{
    int stsz=(int)sz/6+ sz;
    char* final=(char*)malloc(sz*sizeof(char));
    int count;
    for(int i=0, j=0;i<stsz;i++,j++)
    {
        if(stuffed[i]=='0' && stuffed[i+1]=='1')
        {
           final[j]=stuffed[i];
           for(count=1;count<6;count++)
           {
  final[j+count]=stuffed[i+count];
              if(stuffed[i+count]=='1') continue;
              else  break;
           }
           count--;
           i+=count;
           j+=count;
           if(count==5) i++;
           count=1;
        }
        else final[j]=stuffed[i];
    }
    return final;
}

int main()
{
```

```
    int sz;
    cout<<"enter frame size: ";
    cin>>sz;
    char data[sz];
    cout<<"enter data frame: ";
    cin>>data;
    char* stuffed = stuff(data, sz);
    cout<<"stuffed data:\t"<<stuffed<<endl;
    cout<<"framed data: 0111110"<<stuffed<<"0111110\n";
    char* destuffed= destuff(stuffed, sz);
    cout<<"destuffed data:\t"<<destuffed<<endl;
    return 0;
}
```

Output:

```
enter frame size: 10
enter data frame: 1010111111
stuffed data:   10101111101
framed data: 01111101010111111010111110
destuffed data: 1010111111


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

2. Byte stuffing and destuffing:

Code:

```
#include<iostream>
using namespace std;

char* stuffed(char* data, char esc, char fl, int s)
{
    char* final=(char*)malloc(2*s*sizeof(char));
    for(int i=0,j=0;i<s;i++, j++)
    {
        if(data[i]==esc || data[i]== fl)
        {
            final[j]=fl;
            j++;
            final[j]=data[i];
        }
        else final[j]=data[i];
    }
    return final;
}
char* destuffed(char* stuff, char esc, char fl, int sz)
{
    char* final=(char*)malloc(sz*sizeof(char));
    for(int i=0,j=0;i<sz;i++, j++)
    {
        if(stuff[j]==fl)
```

```
        {
          j++;
          final[i]=stuff[j];
        }
        else final[i]=stuff[j];
    }
    return final;
}
int main()
{
    int sz;
    cout<<"Enter number of characters in data frame: ";
    cin>>sz;
    char data[sz];
    char esc='}', flag='~';
    cout<<"enter data sequence: ";
    cin>>data;
    cout<<"escape char:  "<<esc<<endl<<"flag char:  "<<flag<<endl;
    char* stuff= stuffed(data, esc, flag, sz);
    cout<<"stuffed data= "<<stuff<<endl;
    cout<<"framed data:"<<esc<<stuff<<esc<<endl;
    char* destuff=destuffed(stuff, esc, flag, sz);
    cout<<"destuffed data= "<<destuff;
    return 0;
}
```

Output:

```
Enter number of characters in data frame: 10
enter data sequence: aBc~D}e}~f
escape char:  }
flag char:  ~
stuffed data= aBc~~D~}e~}~~f
framed data:}aBc~~D~}e~}~~f}
destuffed data= aBc~D}e}~f

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

Experiment 2:

Aim: Write a program to generate CRC code for checking error.

Introduction:

When bits are transmitted through the network, they might get corrupted due to interference. This results in error and it is wrongly decoded at the receiver. Error detection techniques are used to check whether any error has occurred or not. It is responsible only for the error detection and does not take into account the number of error bits. This error detection is possible by adding some extra bits to the data. Sender sends the original data along with some additional bits called the redundant bits. The receiver examines these redundant bits and finds out if the data is free from error. These redundant bits are removed before passing it to the upper layers. Cyclic Redundancy Check (CRC) is one of the techniques for detecting error in data frames.

Theory:

CRC is a category of cyclic codes that is used in networks such as LANs and WANs. If the dataword is k bits and codeword is n bits, then the augmented dataword is obtained by adding (n-k) zero's to the right hand side of the dataword or it is obtained by adding (divisor length - 1) number of zero's to the RHS of dataword. The augmented dataword is divided by the divisor (modulo-2 division). The quotient of the division is discarded. The remainder of the division is appended to the dataword to create the codeword. At the receiver, the decoder checks the redundant bits. Again, the codeword are divided with the same divisor. The remainder of the division is the syndrome. If the syndrome bits are all 0's, it means that during the transmission the data bits are not changed. Hence, redundant bits are removed and the dataword is accepted. Otherwise, it is understood that data is corrupted and the dataword is discarded.

Code:

```
#include  <stdio.h>

#include <conio.h>

#include <string.h>

void main()

{

int i,j,keylen,msglen;

char input[100], key[30],temp[30],quot[100],rem[30],key1[30];

printf("Enter Dataword: "); //Read dataword from user

gets(input);

printf("Enter Divisor: "); //Read divisor from user

gets(key);

keylen=strlen(key);

msglen=strlen(input);

char output[msglen+keylen];
```

```
strcpy(output,input);
strcpy(key1,key);
for (i=0;i<keylen-1;i++) //keylen-1 number of zeros is appended to input
input[msglen+i]='0';
printf("Data after padding zeros is ");
for(i=0;i<msglen+keylen-1;i++)
printf("%c",input[i]); //Prints the data sequence after appending 0's
for (i=0;i<keylen;i++)
temp[i]=input[i]; //Copy of input is stored in temp
for (i=0;i<msglen;i++)
{
quot[i]=temp[0]; //Quotient is first bit of dividend
if(quot[i]=='0') //If quotient is 0, then divide it by 0
{
for (j=0;j<keylen;j++)
key[j]='0';
}
else //If quotient=1, then divisor remains the same
{
for (j=0;j<keylen;j++)
key[j]=key1[j];
}
for (j=keylen-1;j>0;j--) //Perform XOR operation
{
if(temp[j]==key[j]) //If two bits are same, then rem=0
rem[j-1]='0';
else //If two bits are different, rem=1
rem[j-1]='1';
}
rem[keylen-1]=input[i+keylen]; //next bit of input is assigned to last bit of rem
strcpy(temp,rem); //Copy rem to temp. Temp is dividend for next iteration
}
```

```
strcpy(rem,temp); //Copy final temp to rem
strcat(output,rem); //Concatenating remainder to dataword
int codelen=strlen(output);
printf("\nCodeword is ");
for(i=0;i<codelen;i++)
printf("%c",output[i]); //Print codeword
char rec_input[msglen+strlen(key1)];
char temp1[30],quot1[100],rem1[30];
char key_r[30],key1_r[30];
strcpy(key_r,key1);
printf("\nEnter received codeword: "); //Reads Received codeword
gets(rec_input);
int rec_len=strlen(rec_input);
for (i=0;i<keylen;i++) //Copy of user input of received bits after transmission to temp1
temp1[i]=rec_input[i];
for (i=0;i<msglen;i++)
{
quot1[i]=temp1[0];
if(quot1[i]=='0') //If quot1 is 0, then divide it by 0
{
for (j=0;j<keylen;j++)
key_r[j]='0';
}
else //If quot1 is 1, then divisor remains the same
{
for (j=0;j<keylen;j++)
key_r[j]=key1[j];
}
for (j=keylen-1;j>0;j--) //Perform XOR operation
{
if(temp1[j]==key_r[j]) //If two bits are same, then rem1=0
rem1[j-1]='0';
```

```
else //If two bits are different, rem1=1
rem1[j-1]='1';
}
rem1[keylen-1]=rec_input[i+keylen];//next bit of received data is assigned to last bit of rem1
strcpy(temp1,rem1);//Copy rem1 to temp1. Temp1 is dividend for next iteration
}
strcpy(rem1,temp1);//Copy temp1 to rem1
int rem1_len=strlen(rem1);
printf("Remainder is ");
for (i=0;i<rem1_len;i++)
printf("%c",rem1[i]);
int flag=0;
for(i=0;i<keylen-1;i++)
{ /*Check rem1 bits
If all bits are not zero, then the data is corrupted during transmission
If all bits are zero then there is no error during transmission*/
if(rem1[i]!='0')
{
flag=1;
break;
}
}
if(flag)
printf("\nDataword is discarded");
else
printf("\nDataword is accepted");
getch();
}
```

Output:

When the receiver receives correct dataword:

```
Enter Dataword: 101010
Enter Divisor: 101
Data after padding zeros is 10101000
Codeword is 10101010
Enter received codeword: 10101010
Remainder is 00
Dataword is accepted

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

When the receiver receives incorrect dataword:

```
Enter Dataword: 101010
Enter Divisor: 101
Data after padding zeros is 10101000
Codeword is 10101010
Enter received codeword: 10111010
Remainder is 01
Dataword is discarded

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

Experiment 3:

Aim: Write a program to simulate Shortest Path Routing Algorithm.

Theory:

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. Routing is a process of finding the path between source and destination upon request of data transmission. Dijkstra's algorithm is used to build a routing table for routers in link state routing. The algorithm divides the nodes into two sets that are tentative and permanent sets. It finds the neighbours of a current node and makes them tentative examines them and if they pass the criteria makes them permanent node. The criteria for making the node permanent is that the cost of the node from the source node must be minimum compared to all other nodes present in the tentative list. The OSPF protocol i.e., open shortest path first protocol which is inter-domain routing protocol is based on this link state routing.

Code:

```
#include <iostream>
using namespace std;
void dijkstra(int g[10][10],int n,int s)
{
   int cost[10][10],dist[10],pred[10];
   int v[10],count,min_dist,next_node;
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n;j++)
      {
         if(g[i][j]==0)
         cost[i][j]=9999;
         else
         cost[i][j]=g[i][j];
      }
   }
   for(int i=0;i<n;i++)
   {
      dist[i]=cost[s][i];
      pred[i]=s;
      v[i]=0;
   }
   dist[s]=0;
   v[s]=1;
   count=1;
   while(count<n-1)
   {
      min_dist=9999;
      for(int i=0;i<n;i++)
      {
         if(dist[i]<min_dist && !v[i])
         {
            min_dist=dist[i];
            next_node=i;
         }
```

```
        }
        v[next_node]=1;
        for(int i=0;i<n;i++)
        {
           if(!v[i])
              {
                 if(min_dist+cost[next_node][i]<dist[i])
                 {
                    dist[i]=min_dist+cost[next_node][i];
                    pred[i]=next_node;      }          }         }
        count++;
    }
    for(int i=0;i<n;i++)
    {
       if(i!=s)
       {
          cout<<"\ndist of node "<<i<<"= "<<dist[i];
          cout<<"\npath= "<<i;
          int j=i;
          do{
             j=pred[j];
             cout<<"<-"<<j;
          }while(j!=s); }            }          }
int main()
{
    int g[10][10],n,u;
    cout<<"enter no. of vertices:";
    cin>>n;
    cout<<"enter matrix\n";
    for(int i=0;i<n;i++)
    {        for(int j=0;j<n;j++)
                    cin>>g[i][j];
    }
    cout<<"enter staring node:";
    cin>>u;
    dijkstra(g,n,u);
}
```

Output:

```
enter no. of vertices:5
enter matrix
1 2 3 4 5
5 4 3 2 1
0 1 0 1 0
1 0 1 0 1
0 4 5 6 0
enter staring node:4

dist of node 0= 7
path= 0<-3<-4
dist of node 1= 4
path= 1<-4
dist of node 2= 5
path= 2<-4
dist of node 3= 6
path= 3<-4

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

Experiment 4:

Aim: To study the Basic Networking Commands

a. Stop and Wait protocol

Theory:

Stop-and-wait ARQ, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order. It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time; it is a special case of the general sliding window protocol with transmit and receive window sizes equal to one and greater than one respectively. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement (ACK) signal. After receiving a valid frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again. The timeout countdown is reset after each frame transmission. The above behaviour is a basic example of Stop-and-Wait. However, real-life implementations vary to address certain issues of design.

Code:

```
#include  <iostream>
#include   <unistd.h>
using namespace std;
int main()
{
   int x,x1=10,x2;
   int frame = 0;
   int frame_num = 10;
   cout<<"\nnumber of frames is:"<<frame;
   while(frame < frame_num)
   {
      cout<<"\nsending frame:"<<frame;
      x = rand()%10;
      if(x%10 == 0)
      {
         cout<<"waiting for 1 second\n";
         sleep(1);
         cout<<"\nsending frame :"<<frame<<"\n";
      }
      cout<<"\nack for frame :"<<frame<<"\n";
      frame++;
   }
   printf("\n end of stop and wait protocol\n");
   return 0;
}
```

Output:

```
number of frames is:0
sending frame:0
ack for frame :0

sending frame:1
ack for frame :1

sending frame:2
ack for frame :2

sending frame:3
ack for frame :3

sending frame:4
ack for frame :4

sending frame:5
ack for frame :5

sending frame:6
ack for frame :6

sending frame:7
ack for frame :7

sending frame:8
ack for frame :8

sending frame:9
ack for frame :9

 end of stop and wait protocol


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

b. Go back N Protocol

Theory:

Go–Back–N ARQ uses the concept of protocol pipelining, i.e. sending multiple frames before receiving the acknowledgment for the first frame. The frames are sequentially numbered and a finite number of frames. The maximum number of frames that can be sent depends upon the size of the sending window. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

Code:

```
#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{
```

```
    int w,i,f,frames[50];
    int start=0, ack;
    cout<<"Enter window size:";
    cin>>w;
    cout<<"\nEnter number of frames to transmit:";
    cin>>f;
    cout<<"\nEnter "<<f<<" frames:";
    for(i=0;i<f;i++)
       cin>>frames[i];
    while(ack != f)
    {
       cout<<"\nTransmitted frames:";
       for(i=start;i<(start+w);i++)
       {
          if(i < f)
              cout<<frames[i]<<" ";
       }
       cout<<"\nEnter last acknowledgement:";
       cin>>ack;
       start = ack;
    }
    cout<<"\nAll frames sent and received\n";
    return 0;
}
```

Output:

```
Enter window size:3

Enter number of frames to transmit:10

Enter 10 frames:1 2 3 4 5 6 7 8 9 10

Transmitted frames:1 2 3
Enter last acknowledgement:2

Transmitted frames:3 4 5
Enter last acknowledgement:5

Transmitted frames:6 7 8
Enter last acknowledgement:6

Transmitted frames:7 8 9
Enter last acknowledgement:9

Transmitted frames:10
Enter last acknowledgement:9

Transmitted frames:10
Enter last acknowledgement:10

All frames sent and received


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

c. Selective Repeat Protocol

Theory:

Selective Repeat protocol provides for sending multiple frames depending upon the availability of frames in the sending window, even if it does not receive acknowledgement for any frame in the interim. The maximum number of frames that can be sent depends upon the size of the sending window. The receiver records the sequence number of the earliest incorrect or un-received frame .It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame. The sender continues to send frames that are in its sending window. Once it has sent all the frames in the window, it retransmits the frame whose sequence number is given by the acknowledgements. It then continues sending the other frames.

Code:

```
#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{

   int windowsize,sent=0,ack,i;

   cout<<"enter window size\n";

   cin>>windowsize;

   while(1)
```

```
    {
        for( i = 0; i < windowsize; i++)
        {
            cout<<"Frame "<<sent<<" has been transmitted\n";
            sent++;
            if(sent == windowsize)
            break;
        }
    cout<<"\nPlease enter the last Acknowledgement received\n";
    cin>>ack;
    if(ack == windowsize)
    break;
    else
    sent = ack;    }
    return 0;
}
```

Output:

```
enter window size
4
Frame 0 has been transmitted
Frame 1 has been transmitted
Frame 2 has been transmitted
Frame 3 has been transmitted

Please enter the last Acknowledgement received
3
Frame 3 has been transmitted

Please enter the last Acknowledgement received
2
Frame 2 has been transmitted
Frame 3 has been transmitted

Please enter the last Acknowledgement received
1
Frame 1 has been transmitted
Frame 2 has been transmitted
Frame 3 has been transmitted

Please enter the last Acknowledgement received
4


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

Experiment 5:

Aim: To demonstrate the Connection of computers in Local Area Network.

Code:

```
#include  <iostream>
using namespace std;
int main()
{
system("ipconfig");
return 0;
}
```

Output:

```
C:\Users\admin>("ipconfig")

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2402:3a80:d22:c0aa:10af:4e53:3ac2:8cc2
   Temporary IPv6 Address. . . . . . : 2402:3a80:d22:c0aa:bc4b:98a6:a901:7c2
   Link-local IPv6 Address . . . . . : fe80::10af:4e53:3ac2:8cc2%17
   IPv4 Address. . . . . . . . . . . : 192.168.77.187
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::247b:e5ff:fe1c:f395%17
                                       192.168.77.188

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

Experiment 6:

Aim: Configure Host IP, Subnet Mask and Default Gateway in a System in LAN (TCP/IP Configuration).

Theory:
Both MAC Address and IP Address are used to uniquely define a device on the internet. NIC Card's Manufacturer provides the MAC Address, on the other hand Internet Service Provider provides IP Address. MAC Address stands for Media Access Control Address and IP Address stands for Internet Protocol Address. MAC address is a six bytes hexadecimal address. IP address is a 4 bytes (IPv4) or 6 bytes (IPv6) address. MAC Address is a physical address whereas, IP Address is a logical address. The MAC Address of a device can be figured out using ARP Protocol. The IP address of a device is figured out using RARP protocol. MAC Addresses can't be found easily by third parties. MAC Address helps in simply identifying the device. IP Address identifies the connection of the device on the network.

Code:

```
//To discover IP Address, subnet mask, default gateway of a system

#include <iostream>

int main()

{
        system("ipconfig");

        return 0;

}
```

```
C:\Users\admin>("ipconfig")

Windows IP Configuration


Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2402:3a80:d22:c0aa:10af:4e53:3ac2:8cc2
   Temporary IPv6 Address. . . . . . : 2402:3a80:d22:c0aa:bc4b:98a6:a901:7c2
   Link-local IPv6 Address . . . . . : fe80::10af:4e53:3ac2:8cc2%17
   IPv4 Address. . . . . . . . . . . : 192.168.77.187
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::247b:e5ff:fe1c:f395%17
                                       192.168.77.188

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

Experiment 7:

Aim: Write a program to encrypt and decrypt a given message using substitution cipher method.

Introduction: Plain text is the message or data that can be readable by the sender, receiver or any third party. When the plain text is modified by using some algorithms or techniques, the resulting data or message is called ciphertext. In short, converting plain text, i.e. readable text, into non-readable text is called ciphertext.

Theory:
In Caesar cipher, the set of plain text characters is replaced by any other character, symbols or numbers. It is a very weak technique of hiding text. In Caesar's cipher, each alphabet in the message is replaced by a fixed number (key) places down.

Code:

```
#include<iostream>

using namespace std;

int main()

{

   char message[100];

   int key;

   cout<<"Enter a message to encrypt:";

   gets(message);

   cout<<"Enter key:";

   cin>>key;

   for(int i=0; message[i] != '\0';i++)

   {

      if(message[i] >= 'a' && message[i] <= 'z')

      {

         message[i] = message[i] + key;

         if(message[i] > 'z')

         {

            message[i] = message[i] - 'z' + 'a' - 1;

         }

      }

      else if(message[i] >= 'A' && message[i] <= 'Z')

      {

         message[i] = message[i] + key;
```

```
            if(message[i] > 'Z')
            {
                message[i] = message[i] - 'Z' + 'A' - 1;
            }
        }
    }
    cout<<endl<<"Encrypted message:"<<message<<endl;
    for(int i=0; message[i] != '\0'; i++)
    {
        if(message[i] >= 'a' && message[i] <= 'z')
        {
            message[i] = message[i] - key;
            if(message[i] < 'a')
            {
                message[i] = message[i] + 'z' - 'a' + 1;
            }
        }
        else if(message[i] >= 'A' && message[i] <= 'Z')
        {
            message[i] = message[i] - key;
            if(message[i] < 'A')
            {       message[i] = message[i] + 'Z' - 'A' + 1;          }       }    }
    cout<<endl<<"Decrypted message:"<<message;
    return 0;
}
```

Output:

```
Enter a message to encrypt:VARSHA78
Enter key:4

Encrypted message:ZEVWLE78

Decrypted message:VARSHA78

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

Experiment 8:

Aim: Write a program to implement Diffie-Hellman Algorithm.

Introduction:
The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

Theory:
The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say end user1 and end user2) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Code:

```
#include<iostream>
#include<math.h>

using namespace std;

double primi_root(double temp)
{
   double a,b,i=2;
   for(int j=1;j<temp;j++)
   {
      if(i!=temp)
      {
         a=pow(i,j);
         a=a-temp*int(a/temp);
         if(a==1)
         {
            i++;
            j=1;
         }
      }
   }
   if(a!=1)
      return i;
   return 0;
}
double Generate_key(double G,double a,double P)
{
   if(a==1)
      return G;
   double temp1;
   temp1=pow(G,a);
   temp1=temp1-P*int(temp1/P);
   return temp1;
}
```

```
int main()
{
   double prime_no,gen,x,y,privkey1,privkey2,secrkey1,secrkey2;
   cout<<"Enter the prime number:";
   cin>>prime_no;
   gen=primi_root(prime_no);
   if(gen==0)
      cout<<"No primitive root";
   else
   {
      cout<<"Primitive root="<<gen<<endl<<endl;
      cout<<"Enter the private key of person A:";
      cin>>privkey1;
      cout<<"Enter the private key of person B:";
      cin>>privkey2;
      x=Generate_key(gen,privkey1,prime_no);
      cout<<endl<<"The public key of person A:"<<x<<endl;
      y=Generate_key(gen,privkey2,prime_no);
      cout<<"The public key of person B:"<<y<<endl;
      secrkey1=Generate_key(y,privkey1,prime_no);
      secrkey2=Generate_key(x,privkey2,prime_no);
      cout<<endl;
      cout<<"Secret Key of Person A="<<secrkey1<<endl;
      cout<<"Secret Key of Person B="<<secrkey2<<endl;
   }
   return 0;
}
```

Output:

```
Enter the prime number:23
Primitive root=5

Enter the private key of person A:4
Enter the private key of person B:3

The public key of person A:4
The public key of person B:10

Secret Key of Person A=18
Secret Key of Person B=18
```

PART B

Experiment 1:

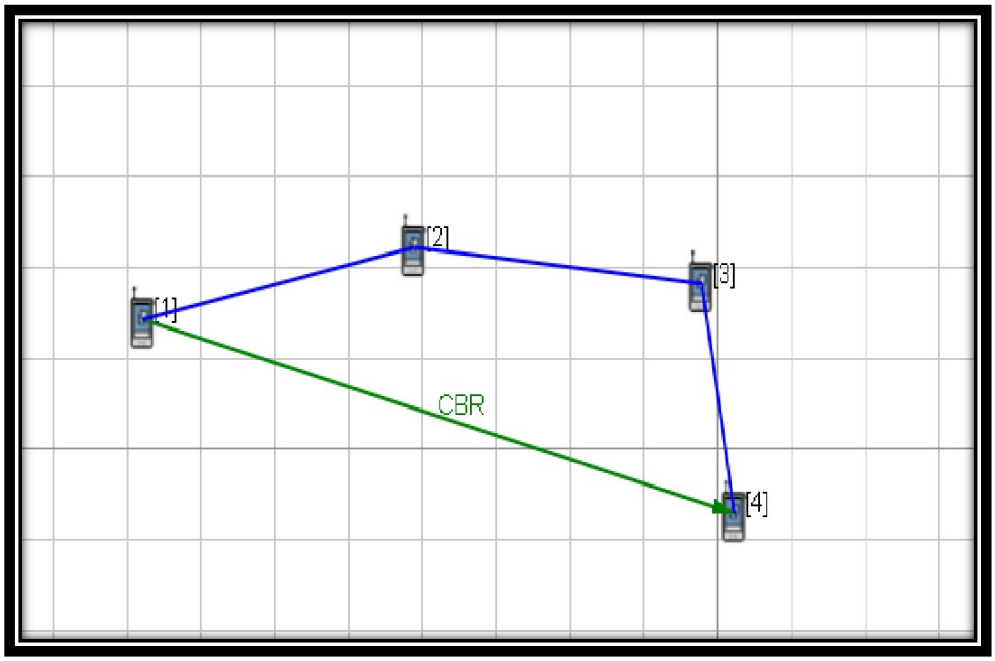
AIM: Configure network with the following topologies and analyze i) BUS ii) RING iii) Fully connected mesh topology, disable a node in each of the topologies and find the changes.

PROCEDURE:

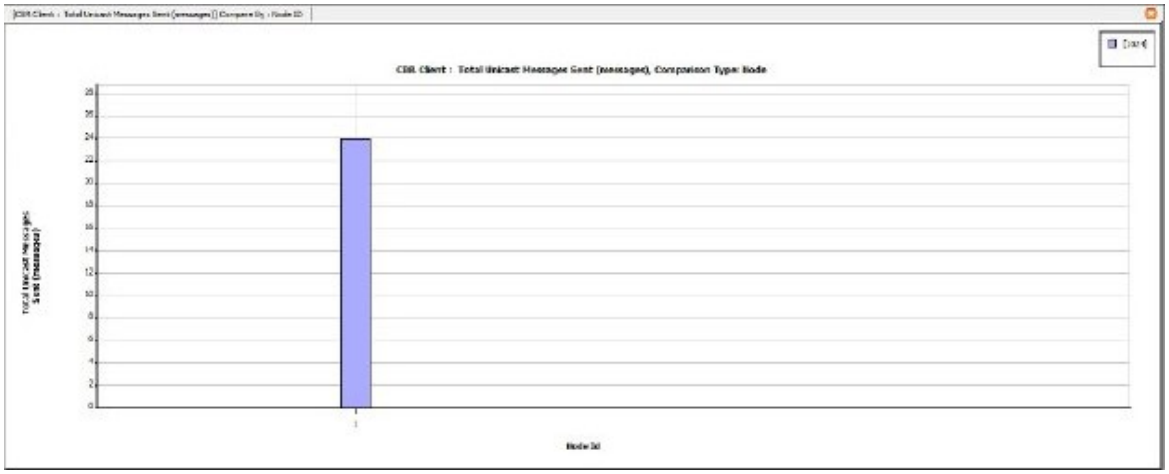
1. Select the nodes and connect them using link option.
2. Select CBR and connect the nodes which are to be observed as client and server.
3. After the connections are made save, simulate and run.
4. To deactivate a node after running, right click on the node that has to be deactivated and click on the deactivate option.
5. To analyse the throughput and other performance variables, click on analysis option.
6. Repeat the same for all topologies.

OBSERVATIONS:

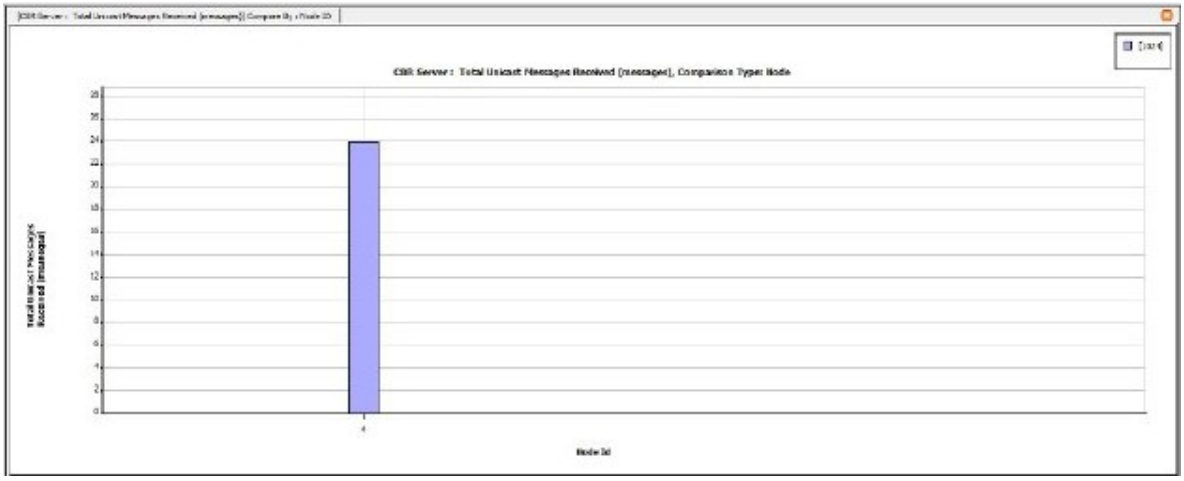
i) BUS



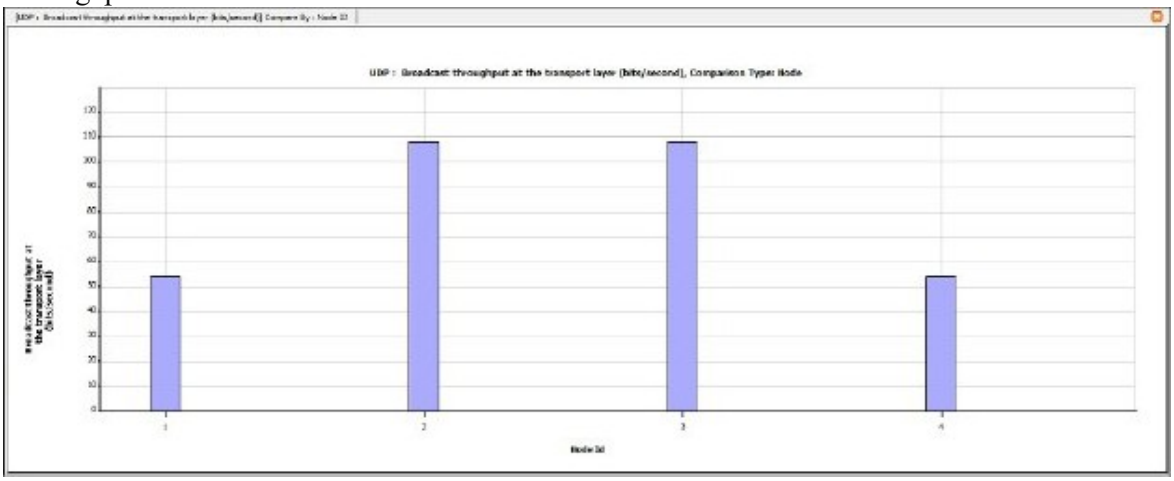
CBR Client



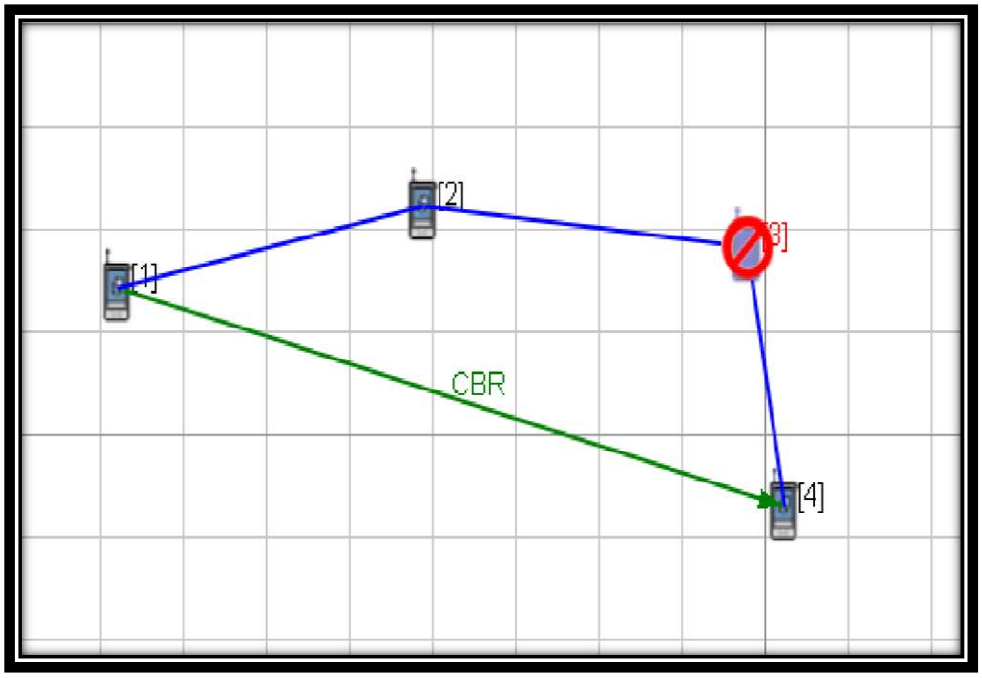
CBR Server



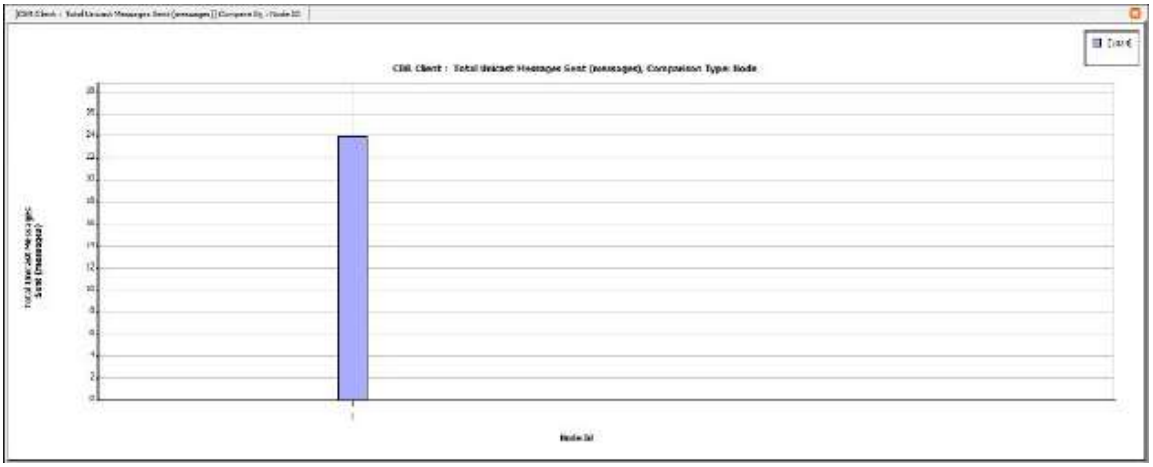
Throughput



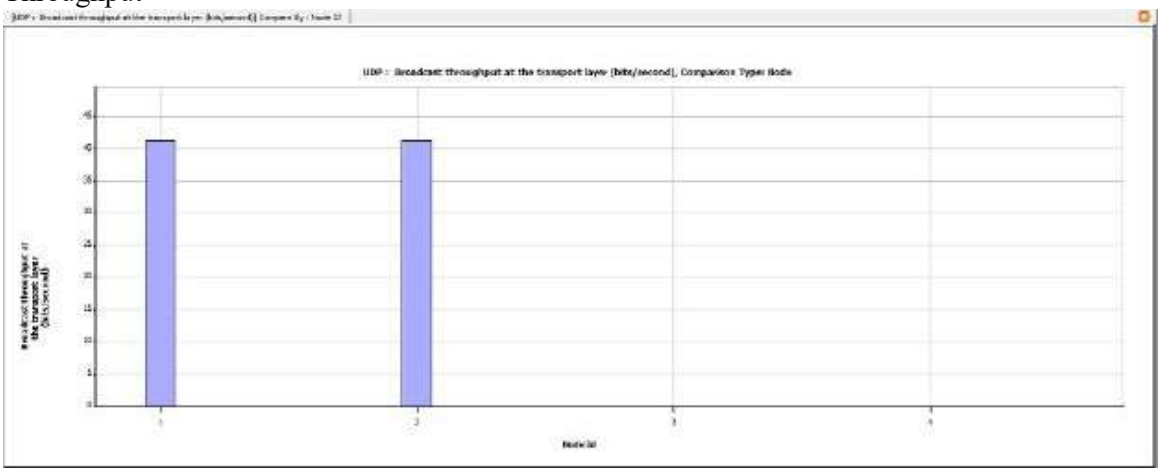
WITH DEACTIVATED NODE



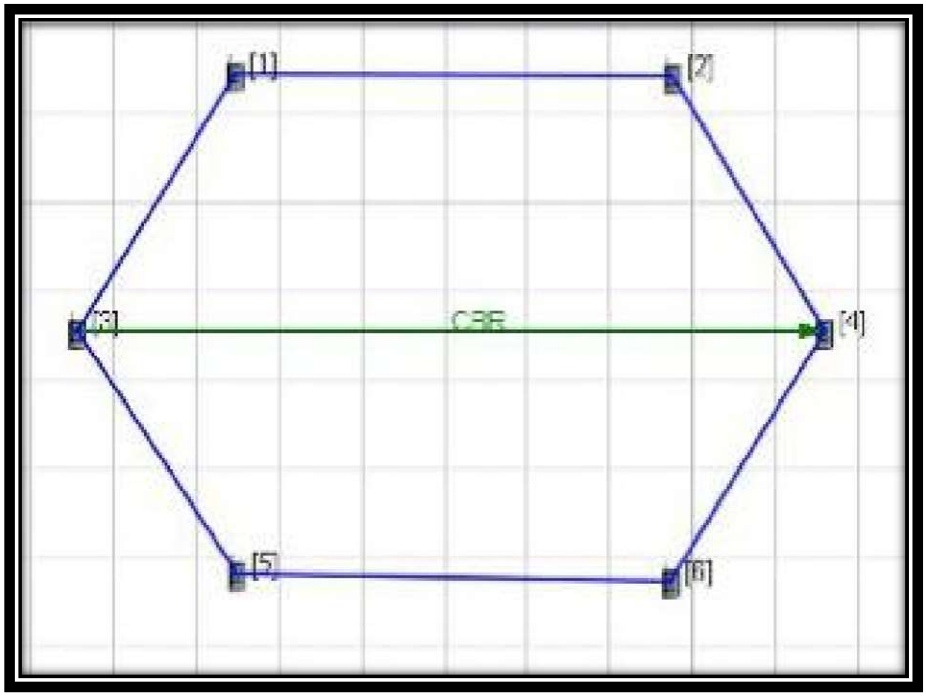
CBR Client



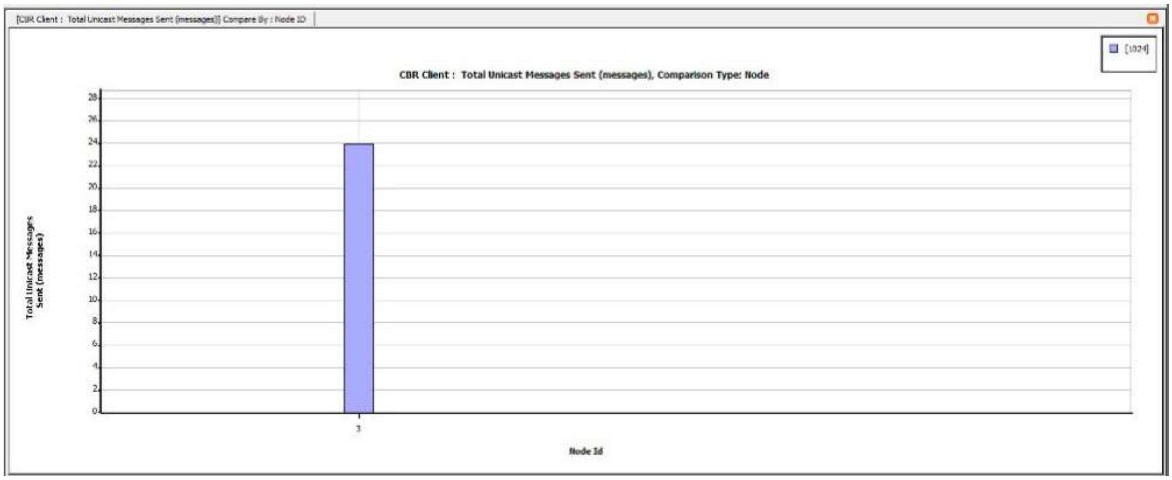
Throughput



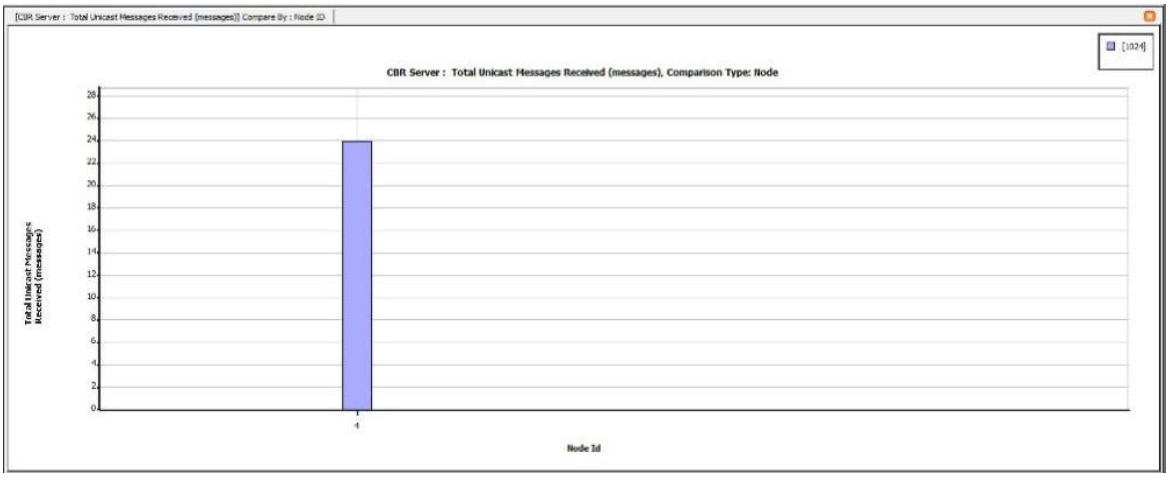
ii) RING



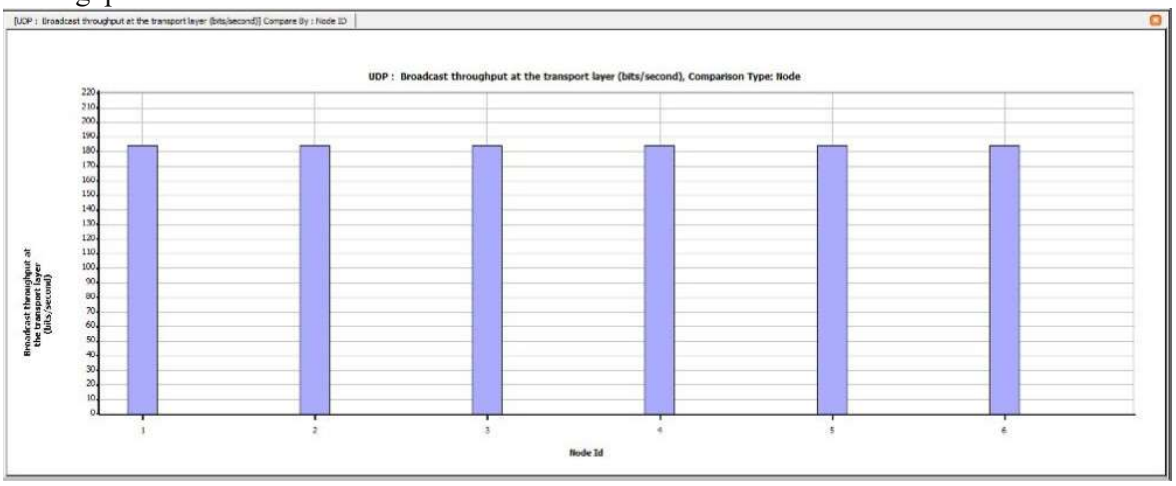
CBR Client



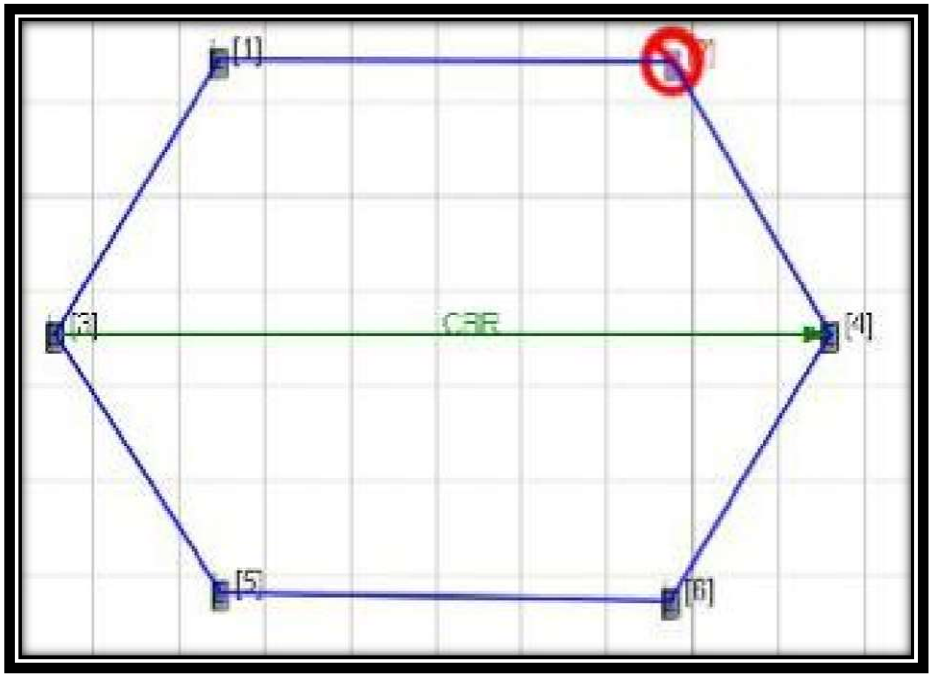
CBR Server



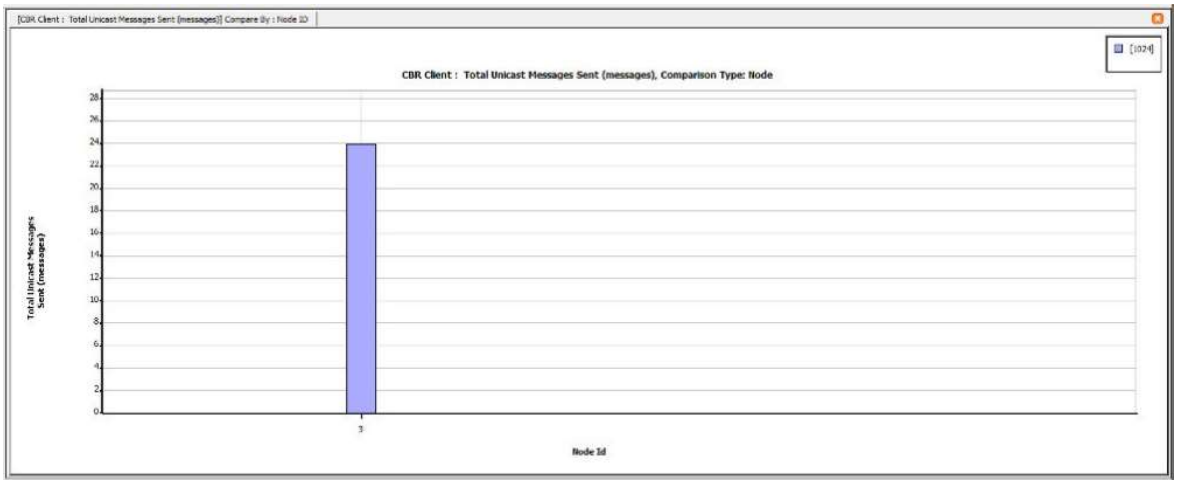
Throughput



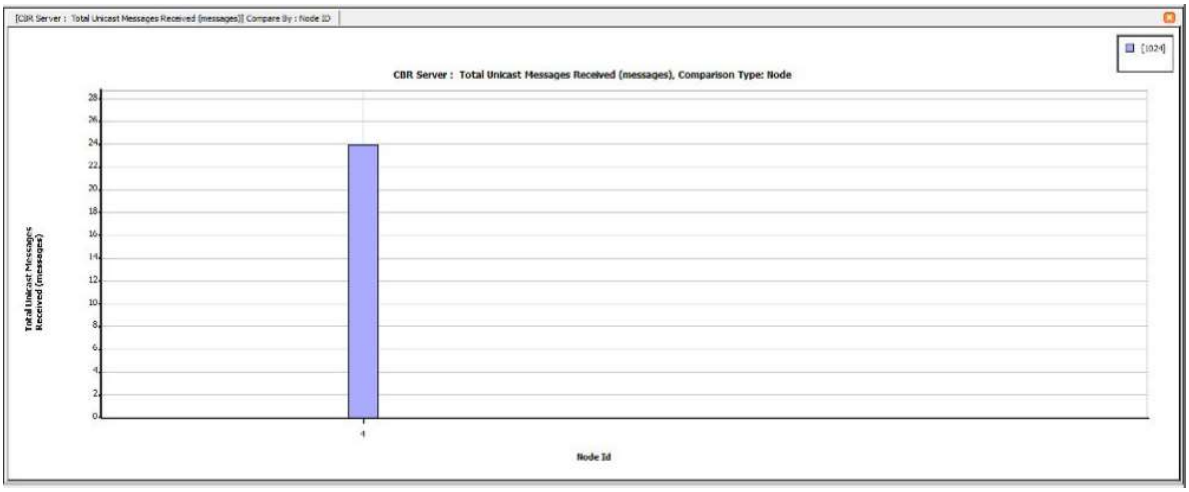
WITH DEACTIVATED NODE



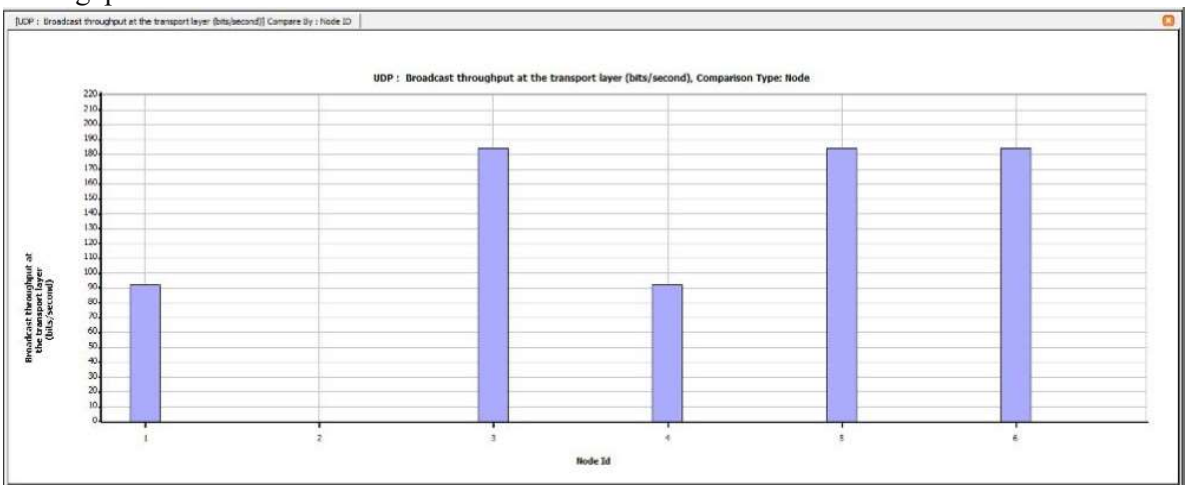
CBR Client



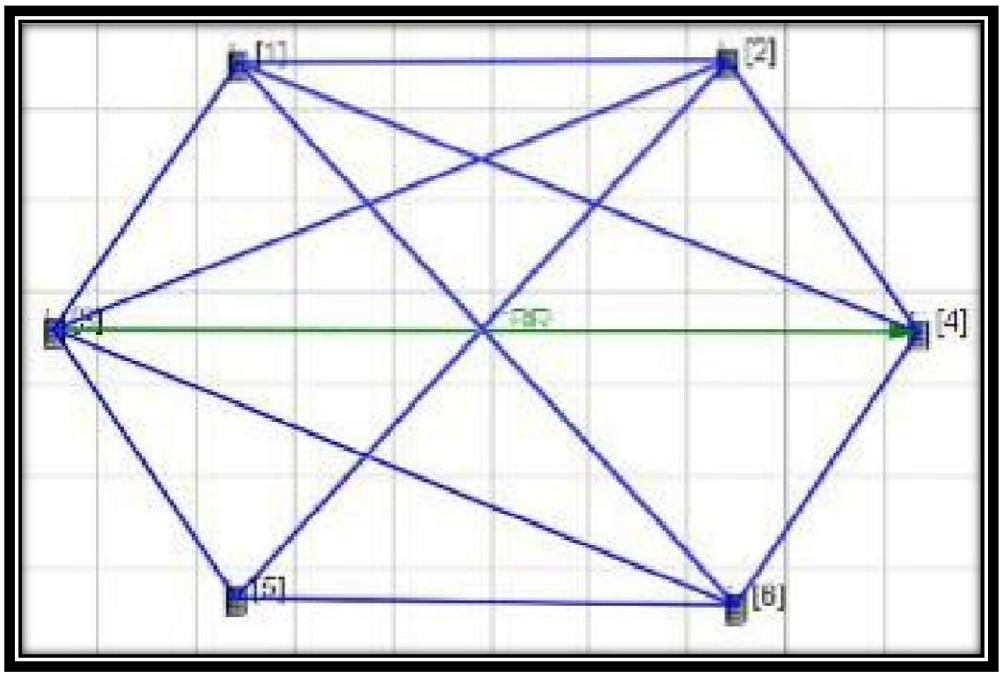
CBR Server



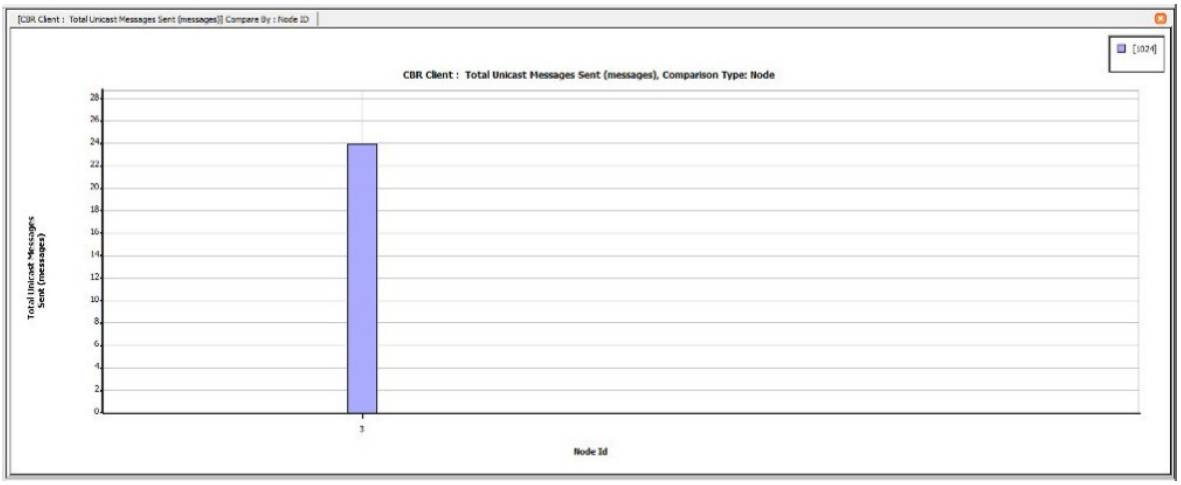
Throughput



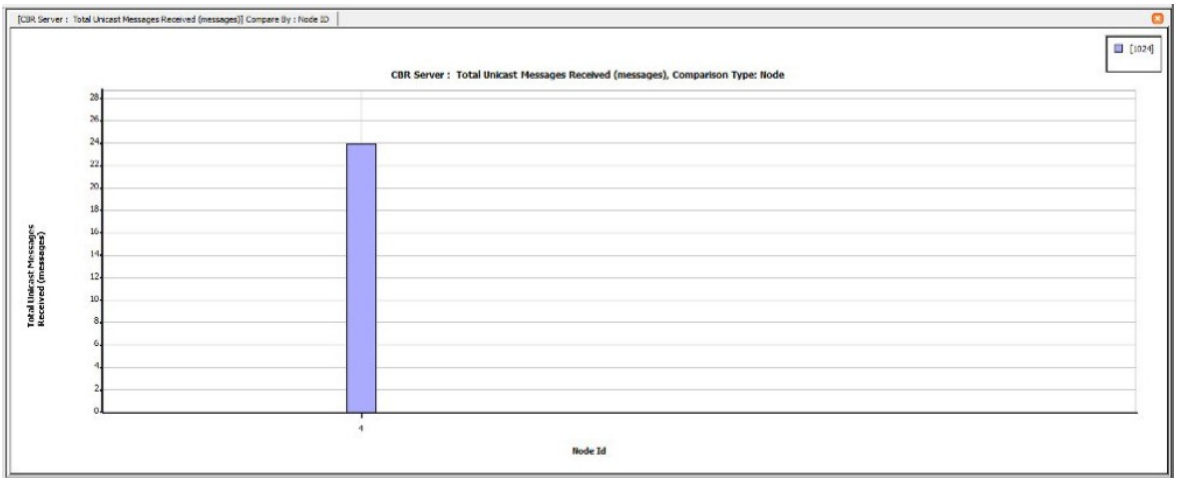
iii) FULLY CONNECTED MESH



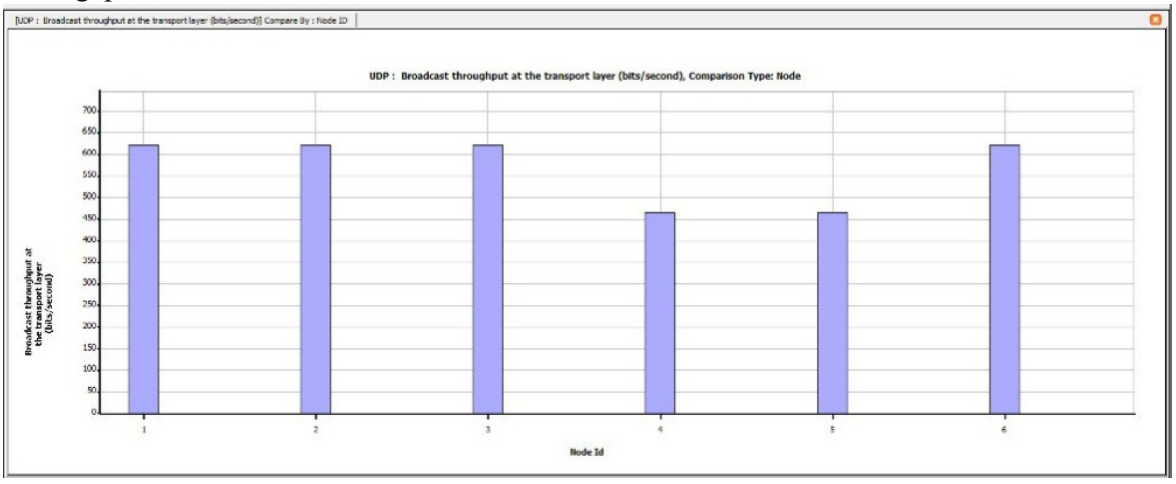
CBR Client



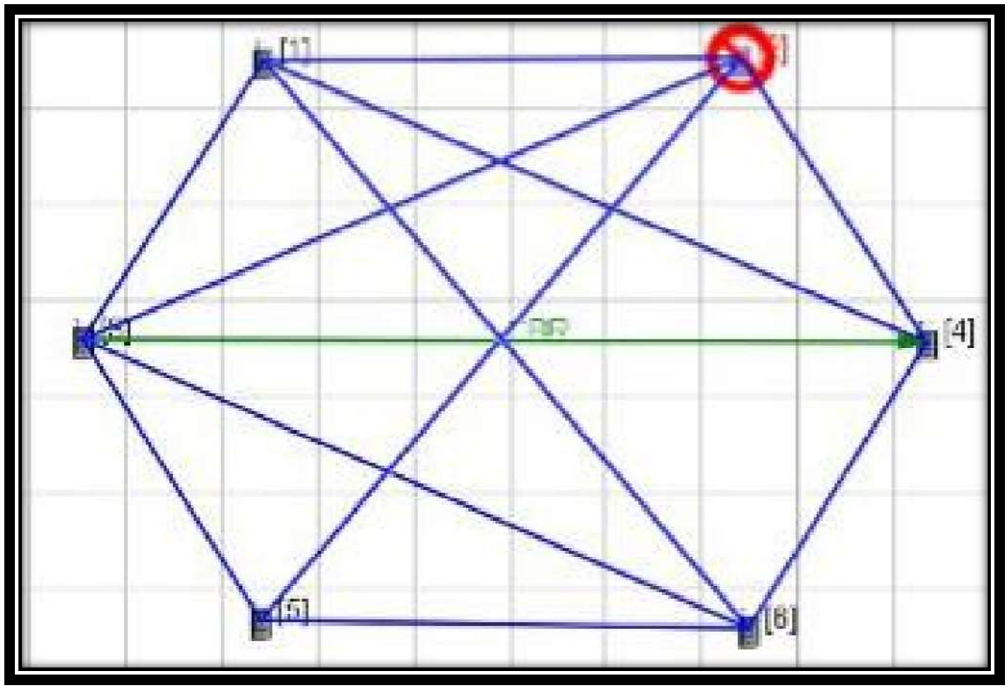
CBR Server



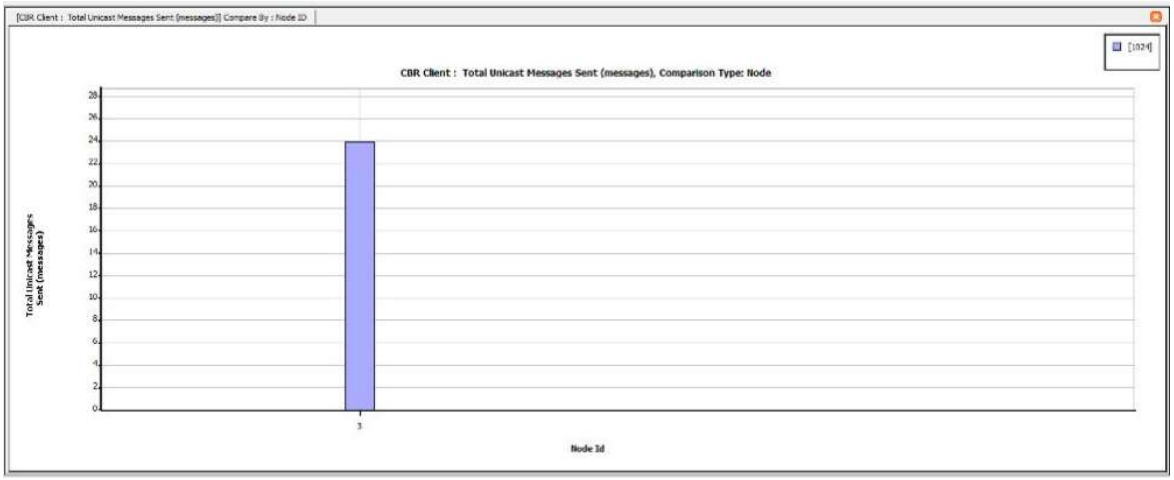
Throughput



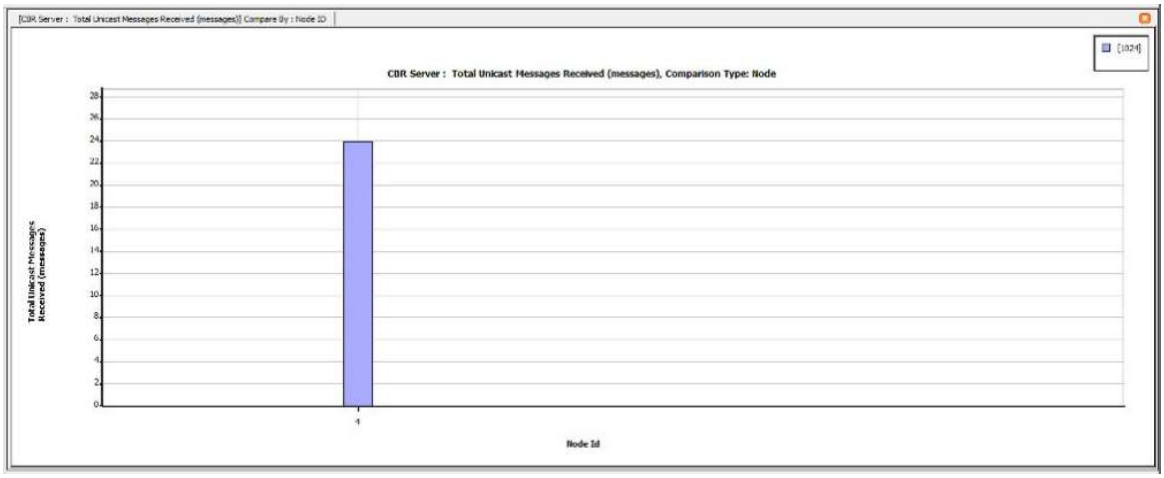
WITH DEACTIVATED NODE



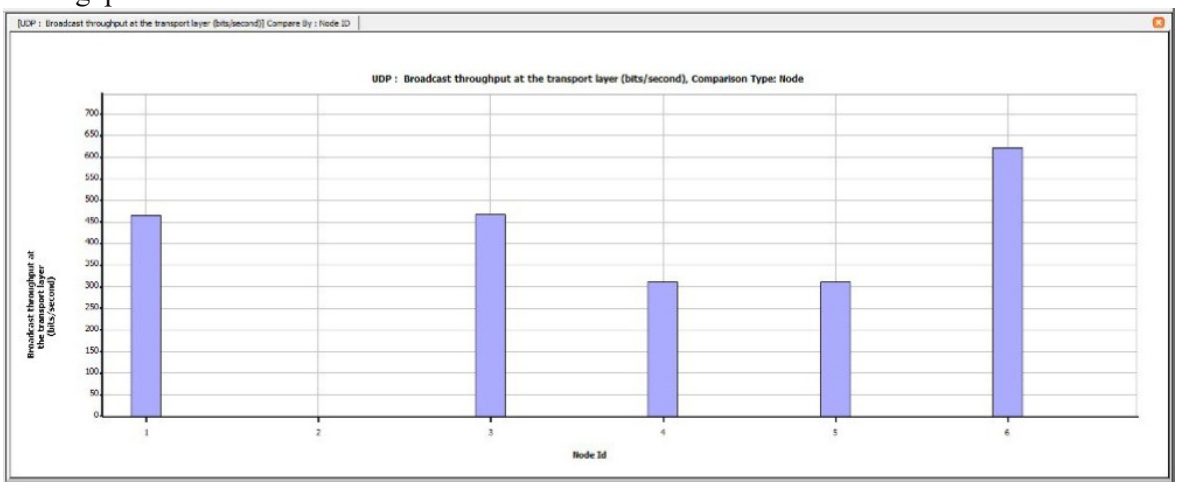
CBR Client



CBR Server



Throughput



RESULTS:

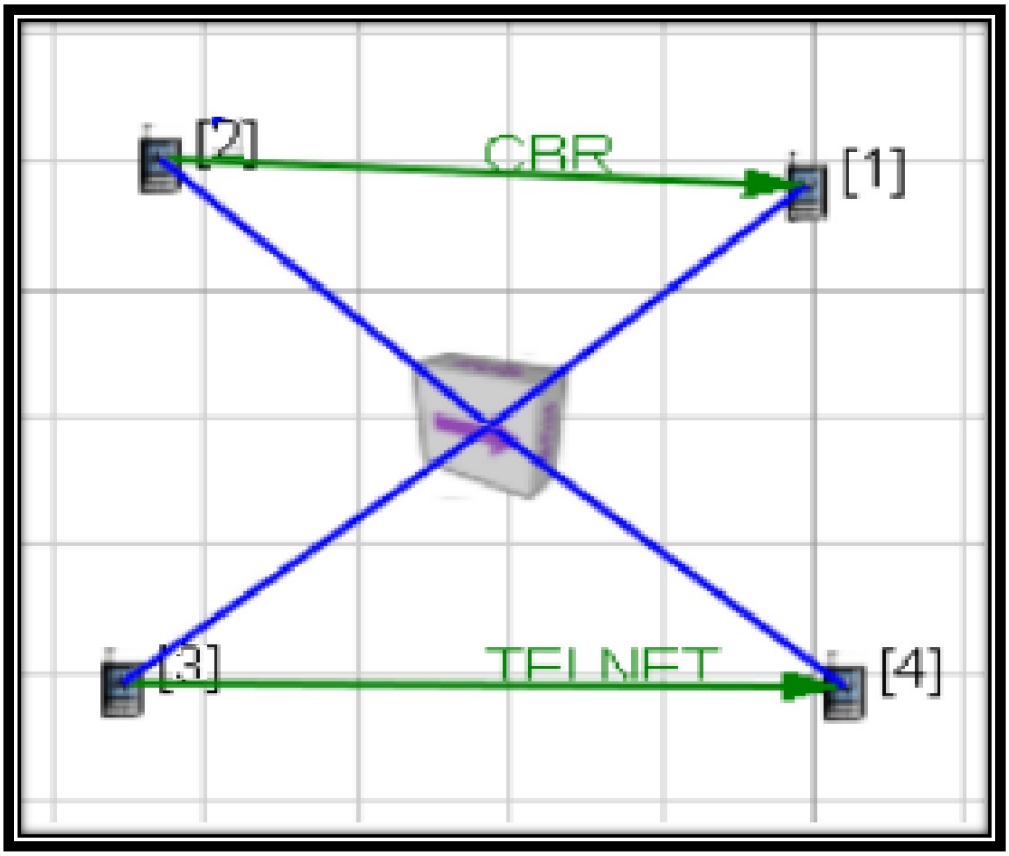
1. Bus topology does not work after one of the nodes is deactivated.
2. Ring topology works fine until the server or the client node is not deactivated as one or the other connecting nodes will complete the path.
3. Mesh topology works perfectly even when one of the nodes is deactivated as all of the nodes are connected to every other node.

Experiment 2:

AIM: Simulate Ethernet LAN with 4 nodes, apply relevant TCP and UDP applications and Determine:
i) The number of data packets sent by UDP and TCP
ii) Average jitter of UDP and TCP
iii) Number of periodic updates sent by the routing algorithm
iv) Number of ACK packets sent

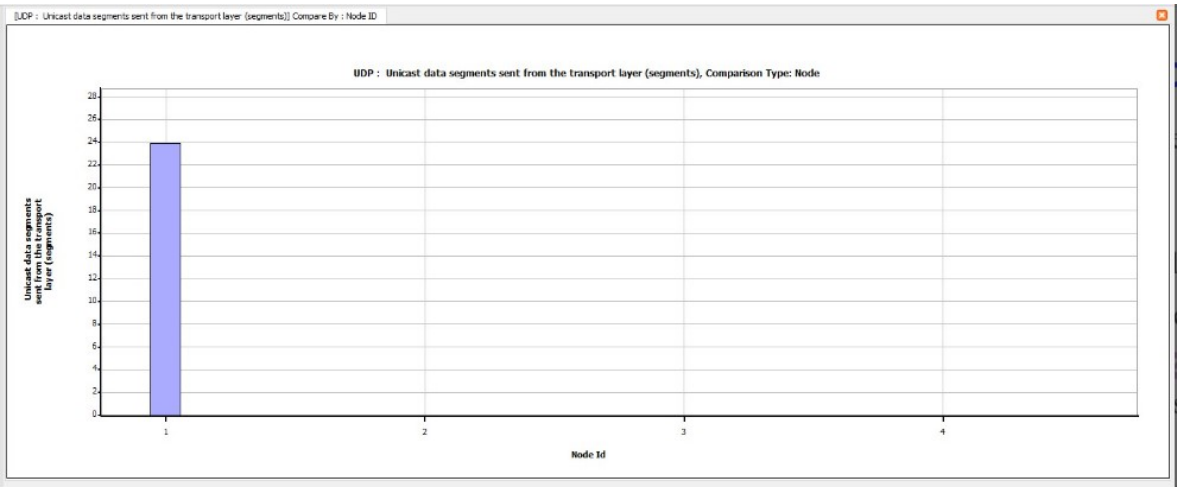
PROCEDURE:

1. Select the nodes and connect them to the hub using links.
2. Select CBR and also a TCP application such as FTP or TELNET and connect them as shown.
3. Save, simulate and run and check the performance parameters by going to analysis.
4. Select transport layer and appropriate application (UDP or TCP) to find the jitter and throughput.

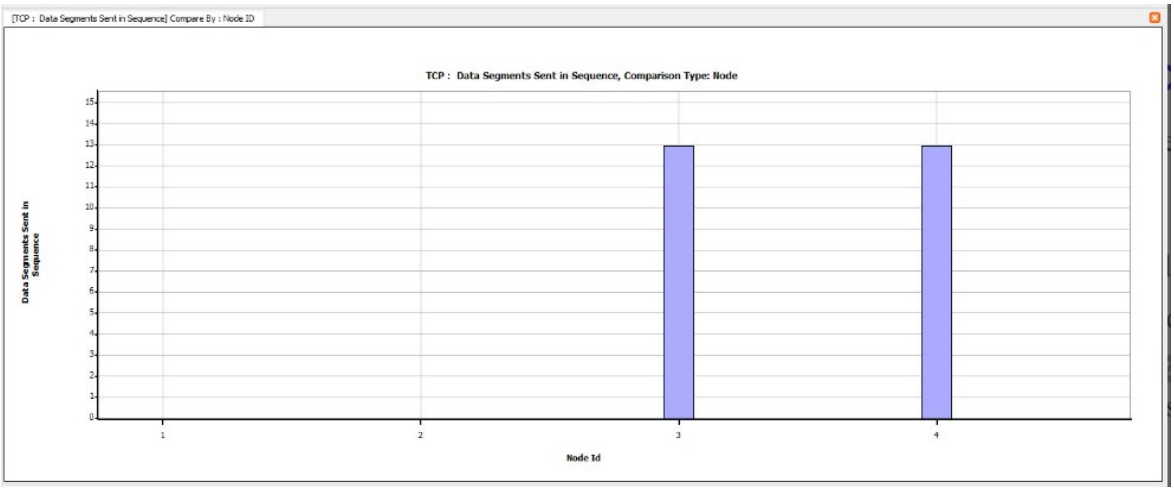


OBSERVATIONS:

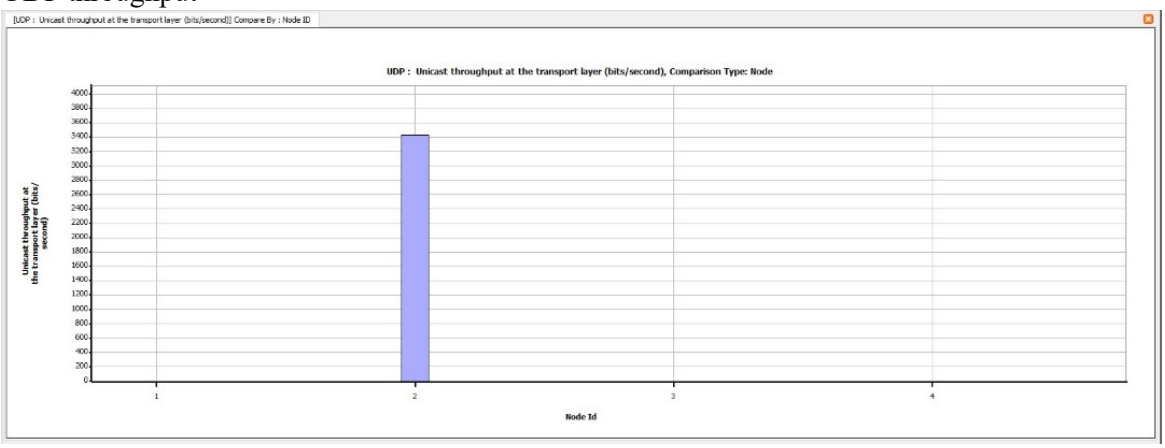
UDP data sent



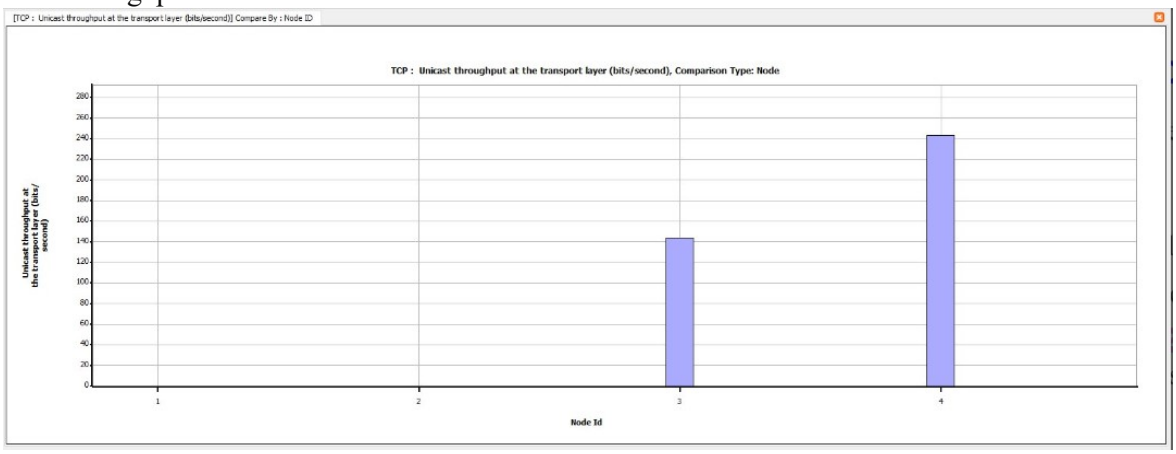
TCP data sent



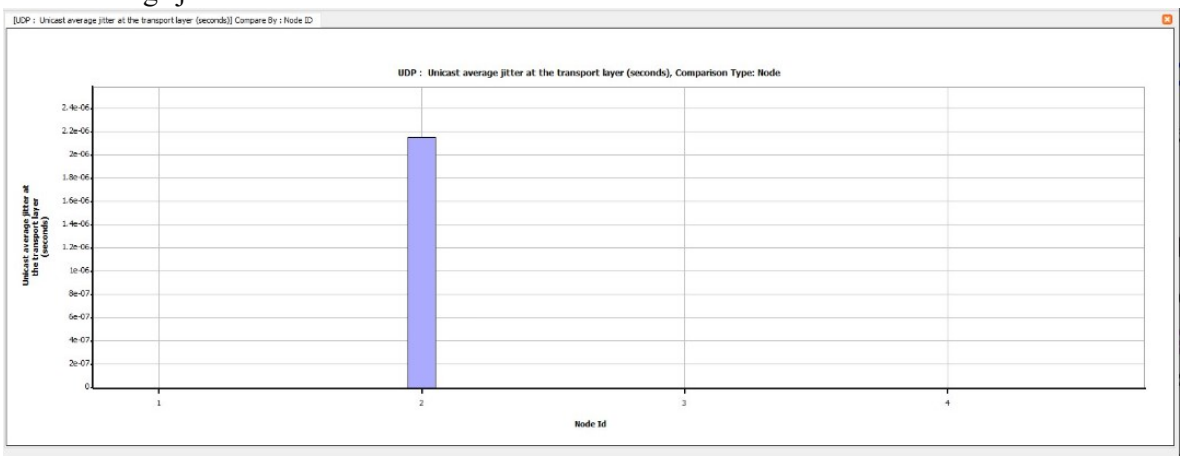
UDP throughput



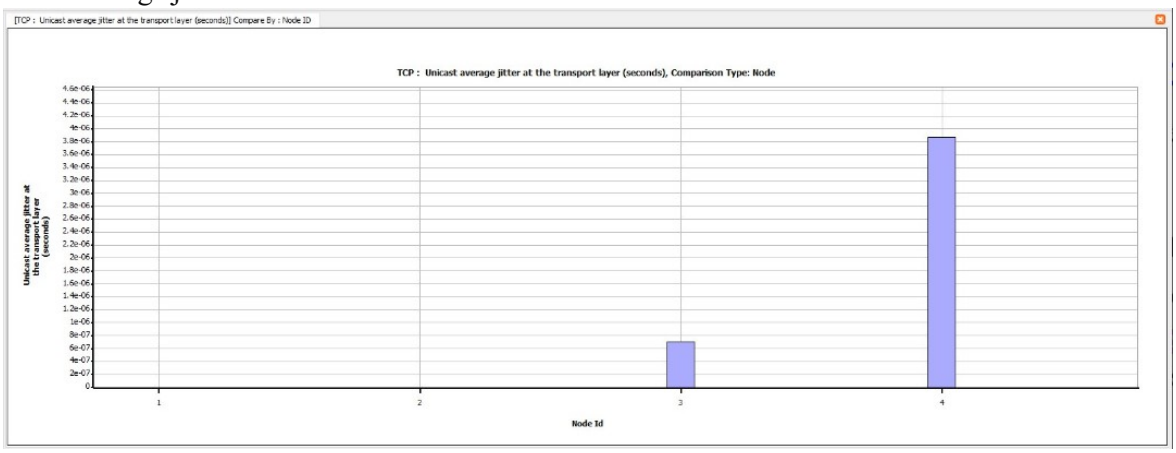
TCP throughput



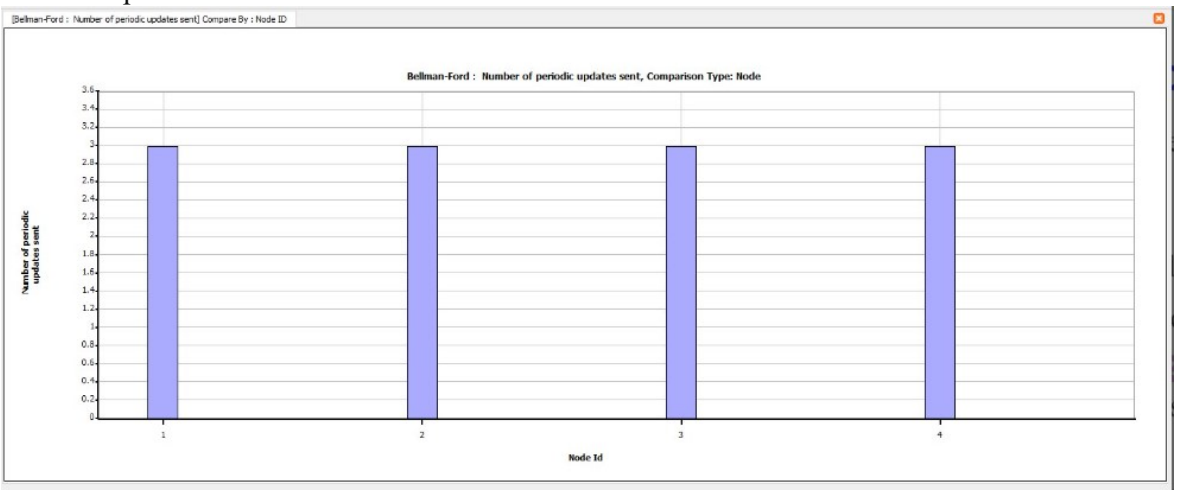
UDP Average jitter



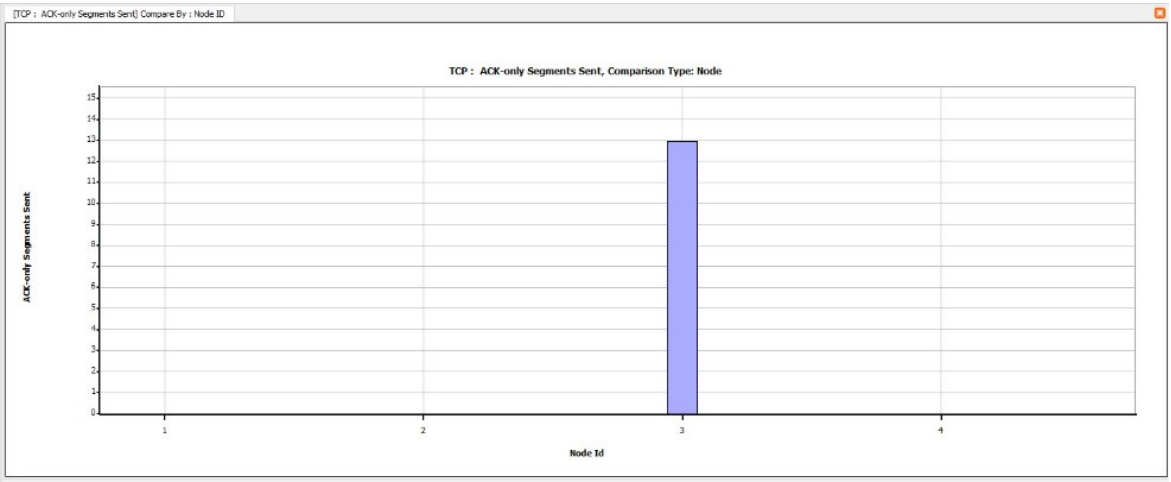
TCP Average jitter



Periodic updates



TCP ACK sent



RESULTS:

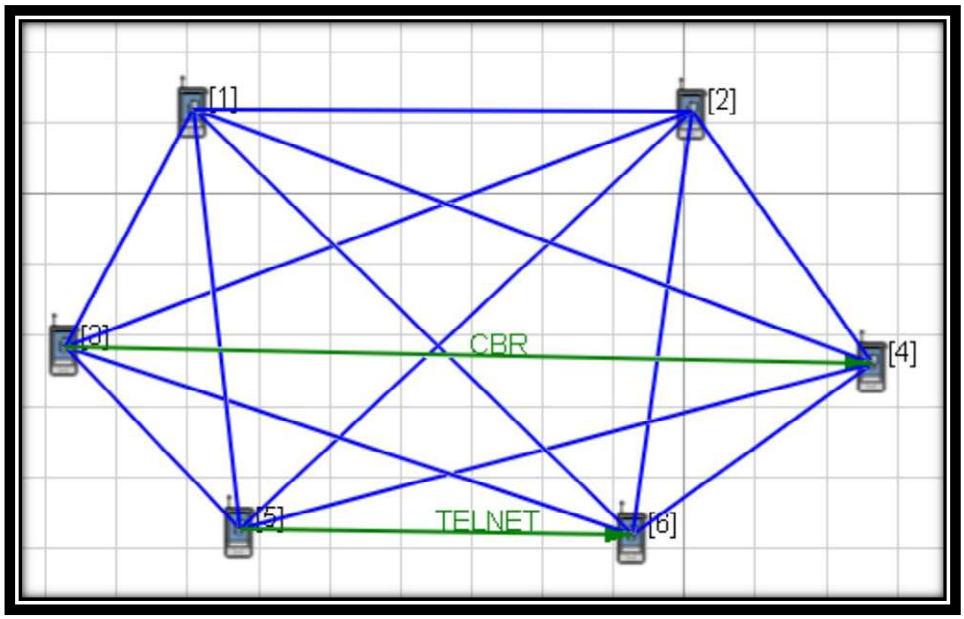
Average number of packet sent in UDP : 24 segments
Average number of packet sent in TCP : 13 segments
Unicast average throughput in UDP : 3400 bits/sec
Unicast average throughput in TCP : 240 bits/sec
Unicast Average jitter in UDP : 2.2 e-06
Unicast Average jitter in TCP : 4 e-06
Number of ACK packet sent in TCP : 13 segments

Experiment 3:

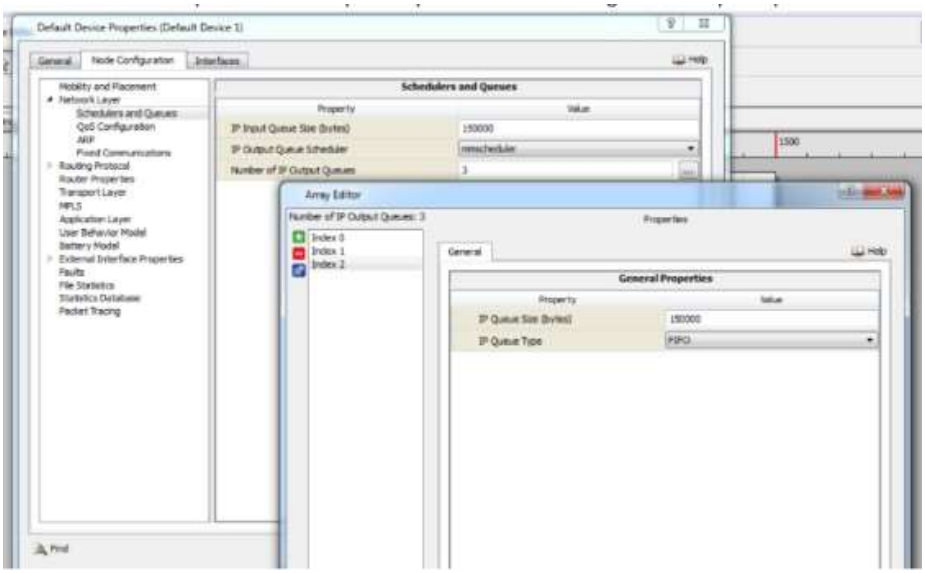
AIM: Simulate a network of N nodes with point to point connection; apply TCP and UDP applications vary the queue size and bandwidth and find
i) Number of packets dropped due to queue overflow
ii) Average hop count for data packets
iii) Average delay and jitter.
iv) Apply FTP and TELNET traffic between the nodes of the above network and analyse the throughput.

PROCEDURE:

1. Make connections as shown.

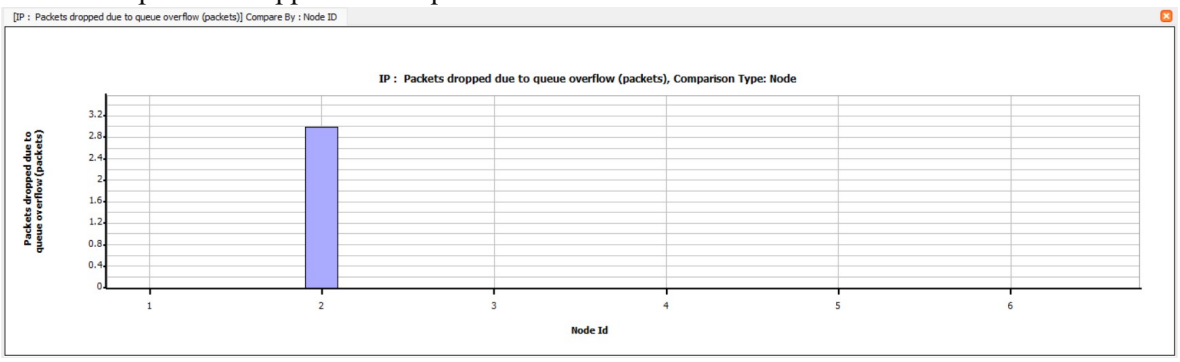


2. Select a node for which queue size needs to be varied.
3. Select the interface, go to properties select interfaces.
4. Go to network layer choose the option: queues and scheduling and modify the queue size.

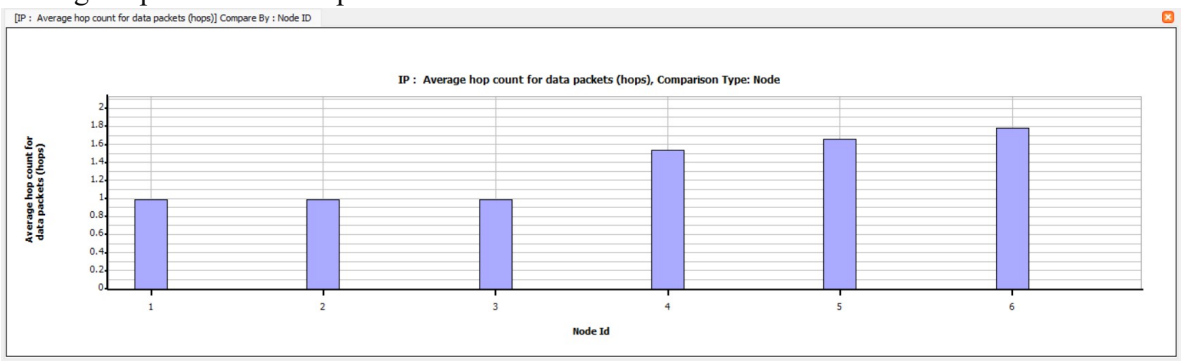


OBSERVATIONS:

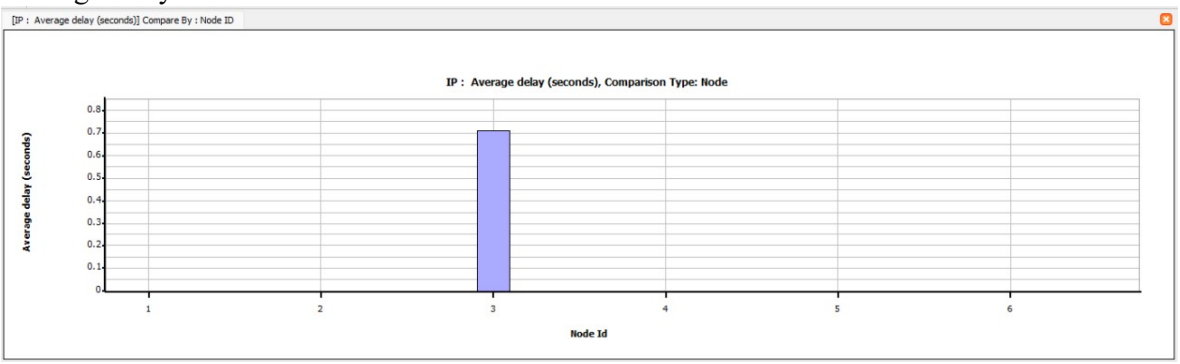
Number of packets dropped due to queue overflow



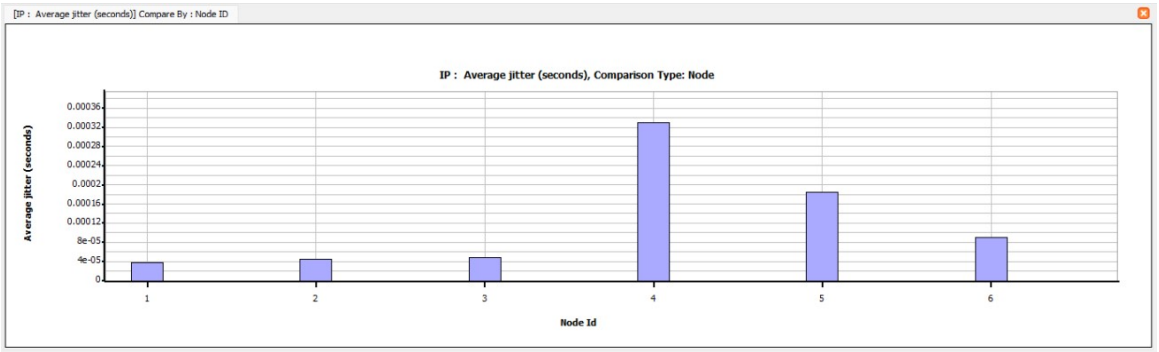
Average hop count for data packets



Average delay



Average jitter



RESULTS:

Number of packets dropped due to queue overflow: 3 packets at node 2

Average hop count for data packets

Node id	number of packets
1	1
2	1
3	1
4	1.5
5	1.7
6	1.8

Average delay: 0.7 seconds

Average jitter

Node id	number of packets
1	4 e-05
2	4.5 e-05
3	5 e-05
4	0.00032
5	0.00018
6	9 e-05

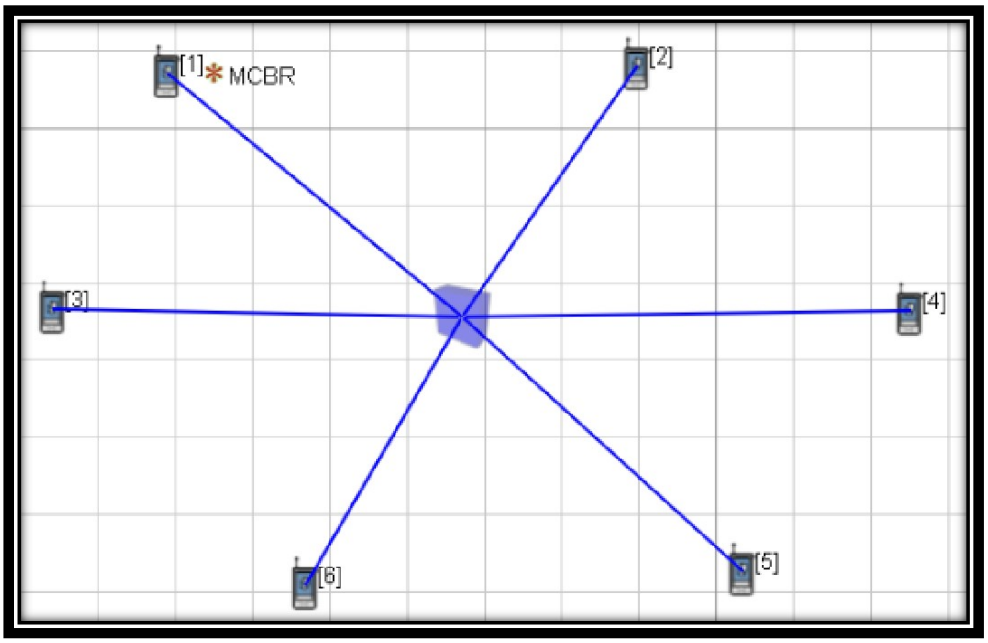
Experiment 4:

AIM: Simulate Ethernet LAN with N nodes, configure multicast traffic and Determine

i) The total multicast data bytes received
ii) Total multicast data bytes transmitted
iii) Multicast average delay at the transport layer for UDP
iv) Packets sent by DVMRP
v) Neighbors for every node as determined by DVMRP
vi) Packets dropped due to expired TTL
vii) Packets dropped due to no route.

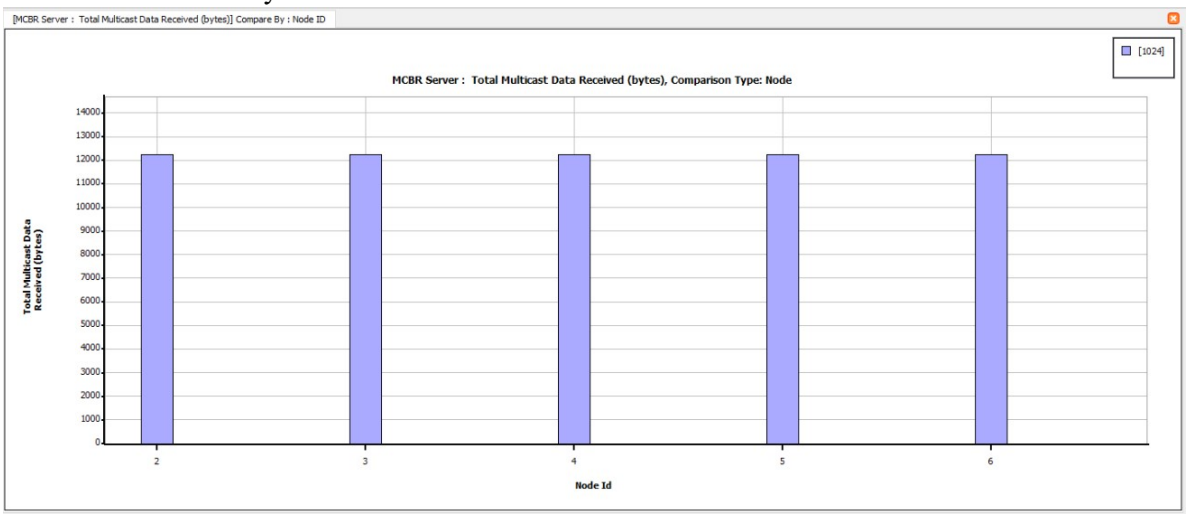
PROCEDURE:

1. Select HUB, select routing protocol under node configuration tab.
2. Set enable multicast field to yes.
3. Group management protocol to IGMP.
4. Multicast routing protocol to DVMRP.
5. Update router list with the node id of the sender.
6. Select multicast group editor from tool.
7. Add the receiving nodes.

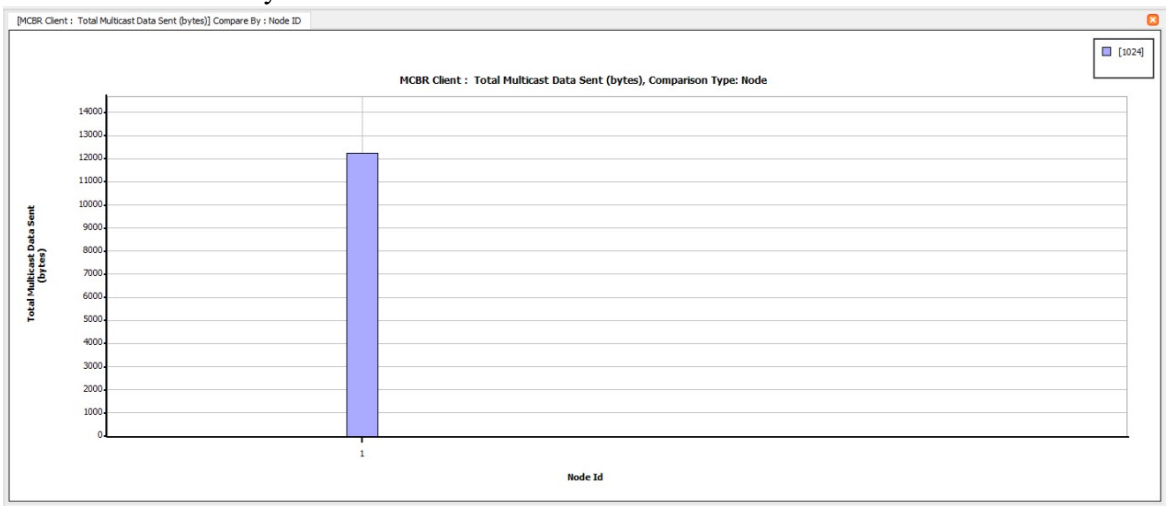


OBSERVATIONS:

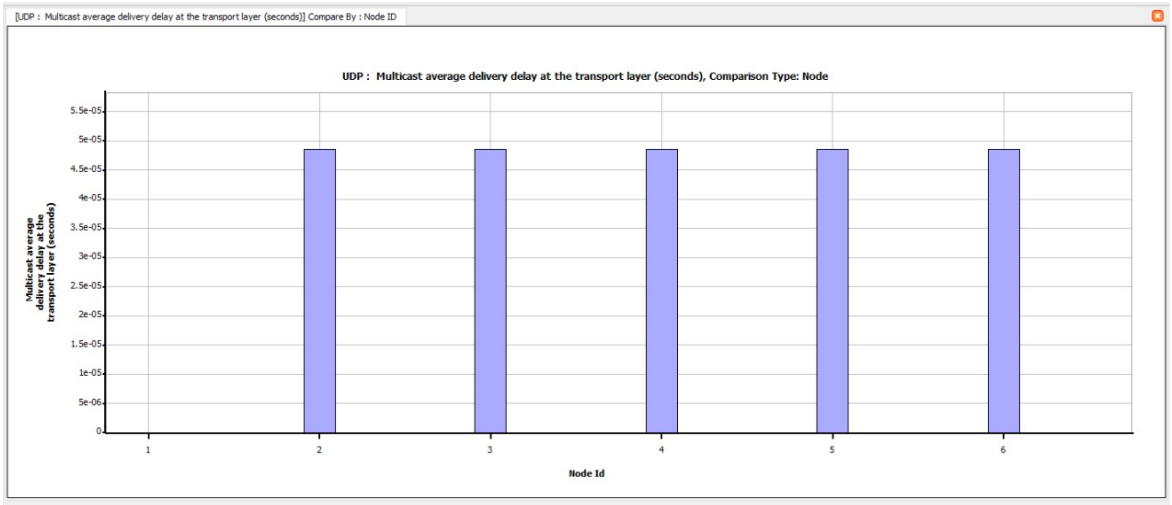
Total multicast data bytes received



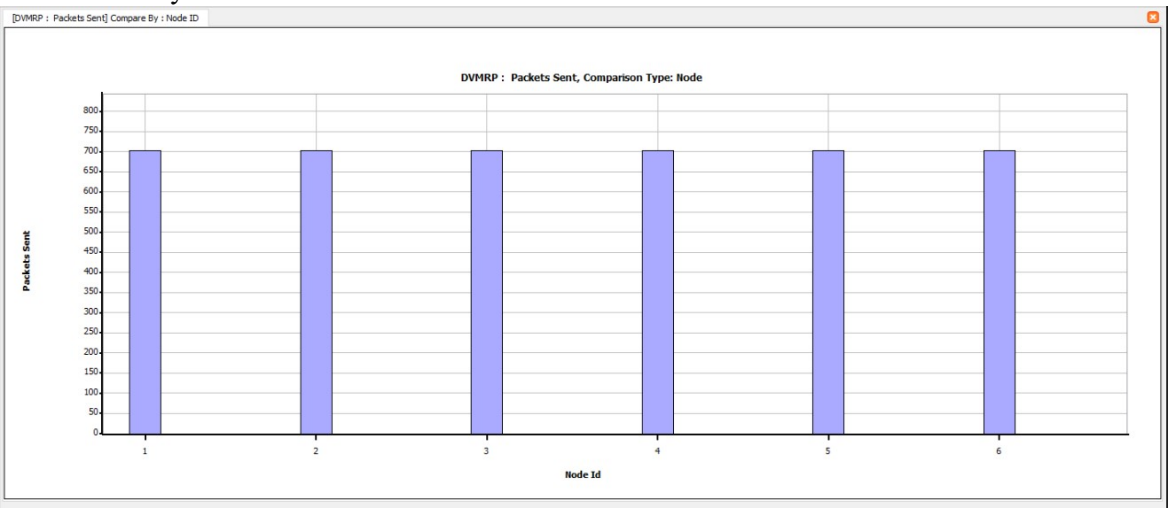
Total multicast data bytes sent



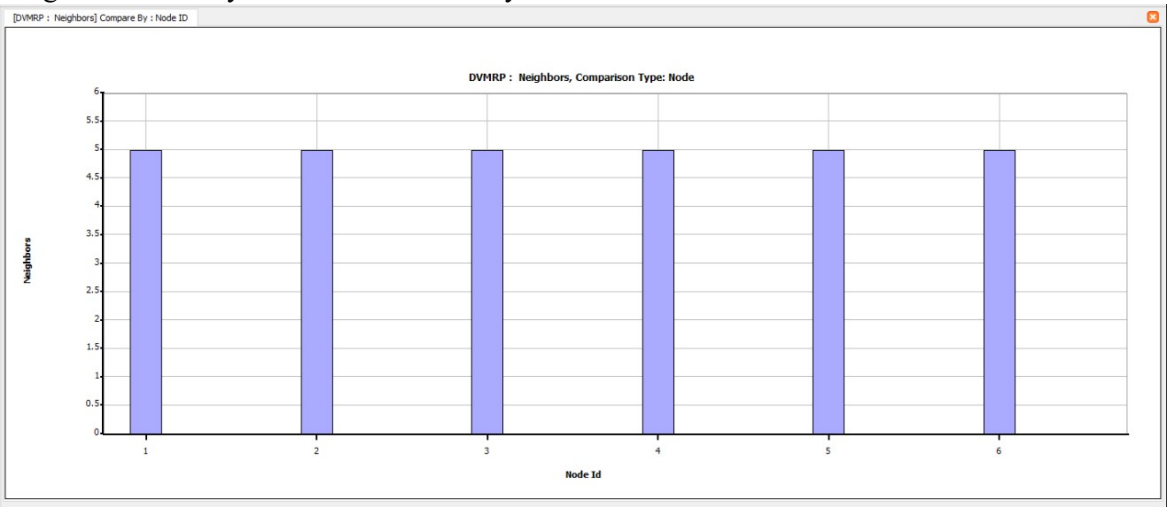
Multicast average delay at the transport layer for UDP



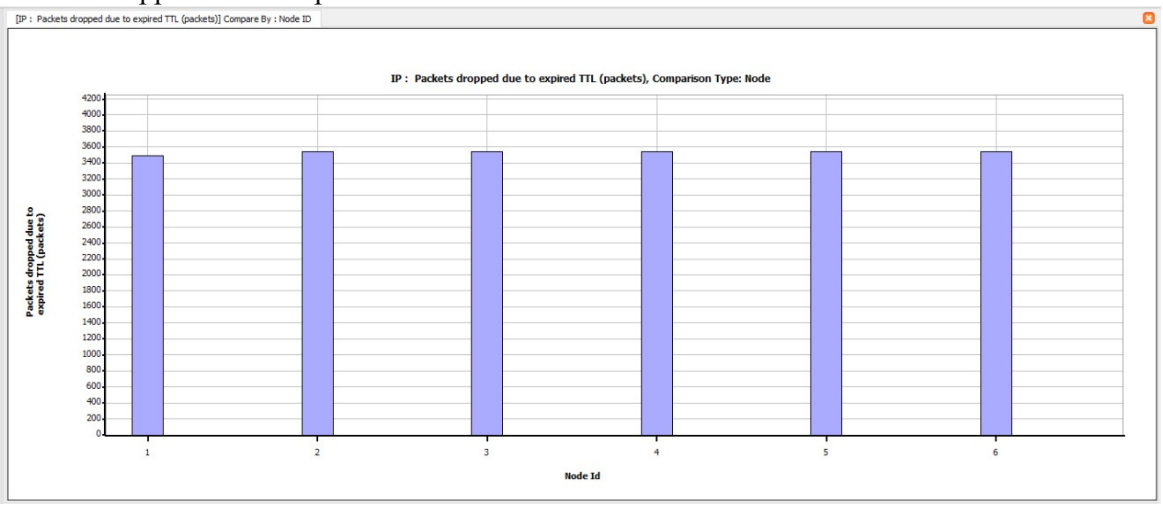
Packets sent by DVMRP



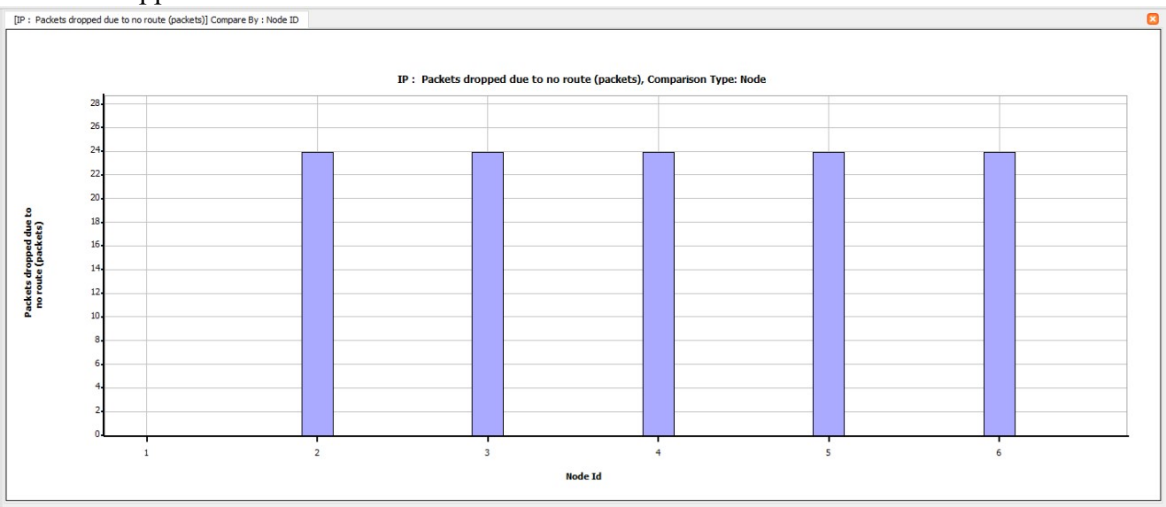
Neighbors for every node as determined by DVMRP



Packets dropped due to expired TTL



Packets dropped due to no route



RESULTS:

The outputs are

Total multicast data bytes received

Node id	number of packets
1	12100
2	12100
3	12100
4	12100
5	12100
6	12100

Total multicast data bytes sent

Node id	number of packets
1	0
2	12100
3	0
4	0
5	0
6	0

Multicast average delay at the transport layer for UDP

Node id	number of packets
1	
2	4.8 e-05
3	4.8 e-05
4	4.8 e-05
5	4.8 e-05
6	4.8 e-05

Packets sent by DVMRP

Node id	number of packets
1	700
2	700
3	700
4	700
5	700
6	700

Neighbors for every node as determined by DVMRP

Node id	number of packets
1	5
2	5
3	5
4	5
5	5
6	5

Packets dropped due to expired TTL

Node id	number of packets
1	3500
2	3500
3	3500
4	3500
5	3500
6	3500

Packets dropped due to no route

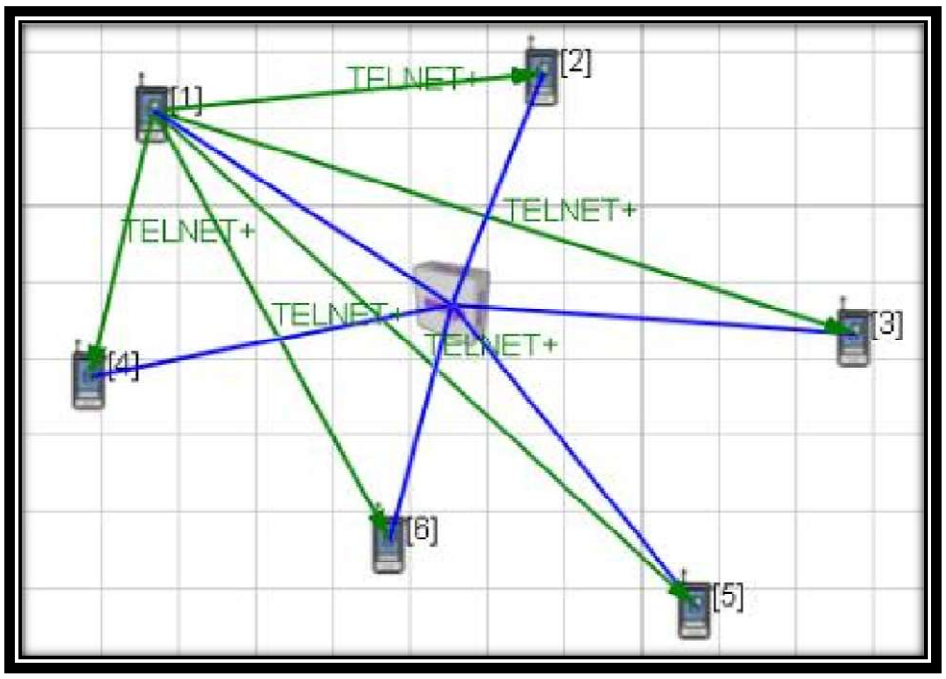
Node id	number of packets
1	24
2	24
3	24
4	24
5	24
6	24

Experiment 5:

AIM: Apply multiple UDP and TCP applications between any 2 nodes of N (N=4)node Ethernet LAN and compare it with experiment number 4.(compare multiple unicast with multicast)

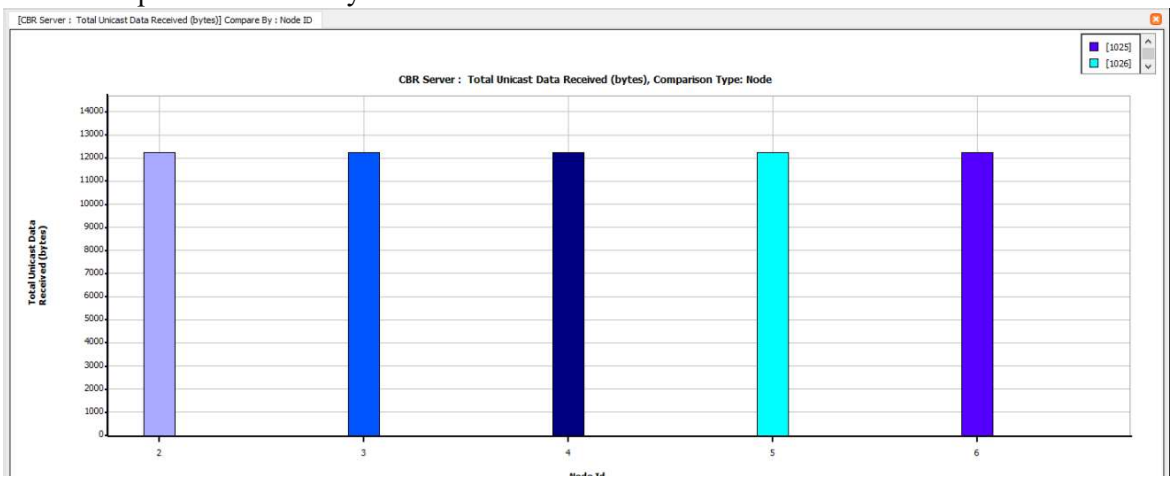
PROCEDURE:

1. Select HUB, select routing protocol under node configuration tab.
2. Set enable multicast field to yes.
3. Group management protocol to IGMP.
4. Multicast routing protocol to DVMRP.
5. Update router list with the node id of the sender.
6. Select multicast group editor from tool.
7. Add the receiving nodes.

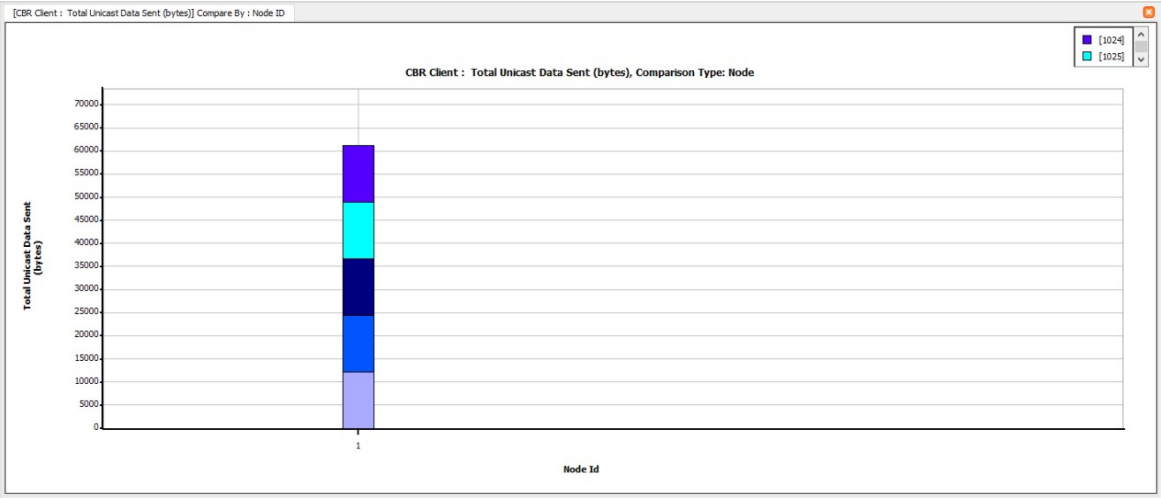


OBSERVATIONS:

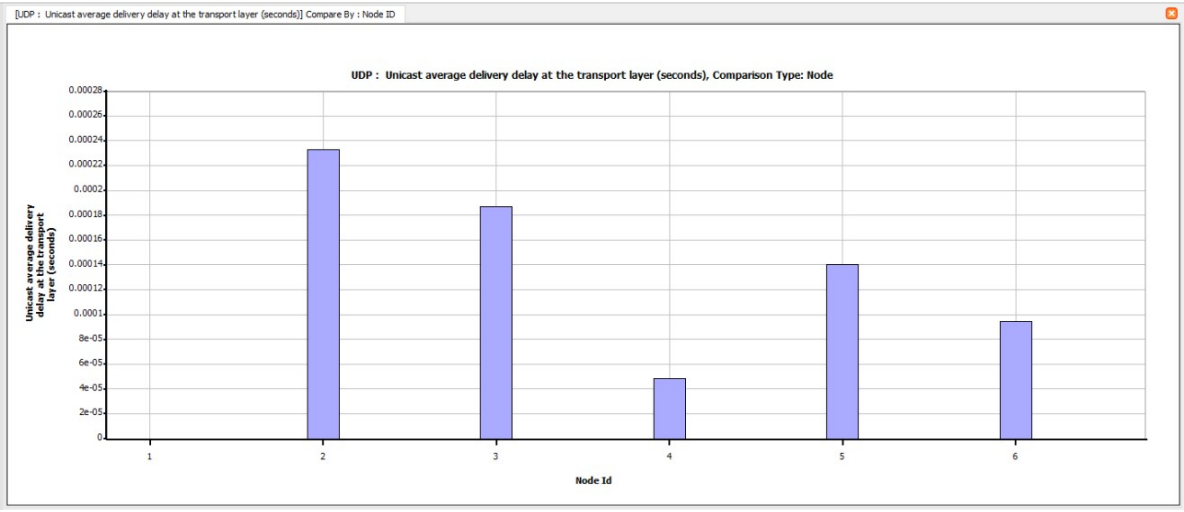
Total multiple unicast data bytes received



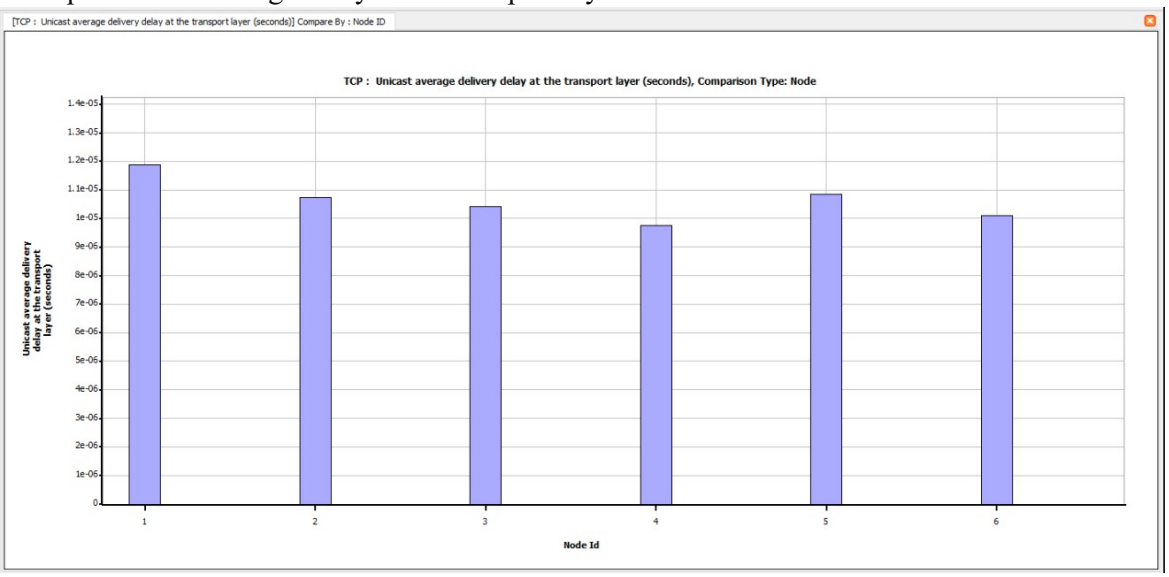
Total multiple unicast data bytes sent



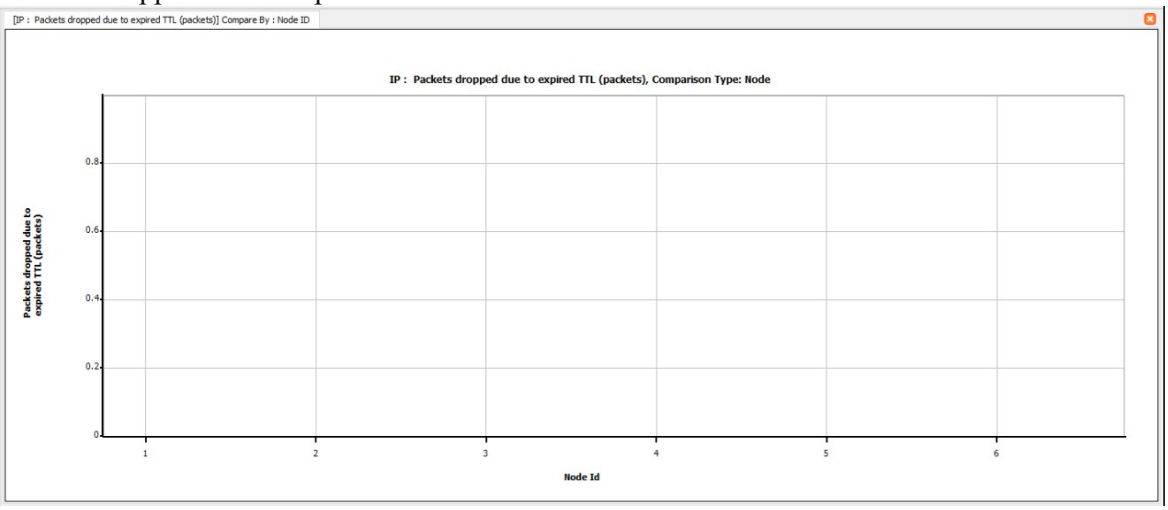
Multiple unicast average delay at the transport layer for UDP



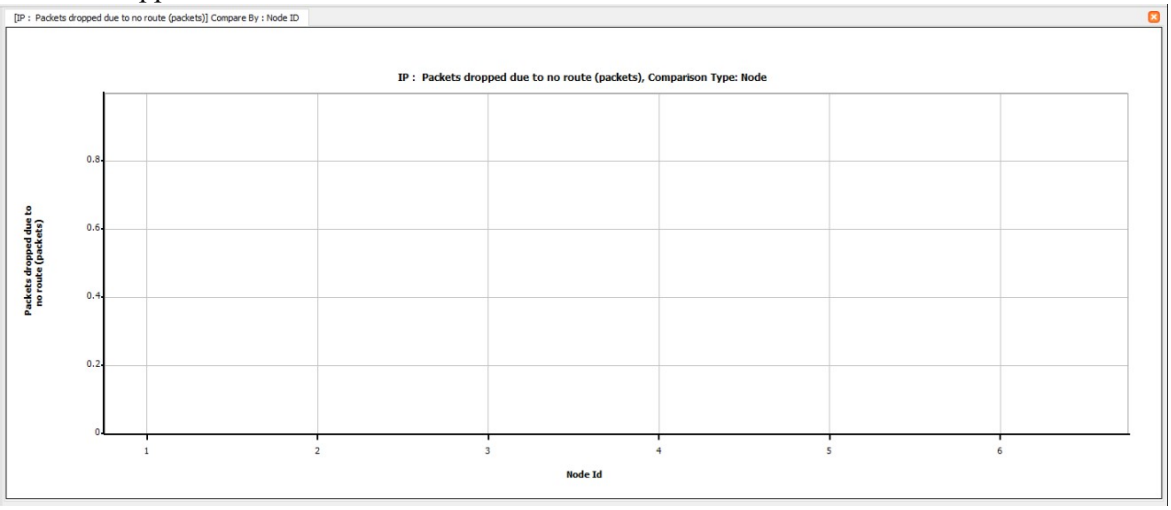
Multiple unicast average delay at the transport layer for TCP



Packets dropped due to expired TTL



Packets dropped due to no route



RESULTS:

The neighbors and packets sent by dvmrp not there as in multicast.
The outputs are

Total multicast data bytes received

Node id	number of packets
1	12100
2	12100
3	12100
4	12100
5	12100
6	12100

Multicast average delay at the transport layer for UDP

Node id	number of packets
1	
2	0.00023
3	0.00019
4	5 e-05
5	0.00014
6	9.5 e-05

Packets dropped due to expired TTL

Node id	number of packets
1	0
2	0
3	0
4	0
5	0
6	0

Packets dropped due to no route

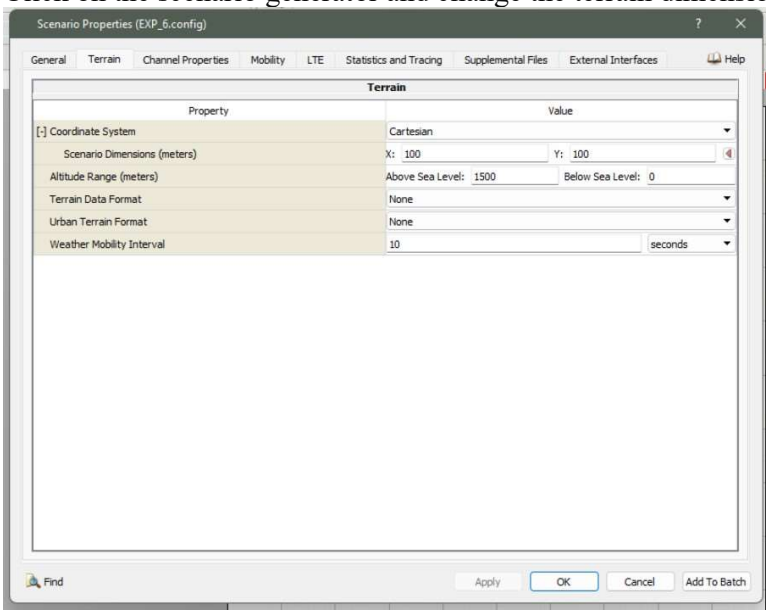
Node id	number of packets
1	0
2	0
3	0
4	0
5	0
6	0

Experiment 6:

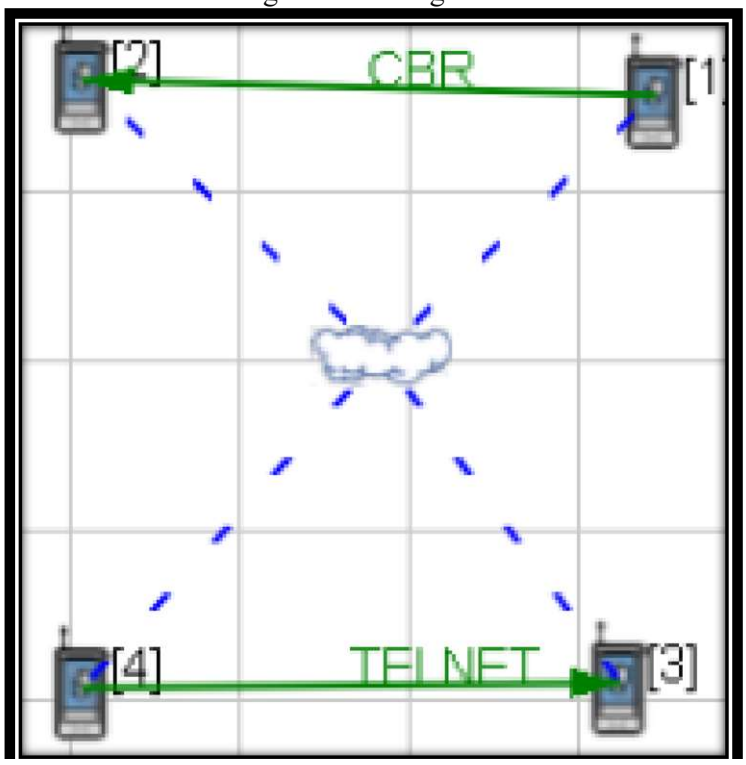
AIM: Simulate a wireless ad hoc network apply relevant TCP and UDP applications between any 2 nodes and determine
i) Number of packets dropped due to retransmission limit
ii) Number of CTS packets sent by the node
iii) Number of RTS packets sent and ACK packets sent by the node
iv) Determine the number of RTS retransmission due to timeout
v) Packet retransmission due to ACK timeout
vi) Signals received with error

PROCEDURE:

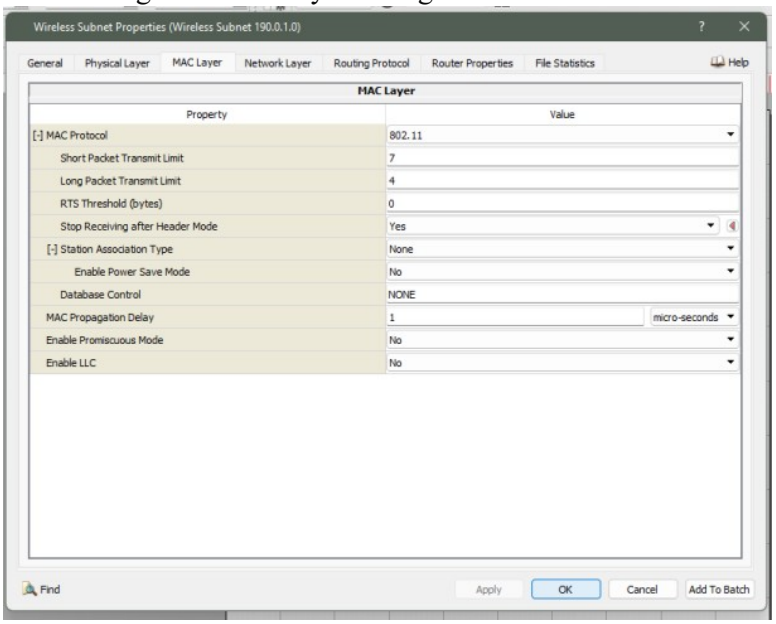
1. Click on the scenario generator and change the terrain dimensions as follows.



2. Make connections as given in the figure.



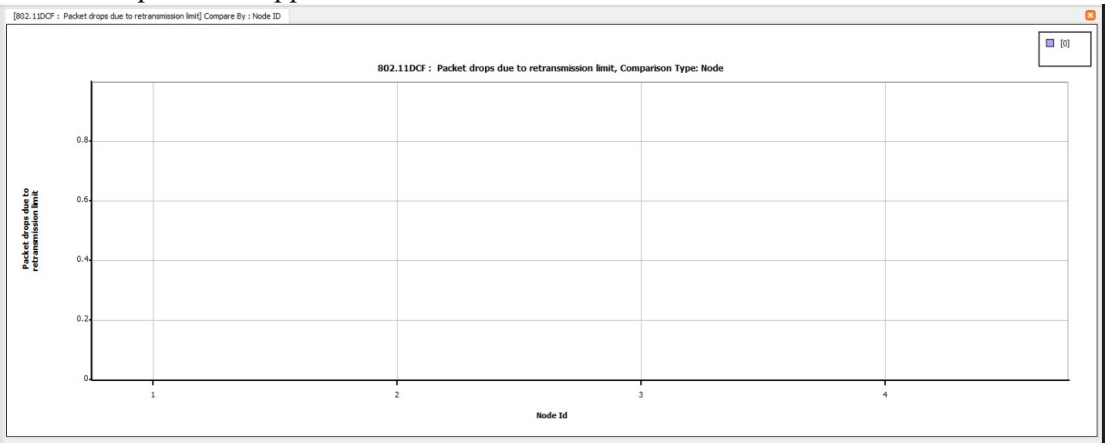
3. Make changes in MAC layer configuration of cloud.



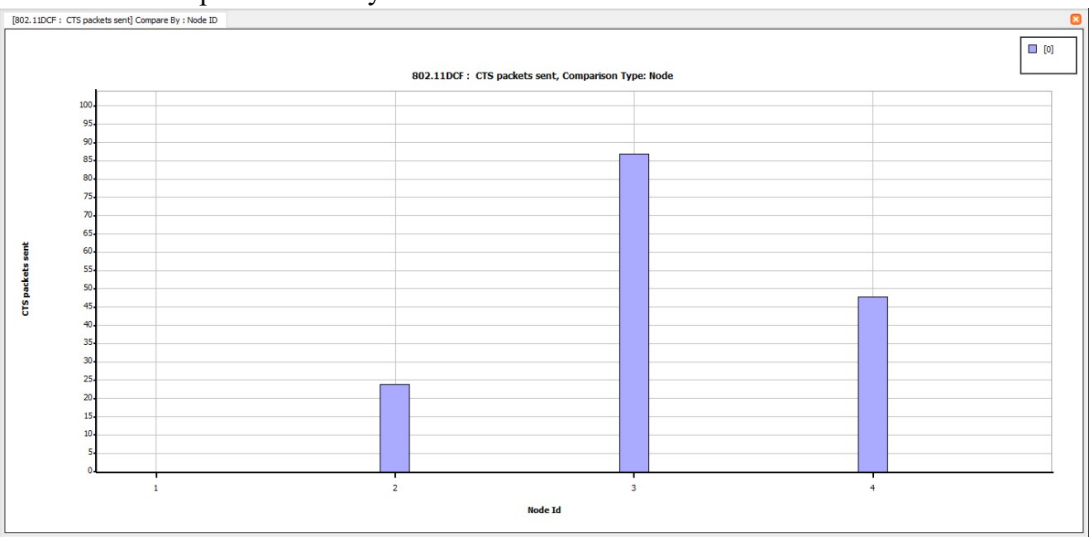
4. Save, simulate and run.

OBSERVATIONS:

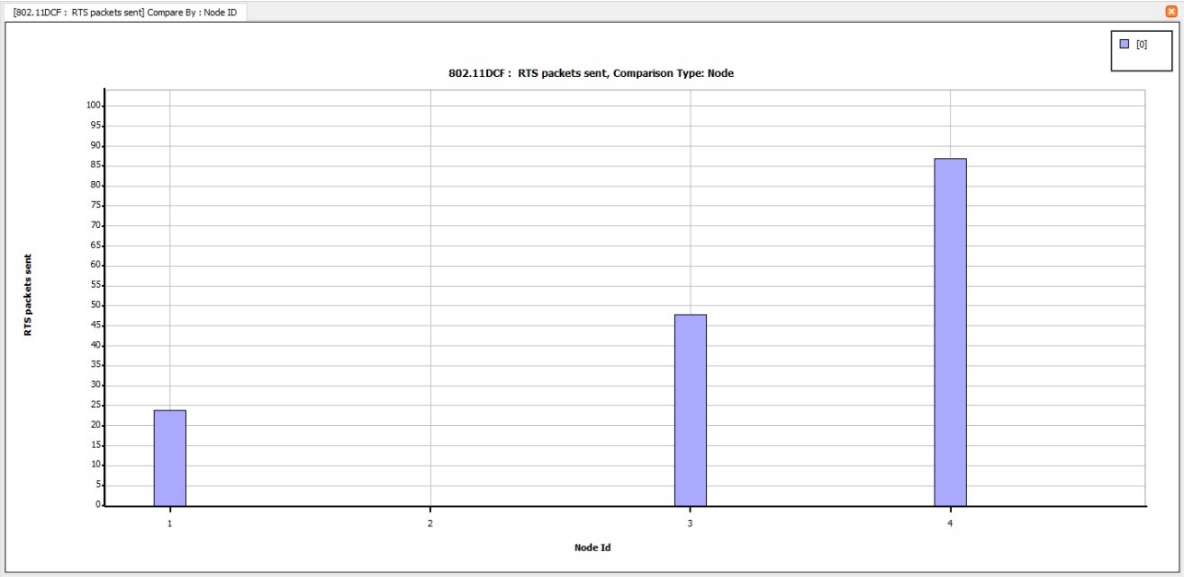
Number of packets dropped due to retransmission limit



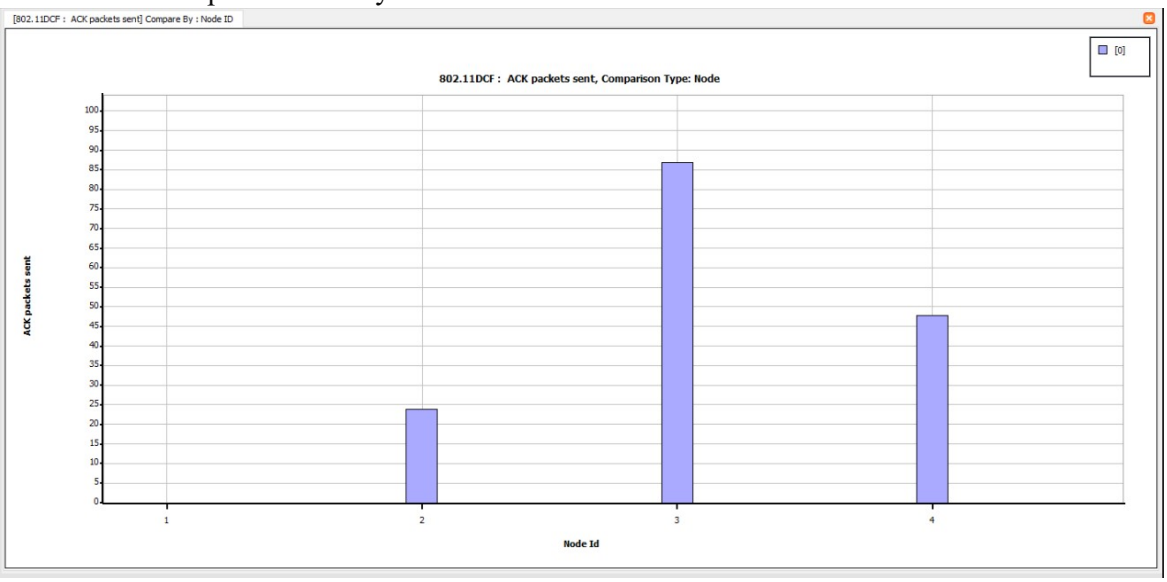
Number of CTS packets sent by the node



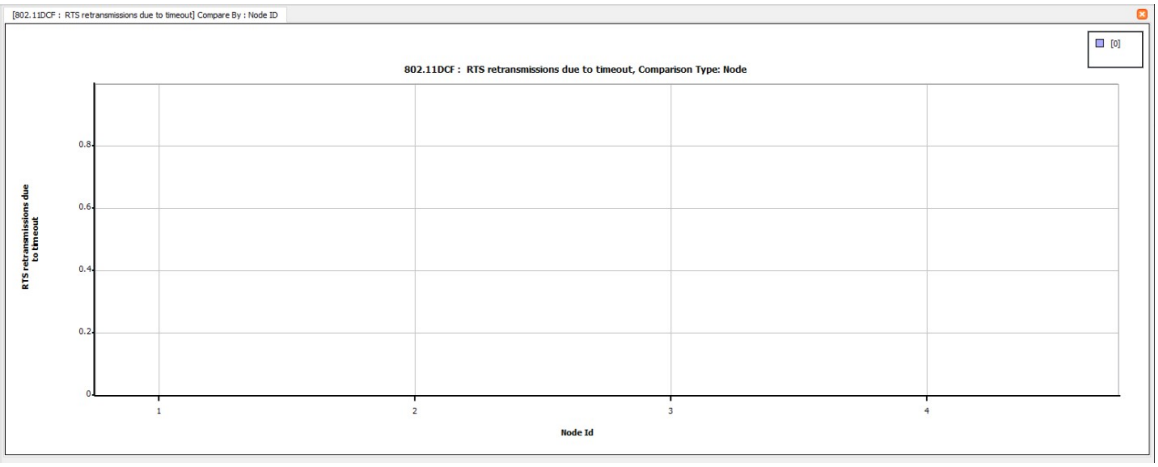
Number of RTS packets sent by the node



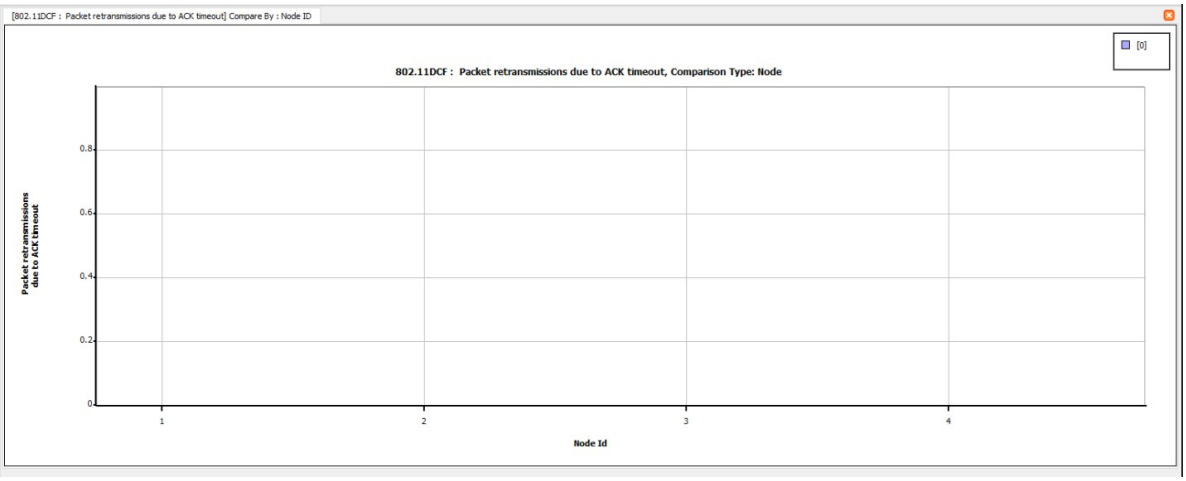
Number of ACK packets sent by the node



The number of RTS retransmission due to timeout



Packet retransmission due to ACK time out



RESULTS:

Number of packets dropped due to retransmission limit

Node id	number of packets
1	0
2	0
3	0
4	0

Number of CTS packets sent by the node

Node id	number of packets
1	0
2	24
3	87
4	47

Number of RTS packets sent by the node

Node id	number of packets
1	24
2	0
3	47
4	87

Number of ACK packets sent by the node

Node id	number of packets
1	0
2	24
3	87
4	47

The number of RTS retransmission due to timeout

Node id	number of packets
1	0
2	0
3	0
4	0

Packet retransmission due to ACK time out

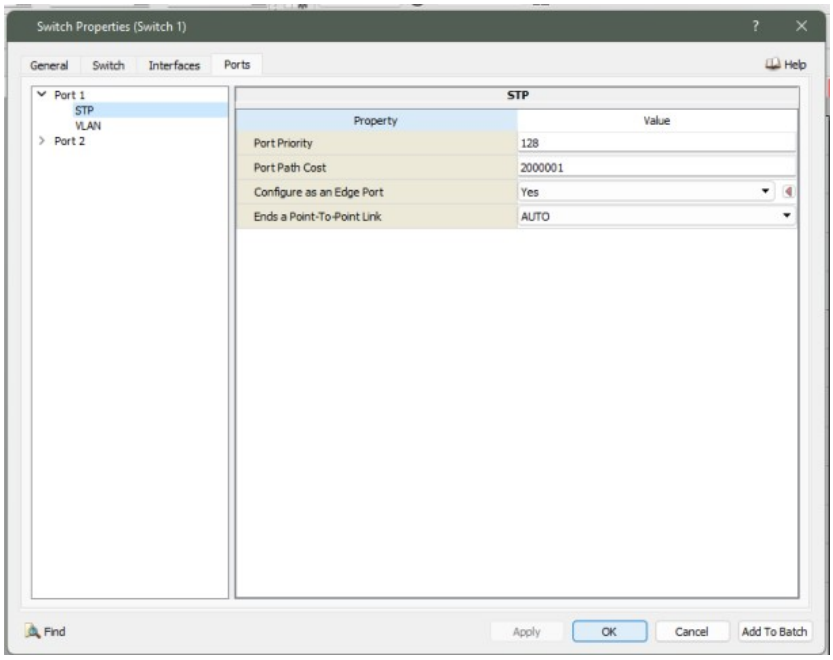
Node id	number of packets
1	0
2	0
3	0
4	0

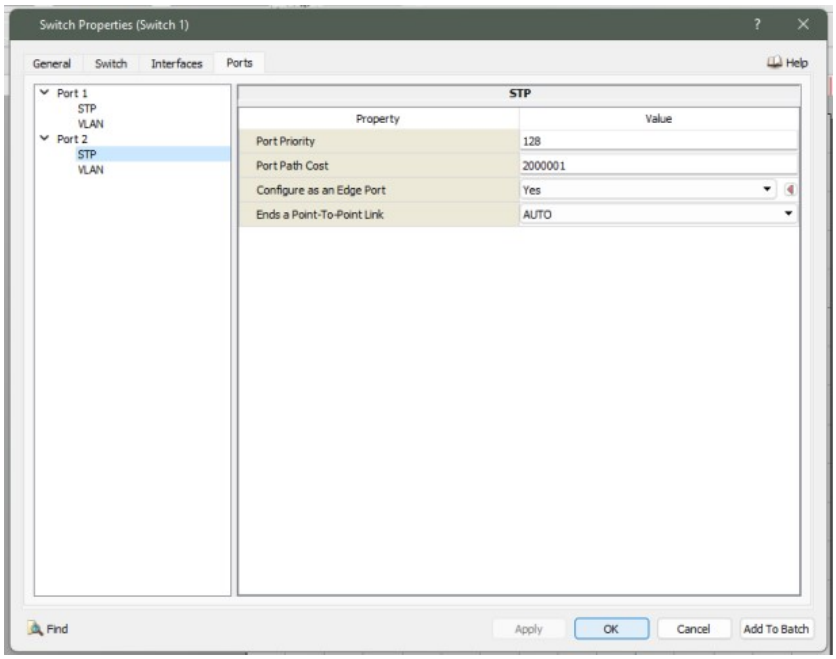
Experiment 7:

AIM: Simulate a network having 2 LANs connected by a switch. Apply relevant TCP and UDP applications between nodes across the LANS (send data from a node in one LAN to a node in another LAN) and determine application layer, transport layer, network layer and MAC layer parameters.

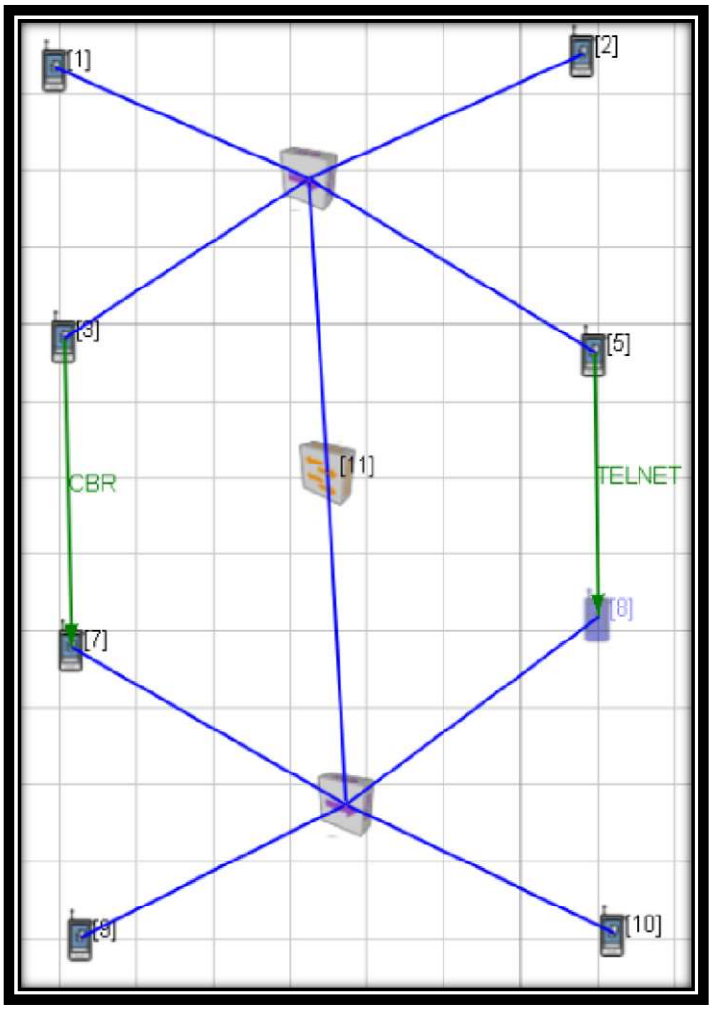
PROCEDURE:

1. Change the port property of the switch as shown below.





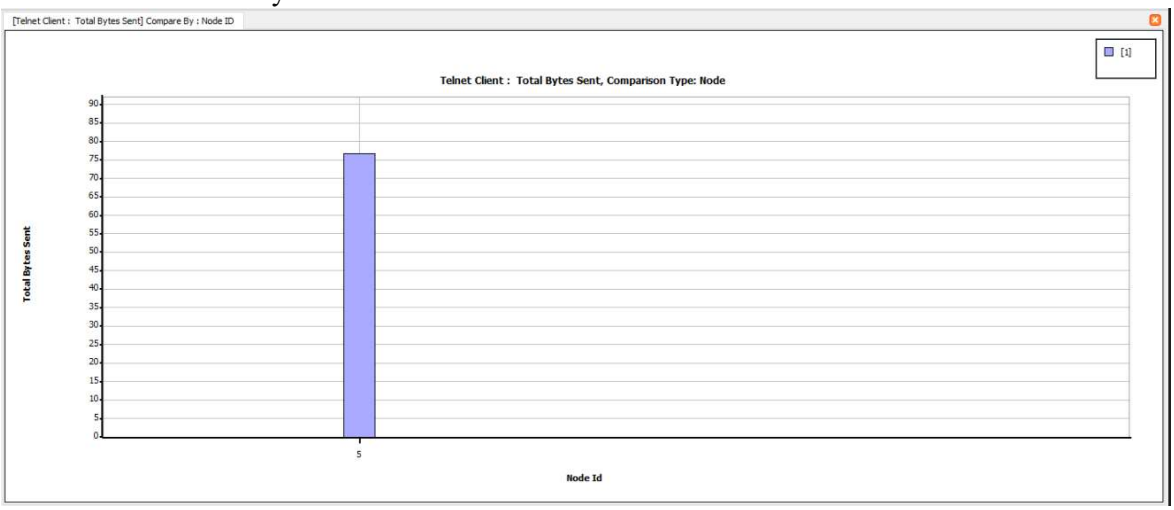
2. Make connections as shown in the figure



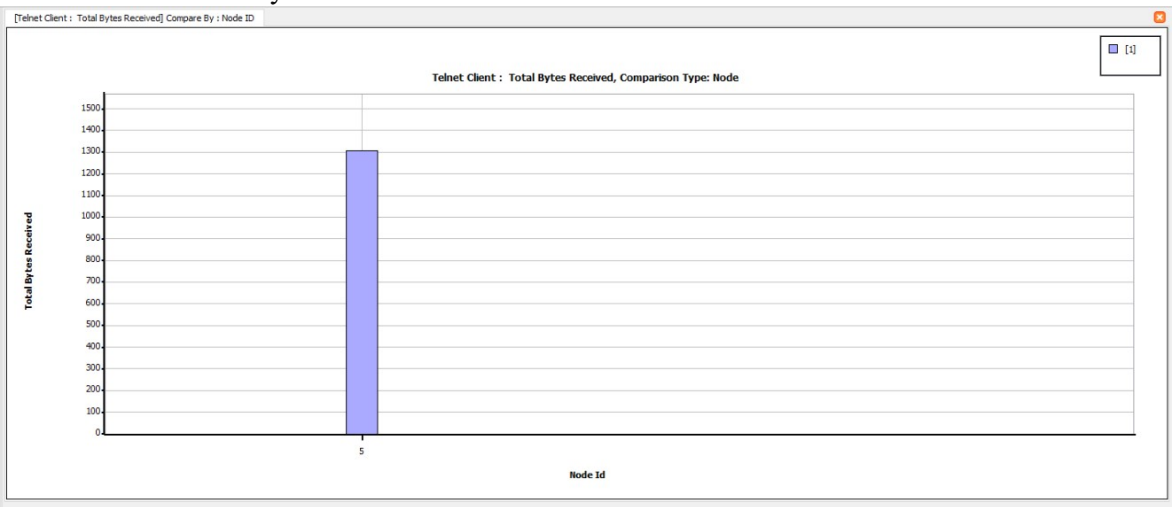
OBSERVATIONS:

APPLICATION LAYER

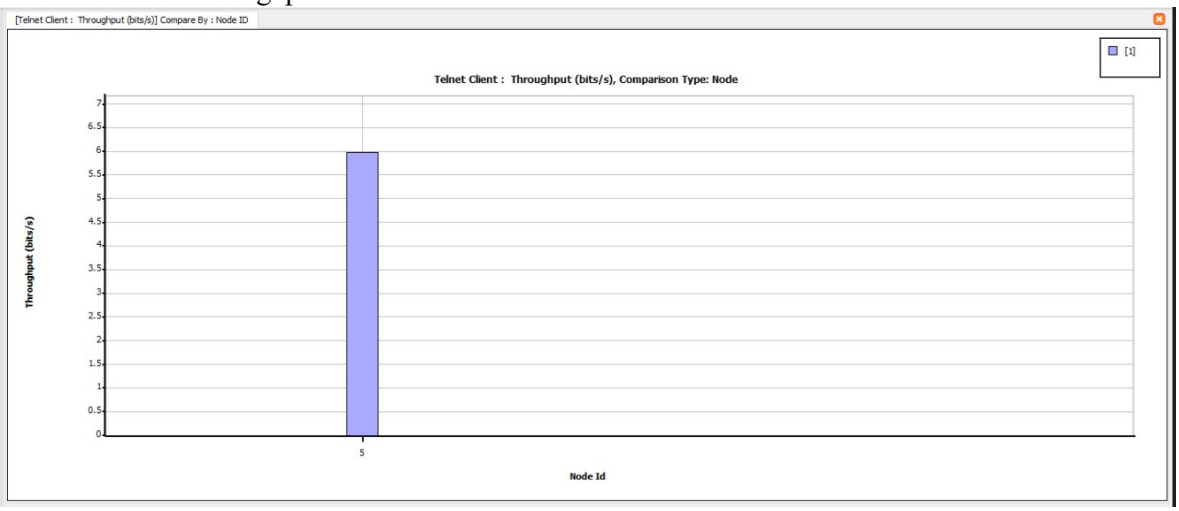
TELNET client total bytes sent



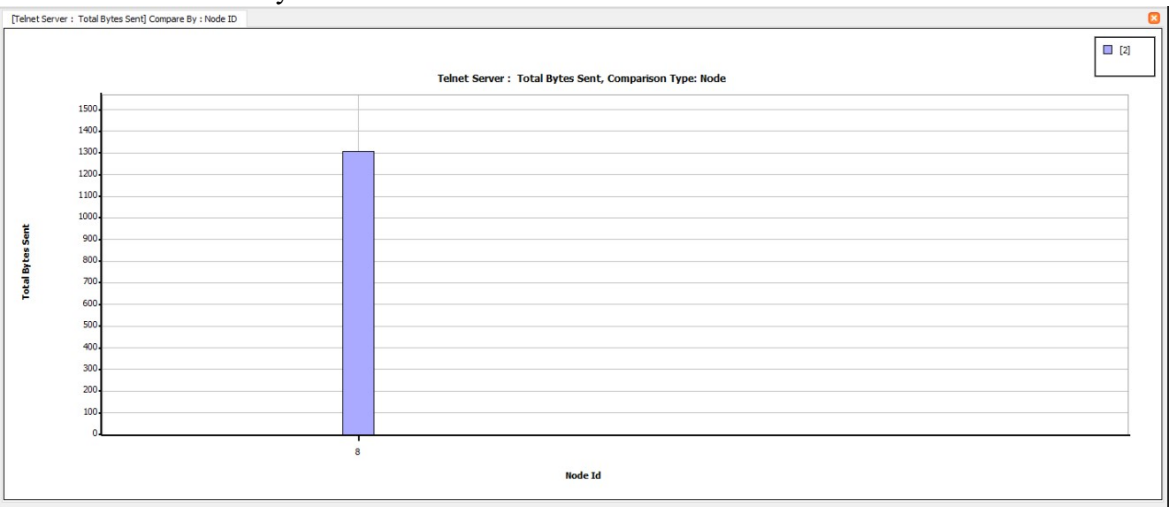
TELNET client total bytes received



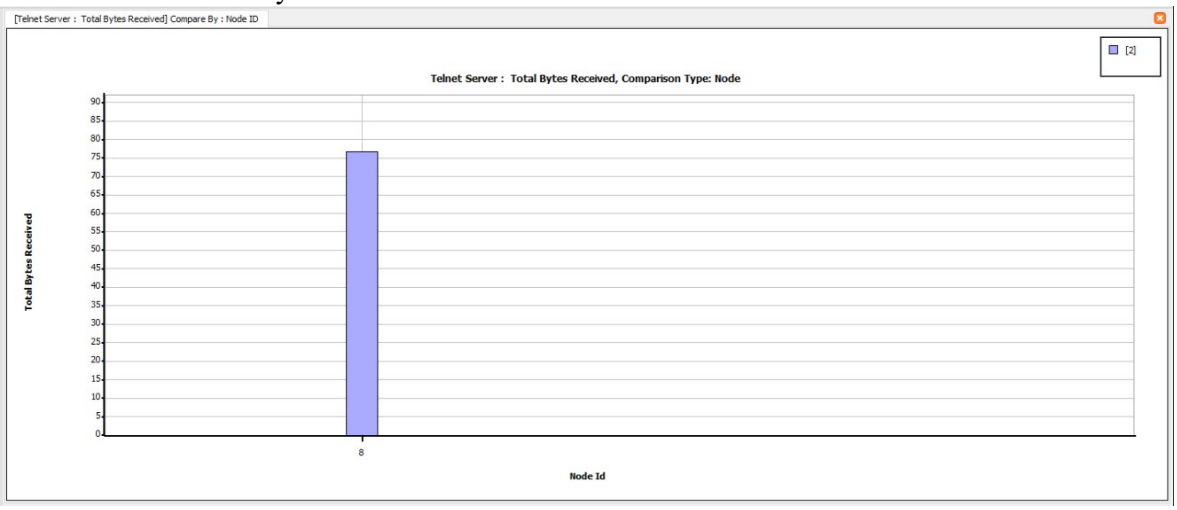
TELNET client throughput



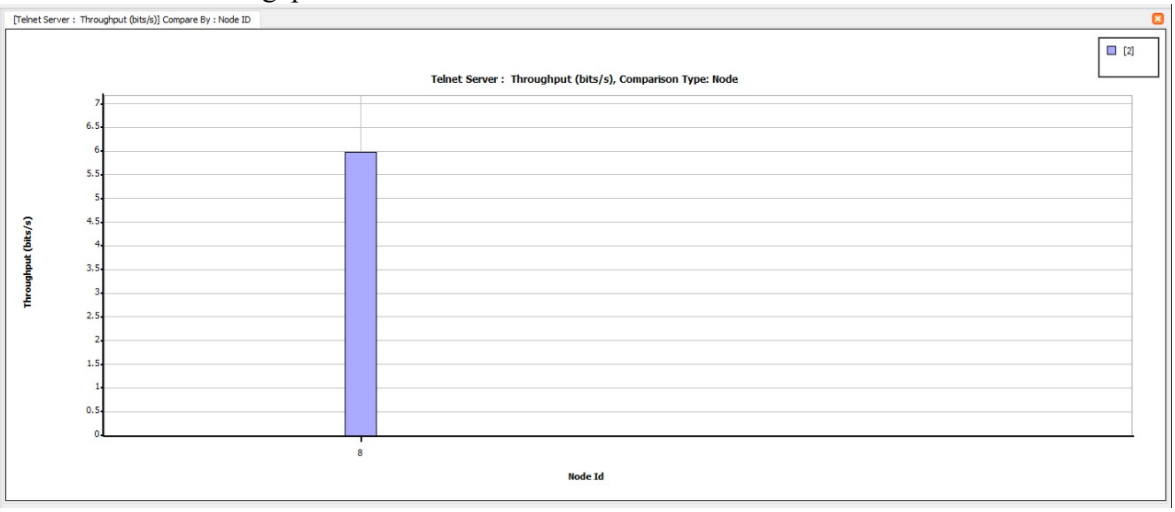
TELNET server total bytes sent



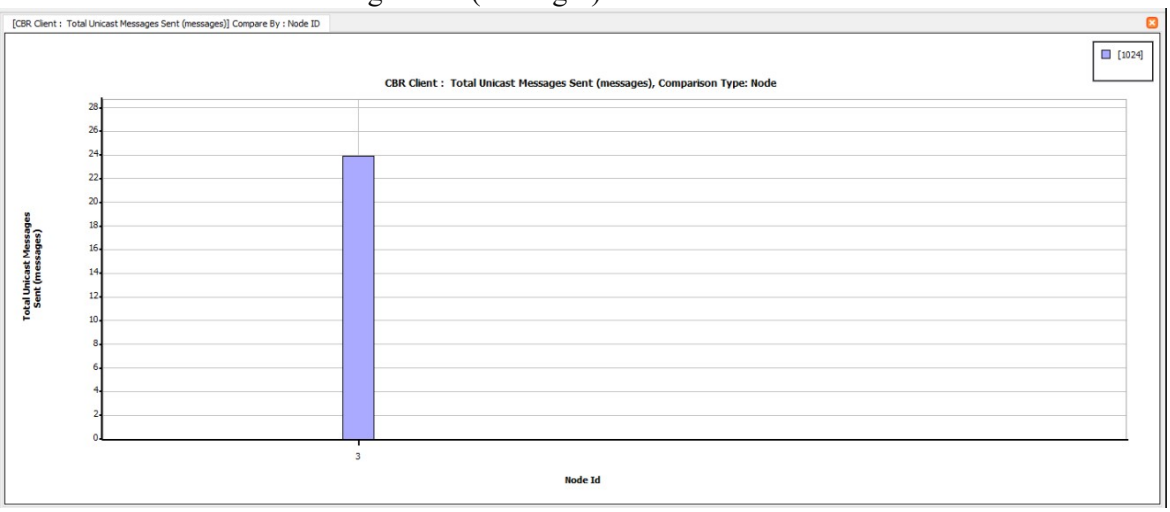
TELNET server total bytes received



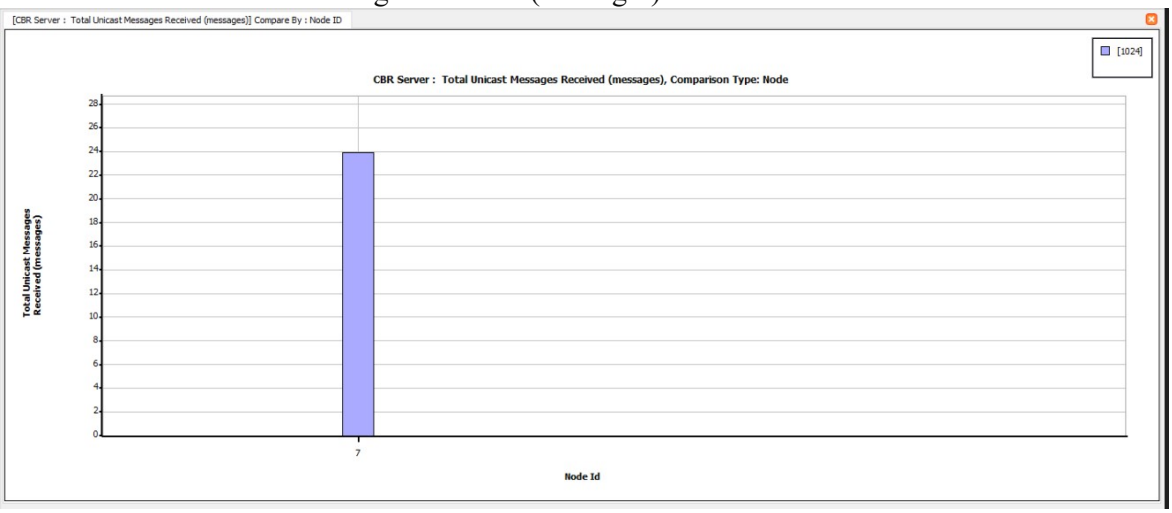
TELNET server throughput



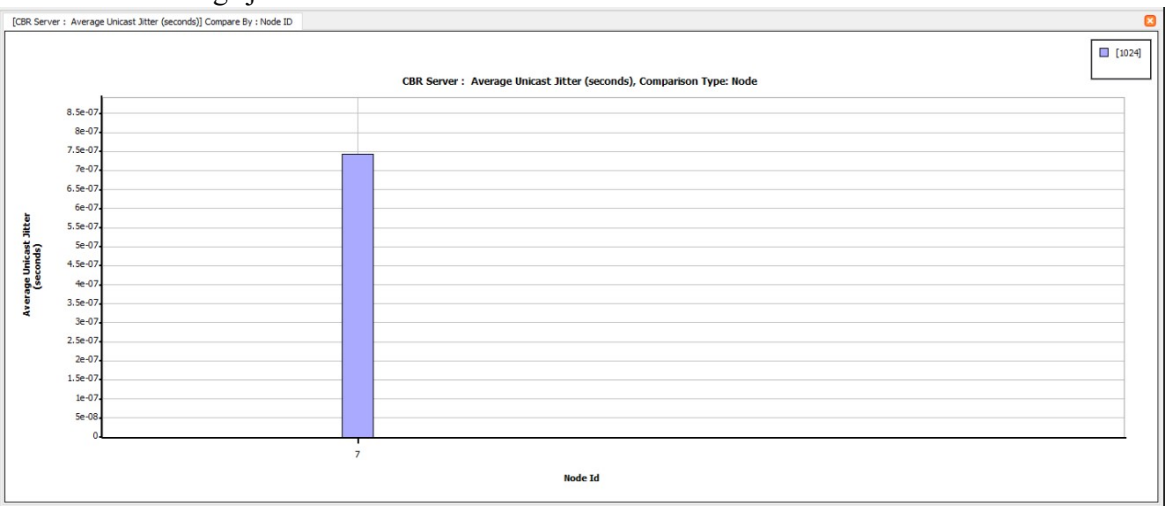
CBR client total unicast messages sent (messages)



CBR server total unicast messages received (messages)

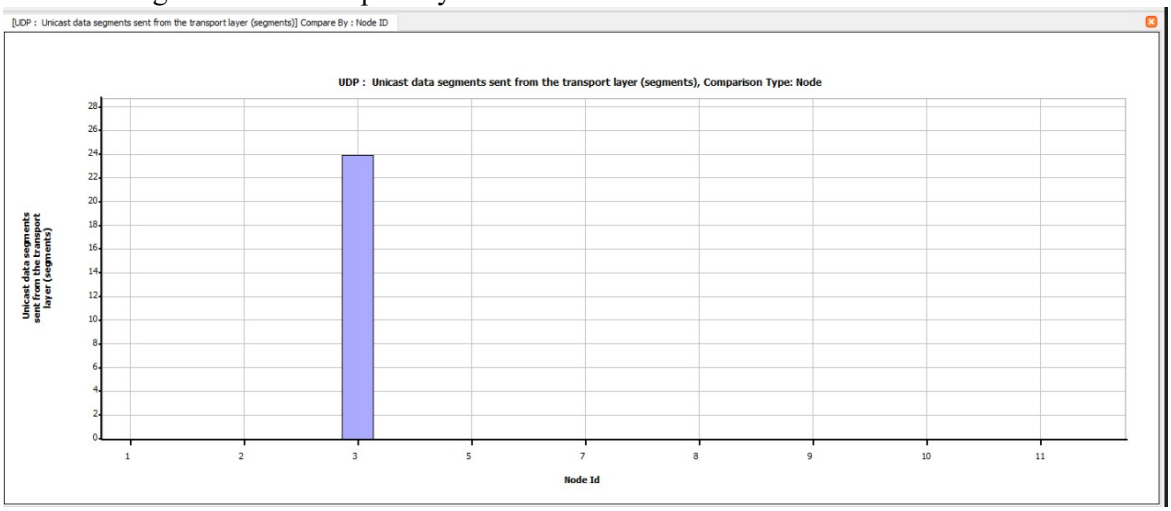


CBR server average jitter

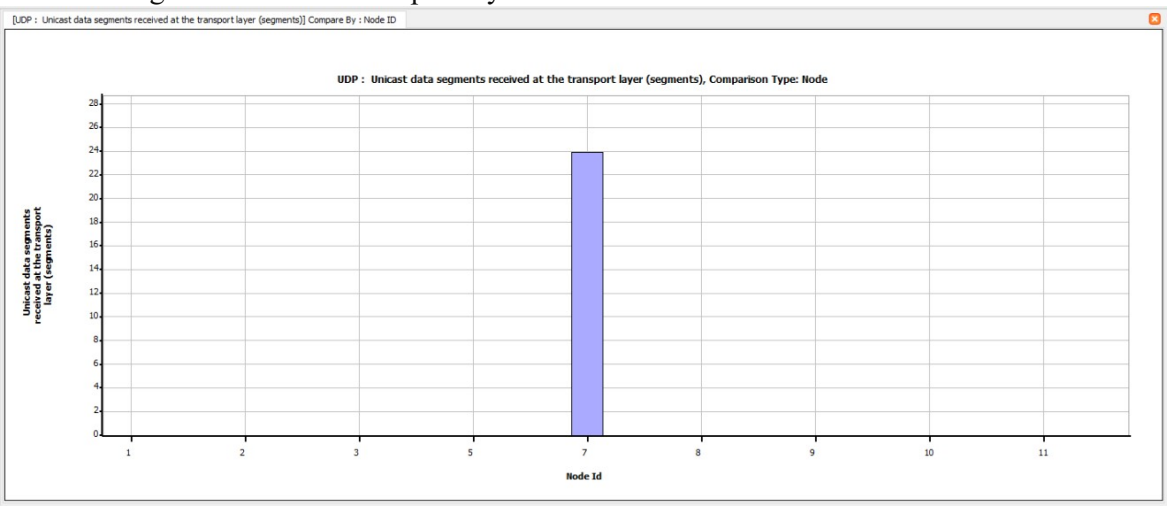


TRANSPORT LAYER

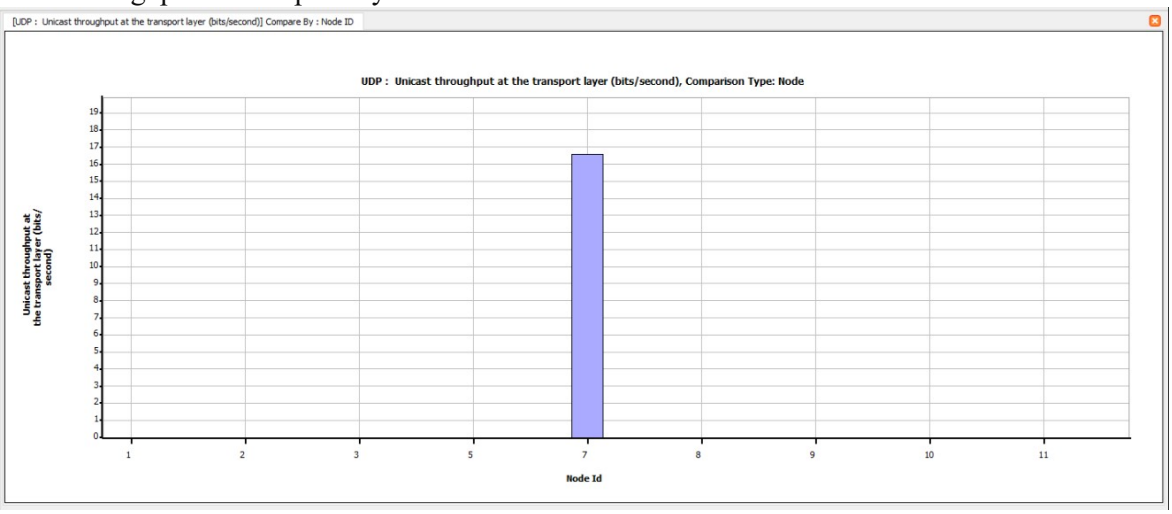
UDP data segments sent transport layer



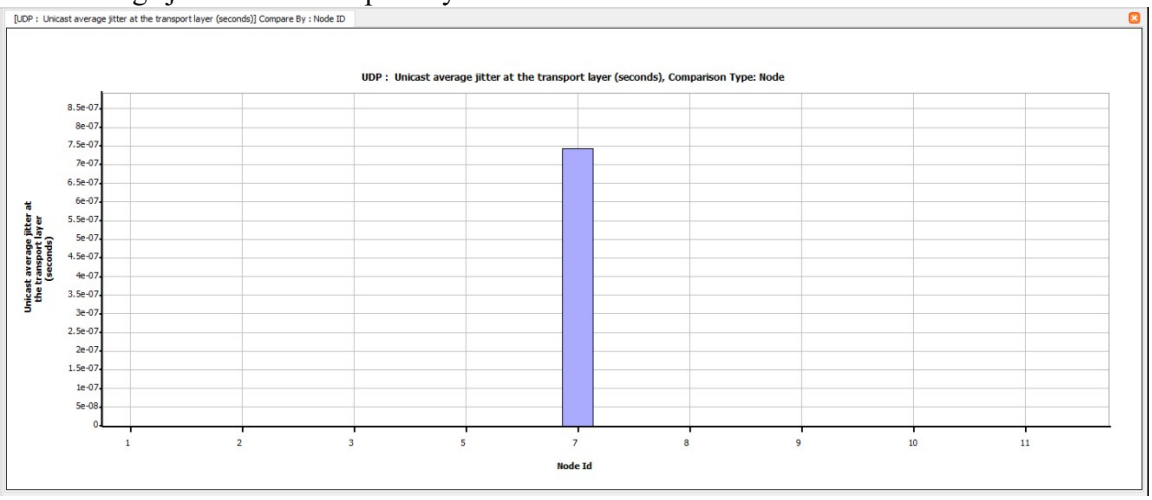
UDP data segments received transport layer



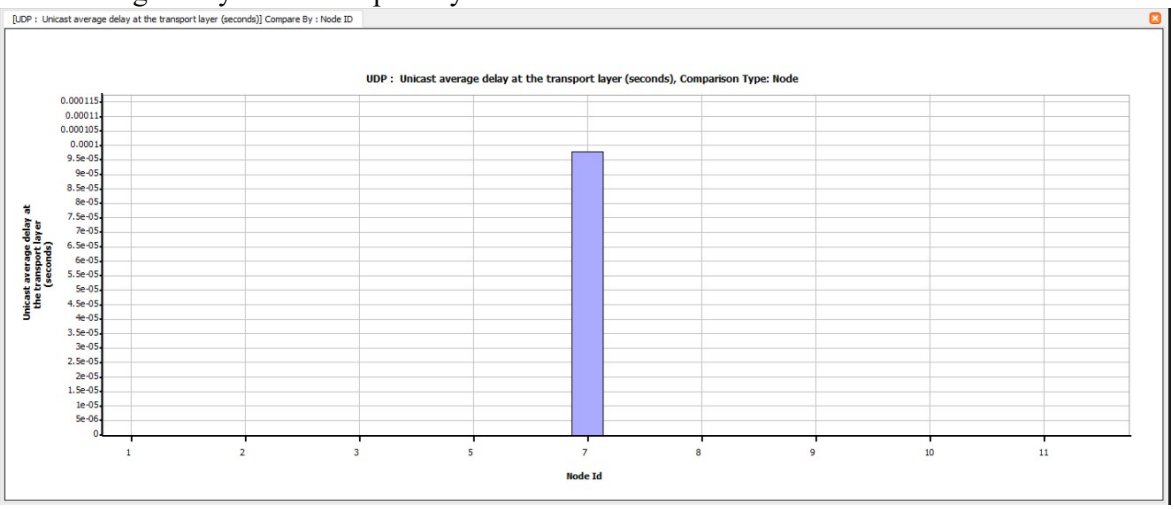
UDP throughput in transport layer



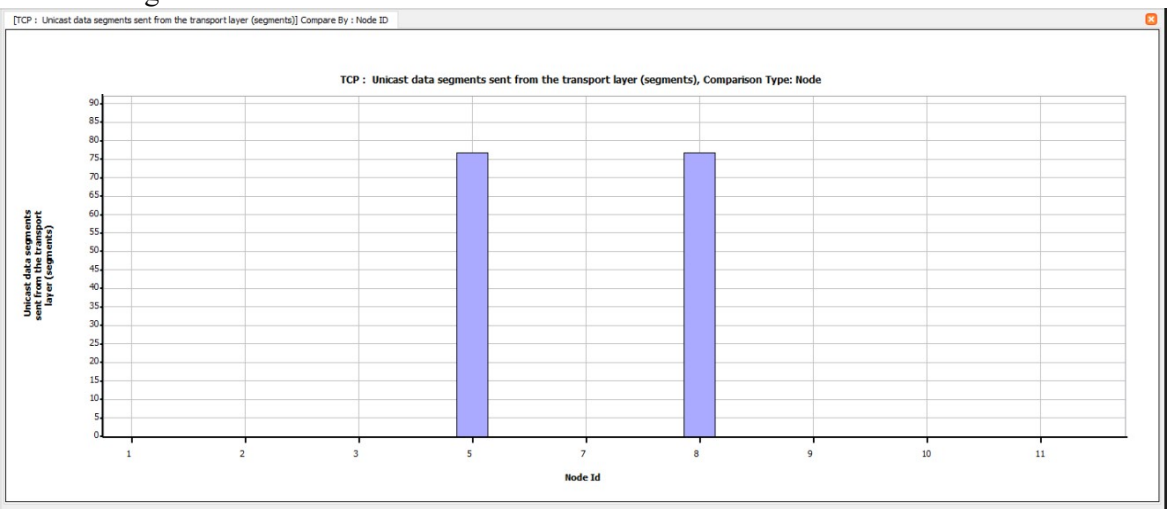
UDP average jitter in the transport layer



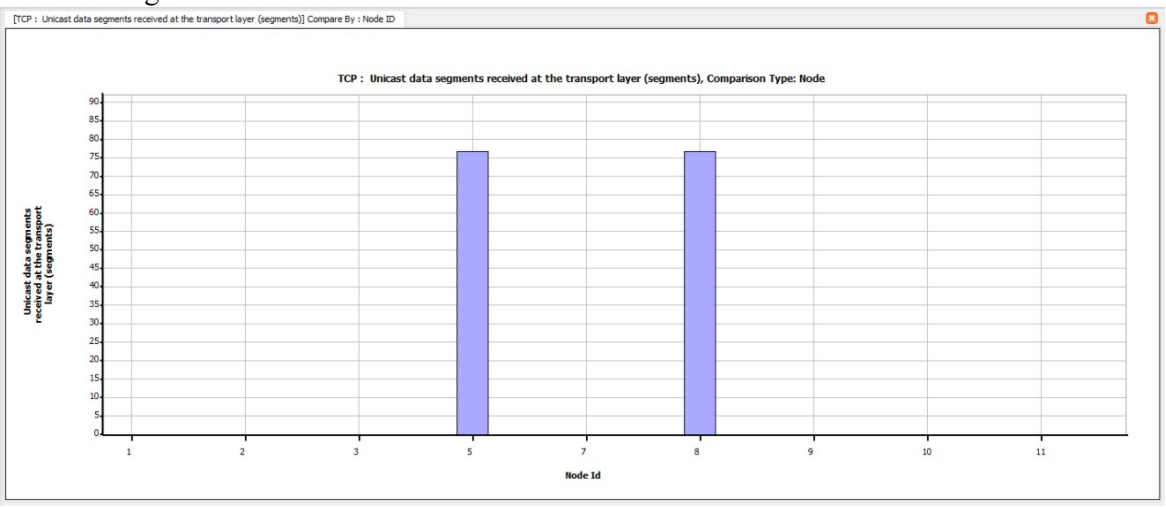
UDP average delay in the transport layer



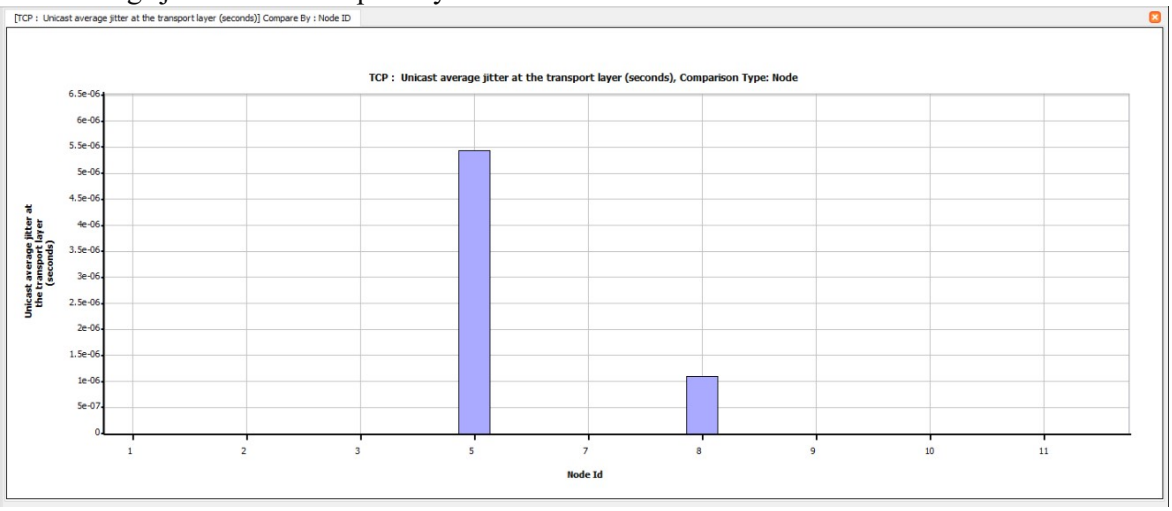
TCP data segments sent



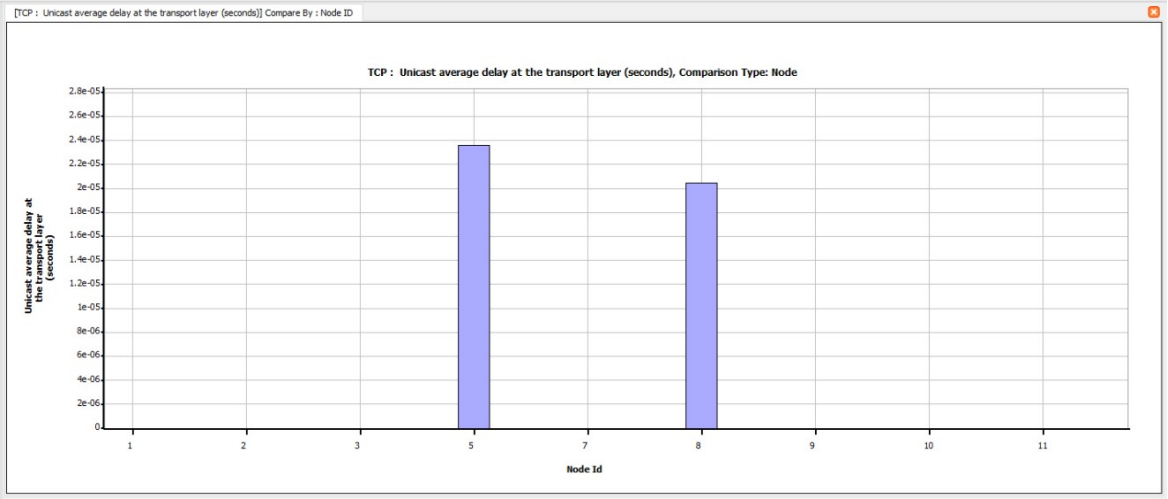
TCP data segments received



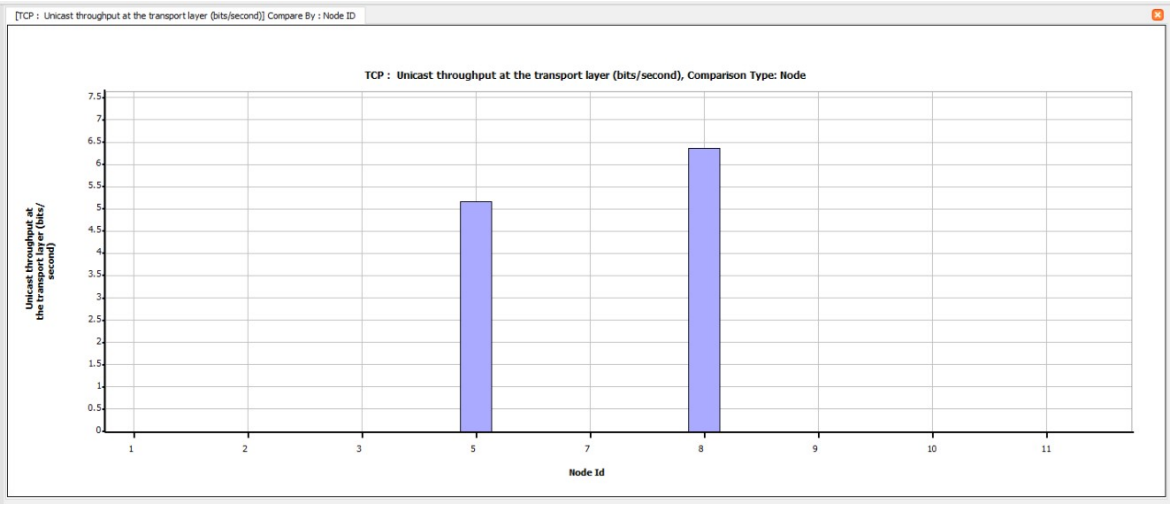
TCP average jitter in the transport layer



TCP average delay in the transport layer

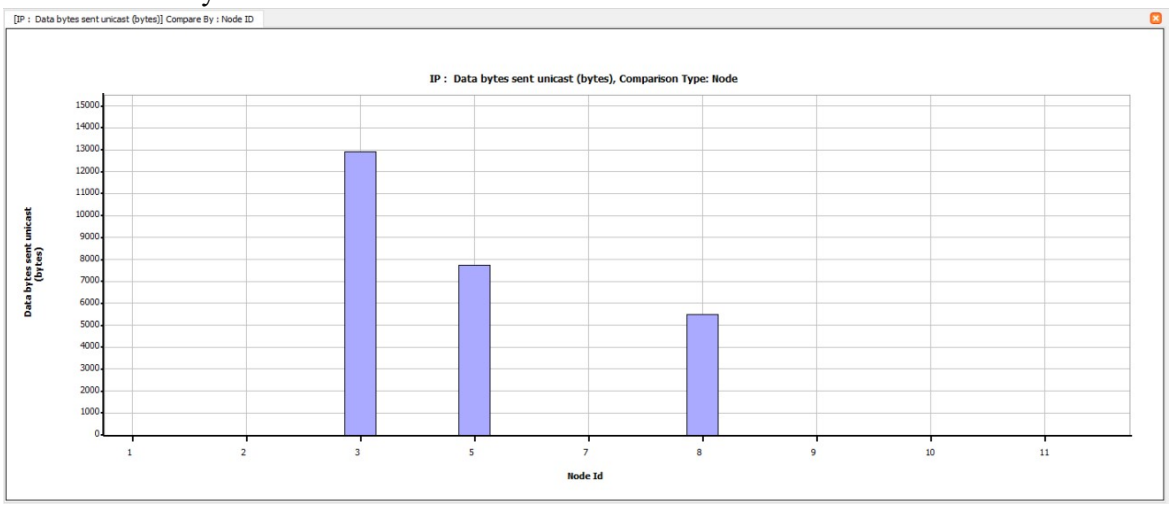


TCP throughput at the transport layer

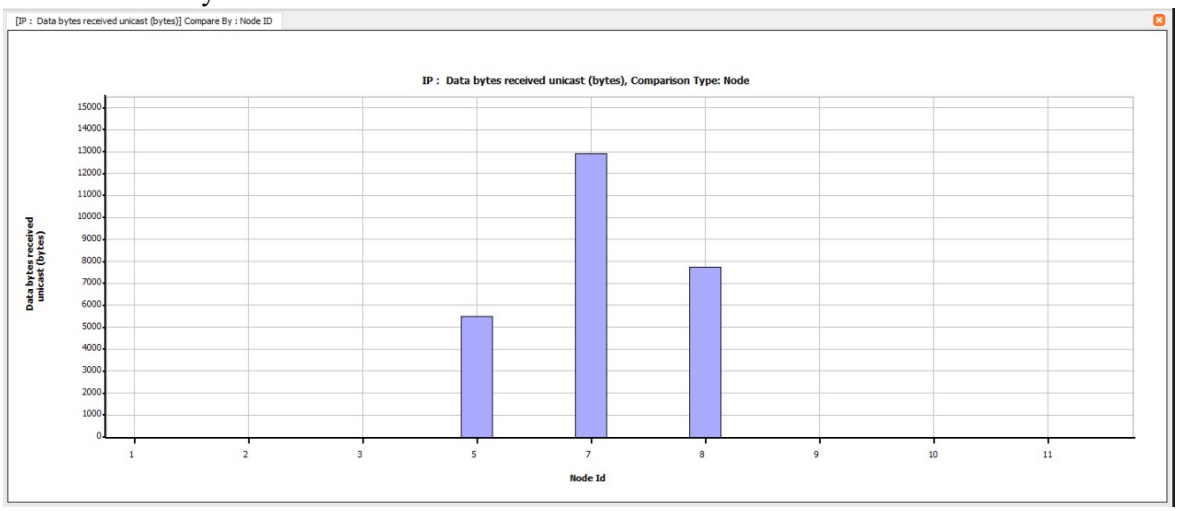


NETWORK LAYER

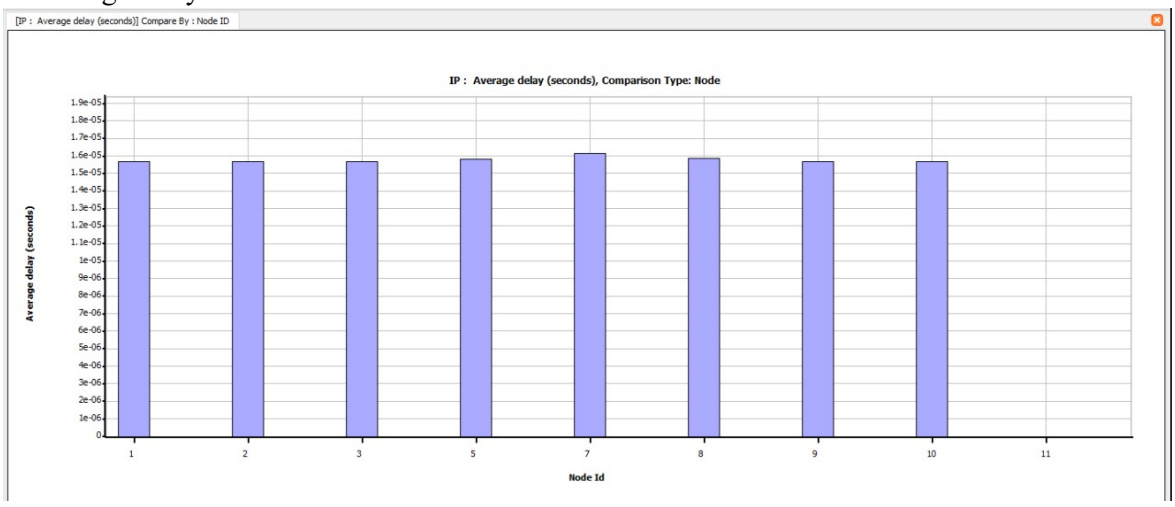
IP unicast data bytes sent



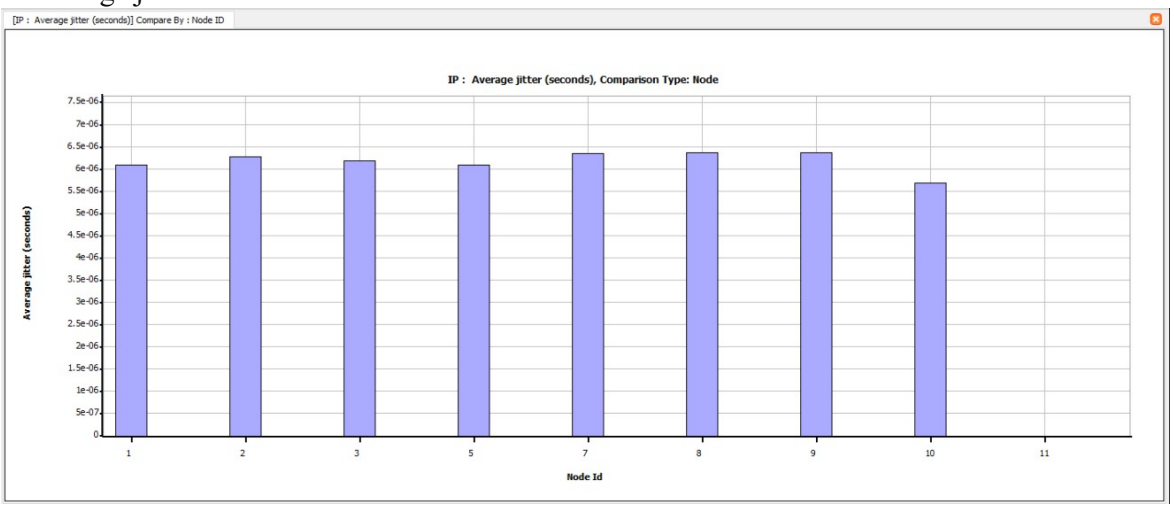
IP unicast data bytes received



IP average delay

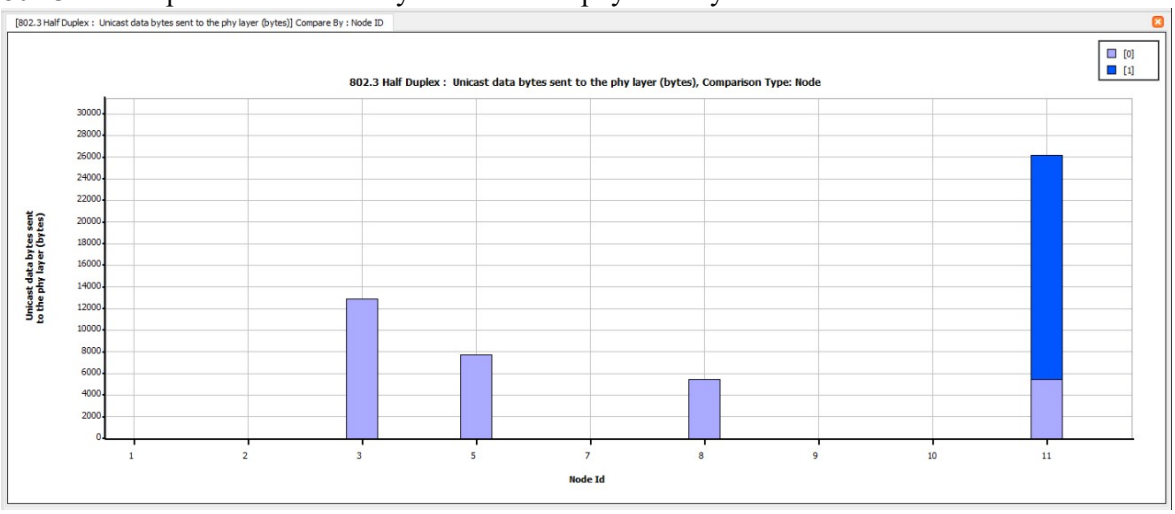


IP average jitter

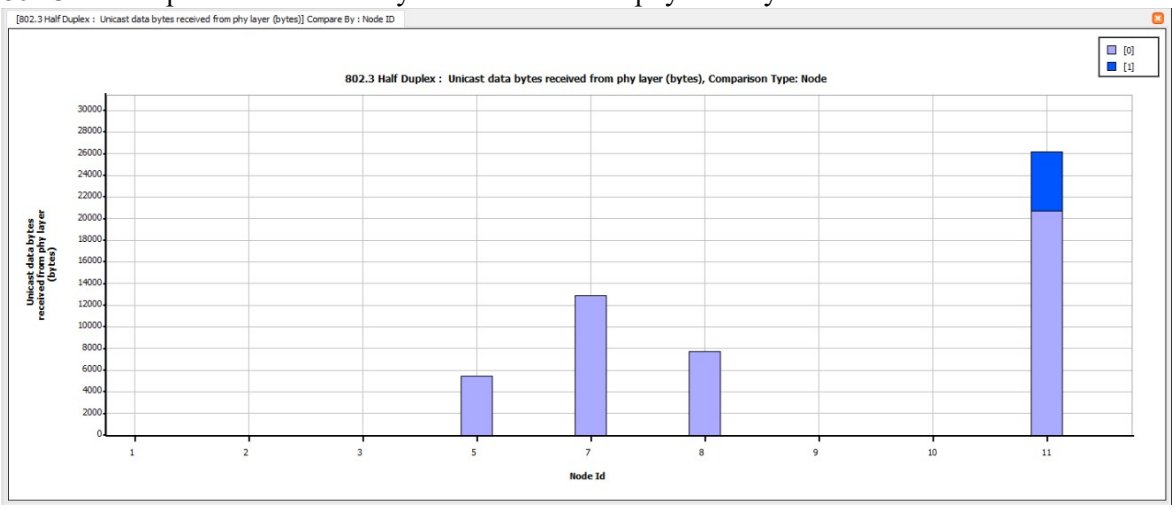


MAC LAYER

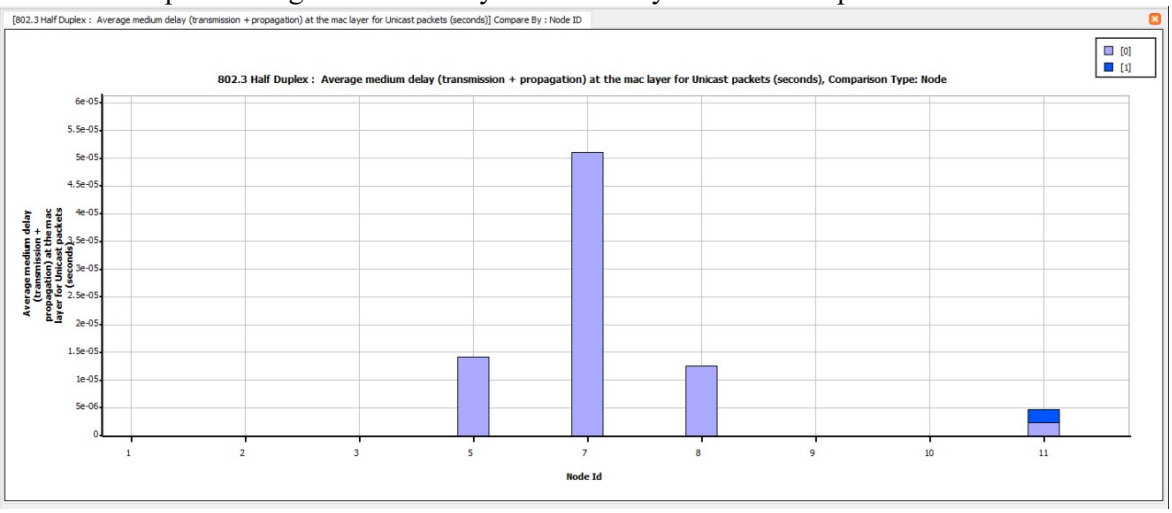
802.3 Half duplex Unicast data bytes sent to the physical layer



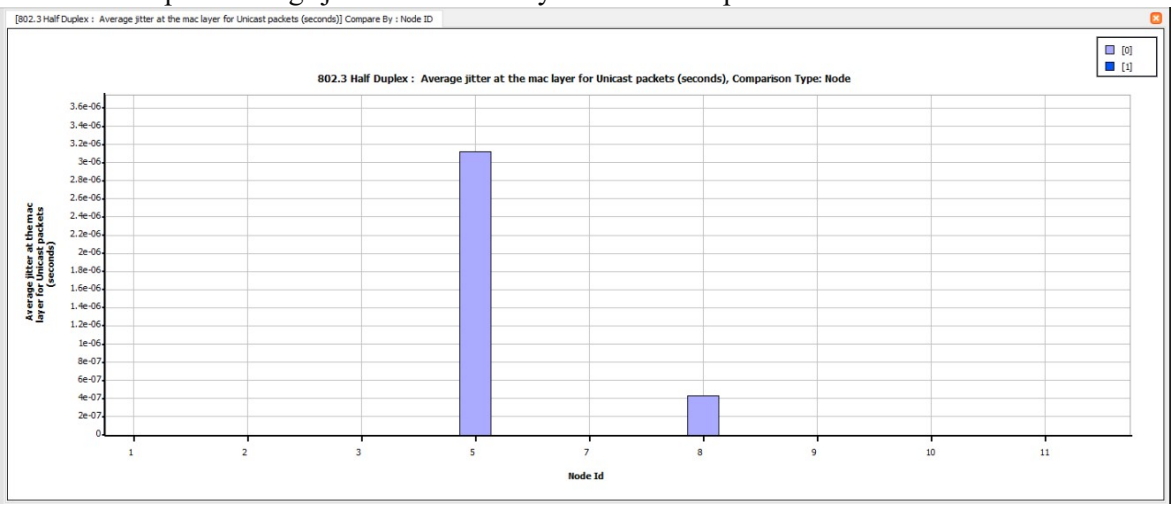
802.3 Half duplex Unicast data bytes received to the physical layer



802.3 Half duplex average medium delay at the mac layer for Unicast packets



802.3 Half duplex average jitter at the mac layer for Unicast packets



RESULTS:

TELNET client total bytes sent

Node id	number of bytes
5	1300

TELNET client total bytes received

Node id	number of bytes
5	6

TELNET client throughput

Node id	bits/s
5	6

TELNET server total bytes sent

Node id	number of bytes
8	1300

TELNET server total bytes received

Node id	number of bytes
8	77

TELNET server throughput

Node id	number of packets
8	6

CBR client total unicast messages sent (messages)

Node id	number of messages
3	24

CBR server total unicast messages received (messages)

Node id	number of messages
7	24

CBR server average jitter

Node id	number of packets
7	7.4 e-07

UDP data segments sent transport layer

Node id	number of segments
3	24

UDP data segments received transport layer

Node id	number of segments
7	24

UDP throughput in transport layer

Node id	bits/s
7	16.5

UDP average jitter in the transport layer

Node id	number of packets
7	7.4 e-07

UDP average delay in the transport layer

Node id	number of packets
7	9.75 e-05

TCP data segments sent

Node id	number of segments
5	77
8	77

TCP data segments received

Node id	number of segments
5	77
8	77

TCP average jitter in the transport layer

Node id	seconds
5	5.4 e-06
8	1.1 e-06

TCP average delay in the transport layer

Node id	number of packets
5	2.4 e-05
8	2.1 e-05

TCP throughput at the transport layer

Node id	number of packets
5	5.1
8	6.4

IP unicast data bytes sent

Node id	number of bytes
3	13000
5	7800
8	5500

IP unicast data bytes received

Node id	number of bytes
5	5500
7	13000
8	7800

IP average delay: 1.6 e-05

IP average jitter: 6.3 e-06

802.3 Half duplex Unicast data bytes sent to the physical layer

Node id	number of packets
3	13000
5	7000
8	5000
11	26000

802.3 Half duplex Unicast data bytes received to the physical layer

Node id	number of packets
5	5000
7	13000
8	7000
11	26000

802.3 Half duplex average medium delay at the mac layer for Unicast packets

Node id	number of packets
5	1.5 e-05
7	5 e-05
8	1.25 e-05
11	5 e-06

Half duplex average jitter at the mac layer for Unicast packets

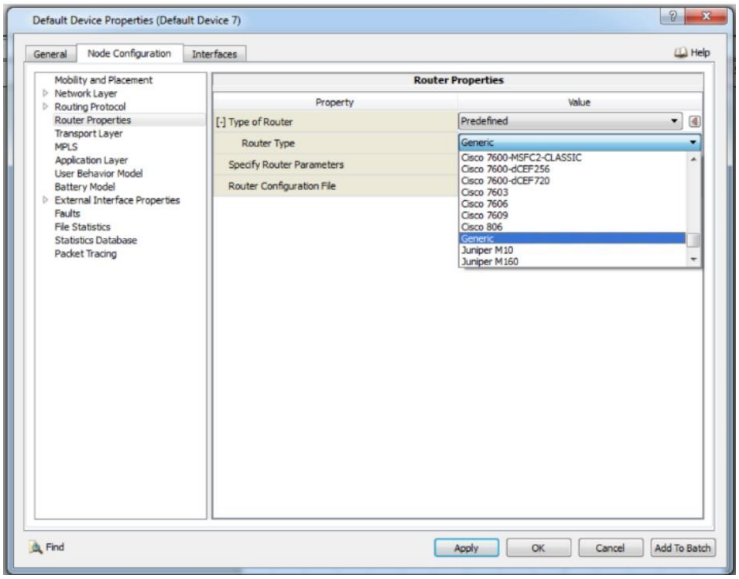
Node id	number of packets
5	3.1 e-06
8	4 e-07

Experiment 8:

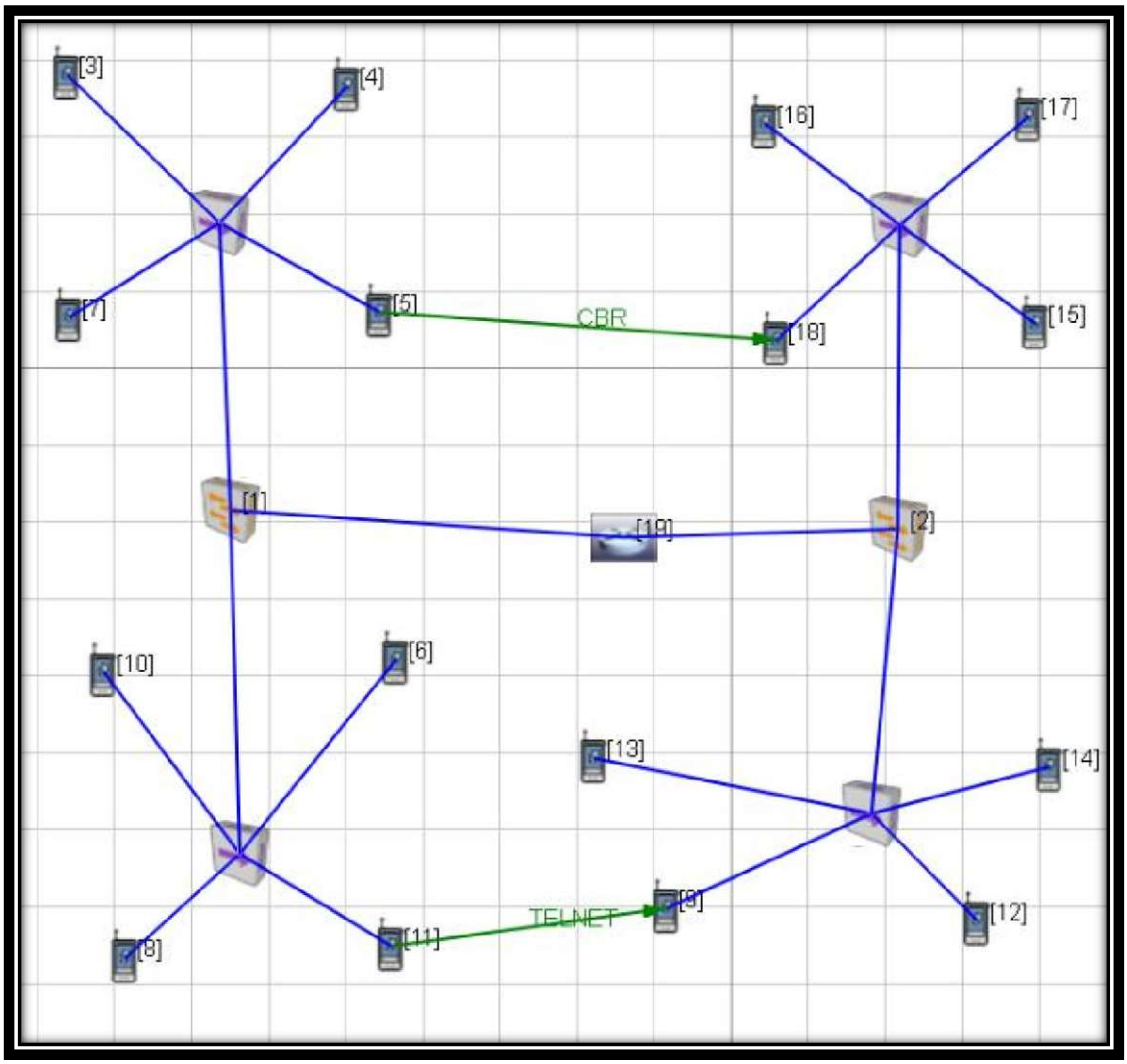
AIM: Simulate a network with the topology as shown in the figure, apply TCP and UDP applications between nodes shown in the figure. Modify the network to make communication happen between node 1 and 9 and node 6 and 16.

PROCEDURE:

1. To set a node as router :
   a. Choose the node and go to it properties.
   b. Then go to node configuration, choose router properties
   c. Set type of router field to predefined and choose router type = generic.

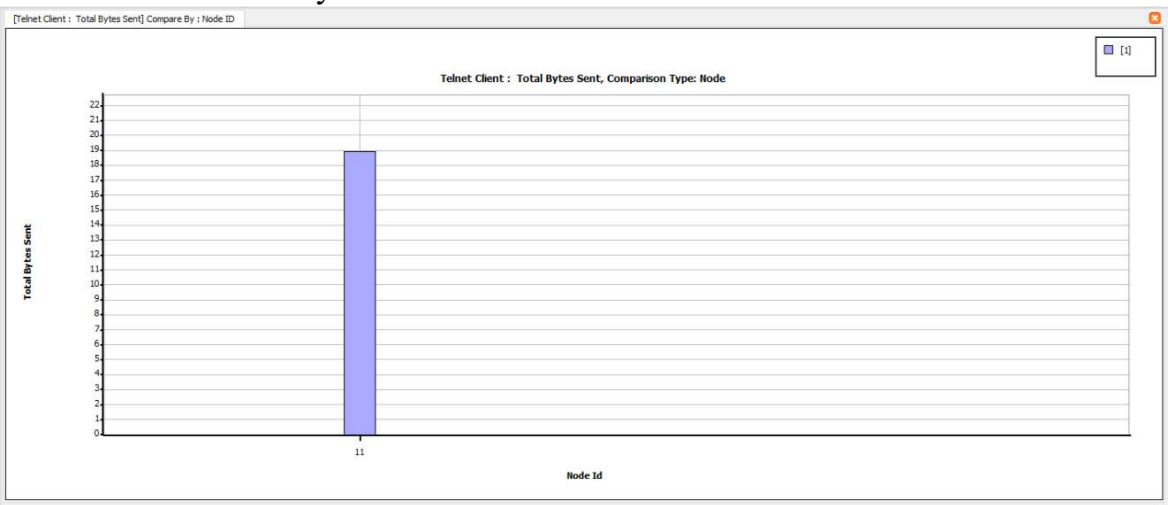


2. After making connections the nodes chosen are 5 and 18 for CBR and 11 and 8 for TELNET

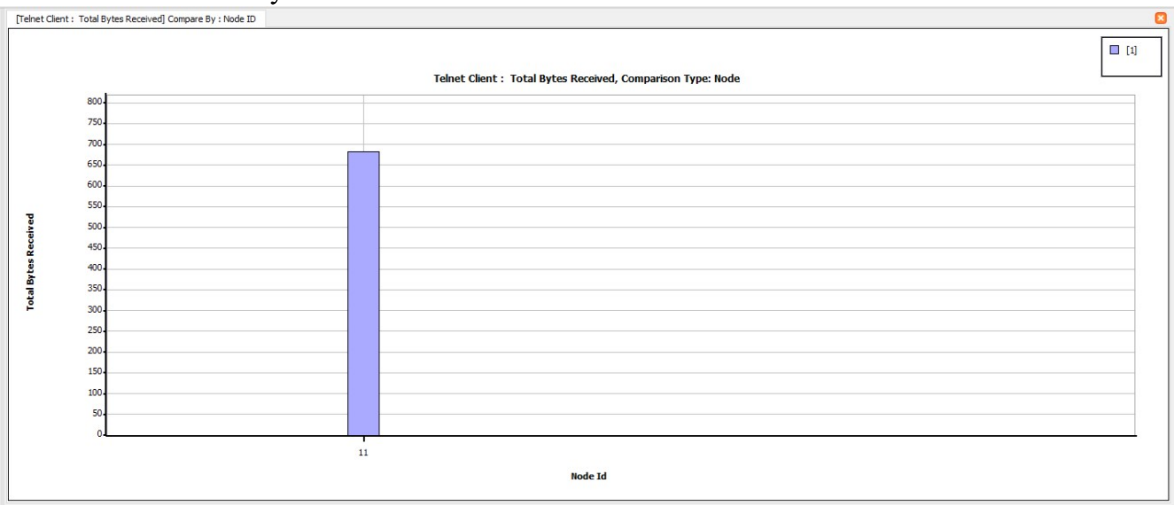


OBSERVATIONS:

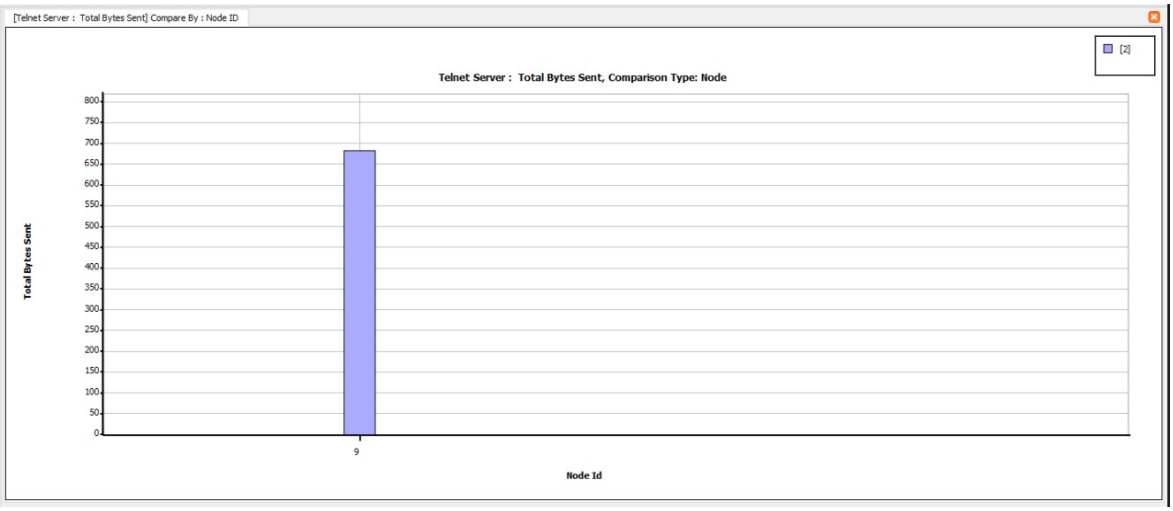
TELNET client total bytes sent



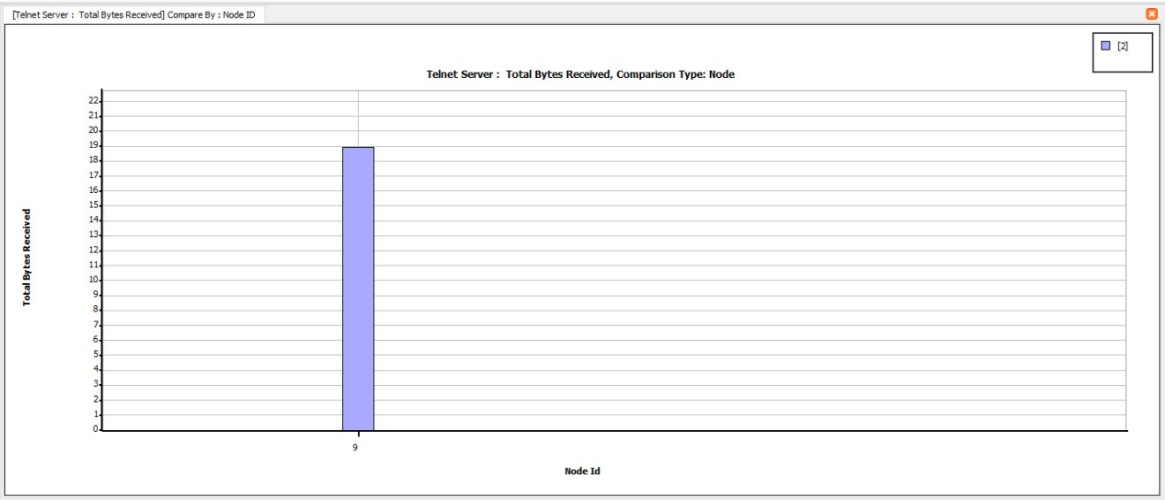
TELNET client total bytes received



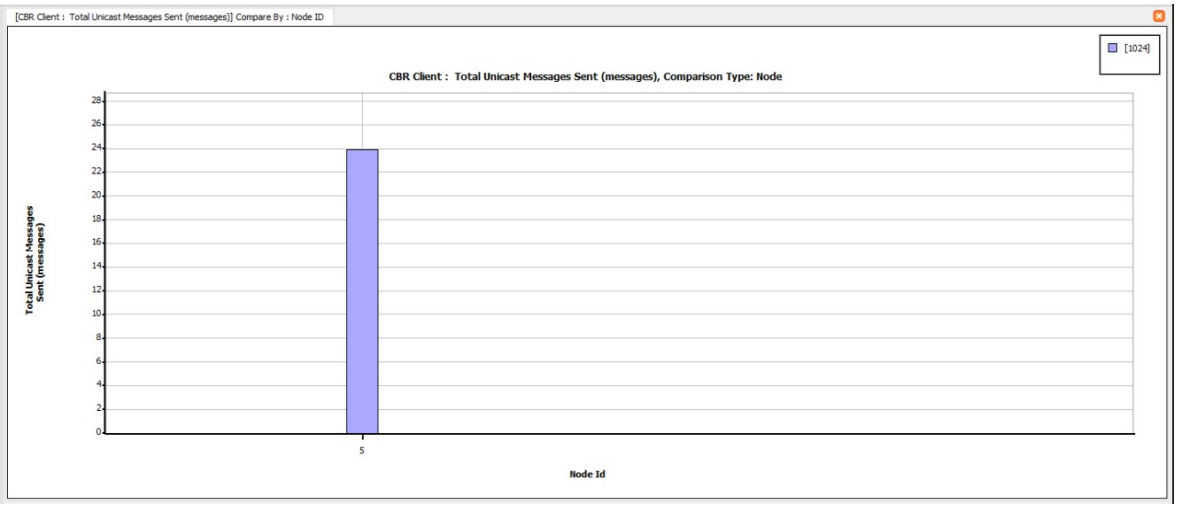
TELNET server sent



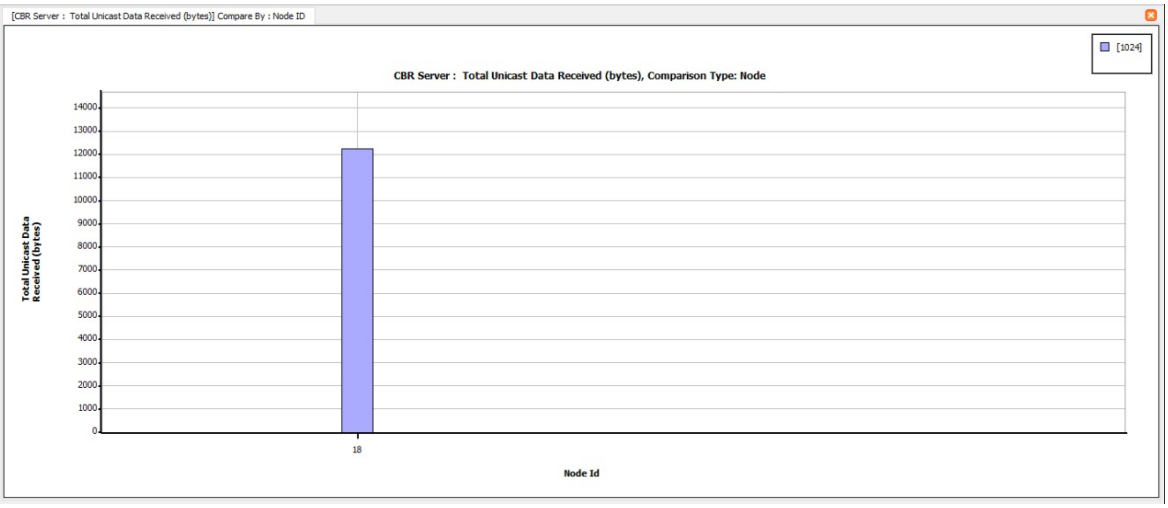
TELNET server rec

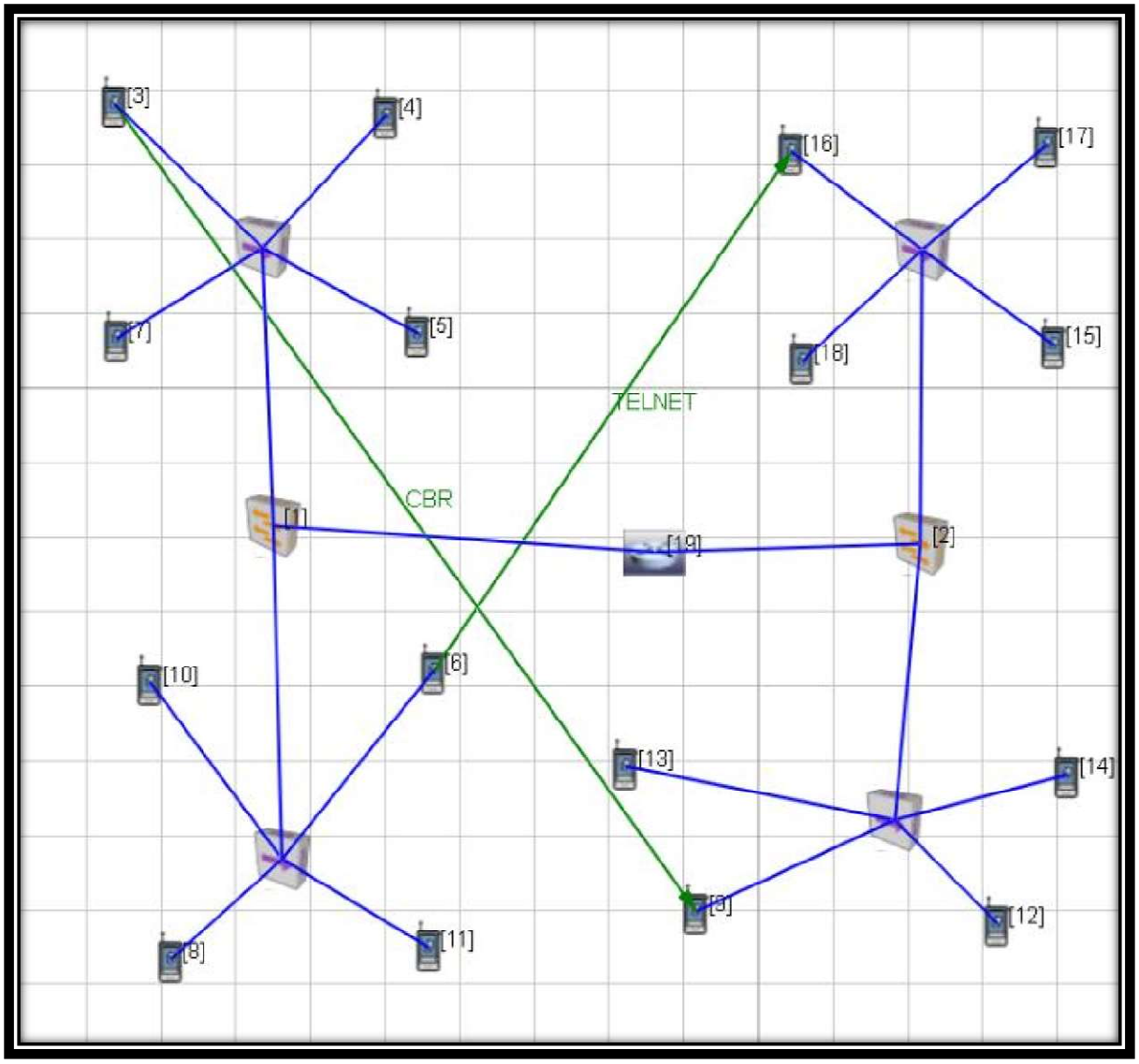


CBR client sent

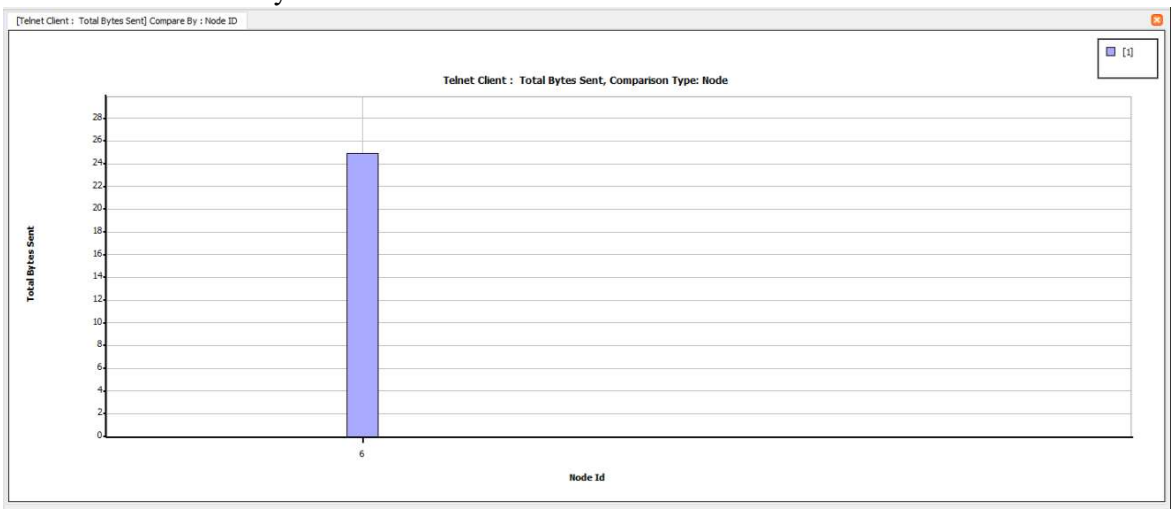


CBR server rec

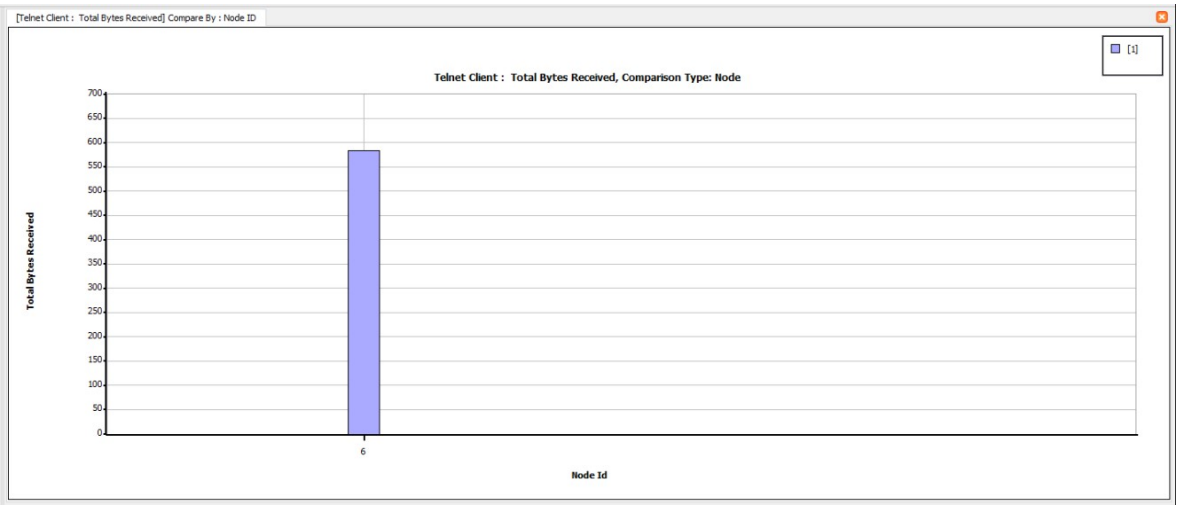




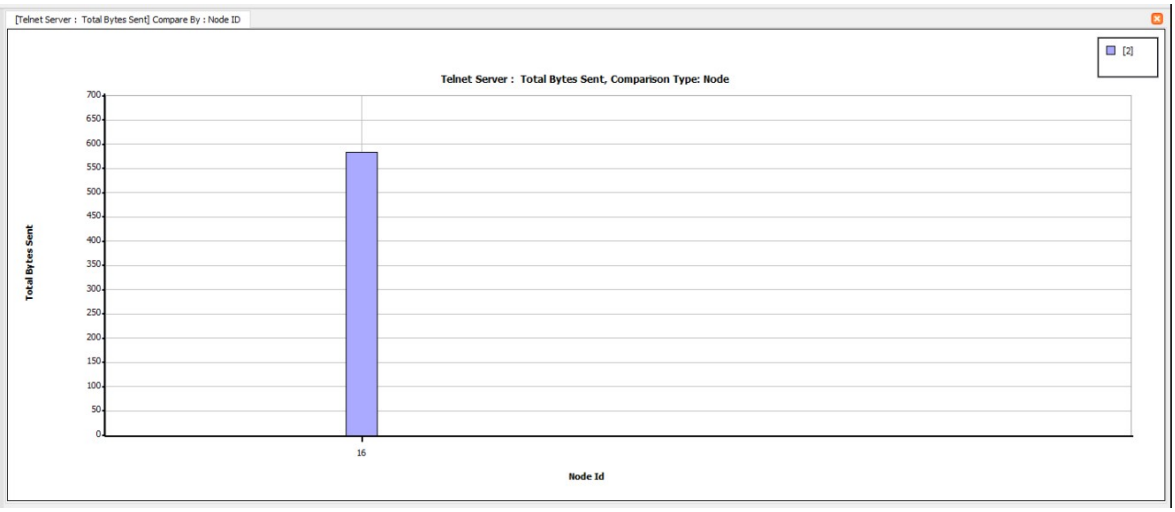
TELNET client total bytes sent



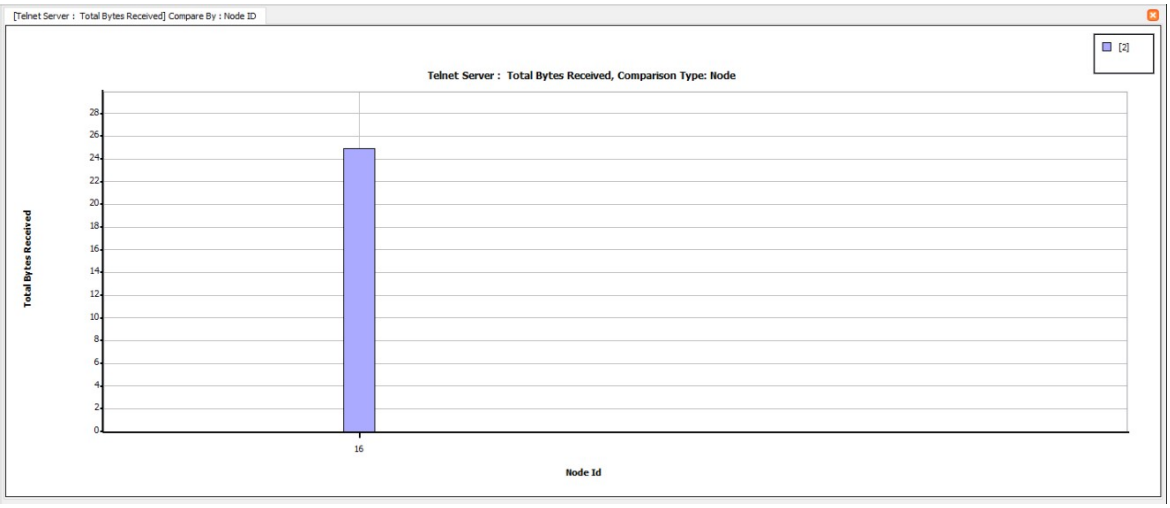
TELNET client total bytes received



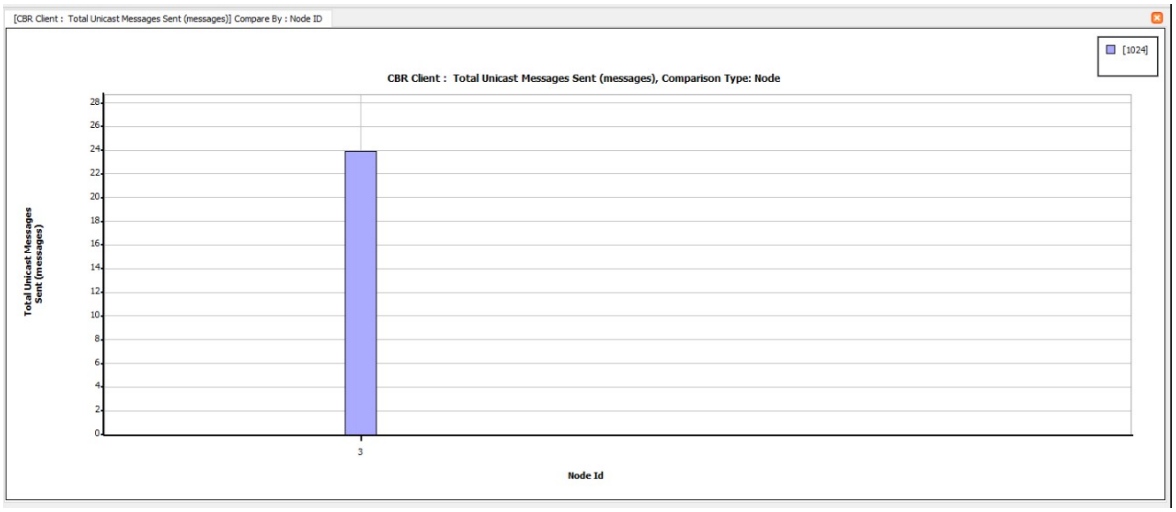
TELNET server sent



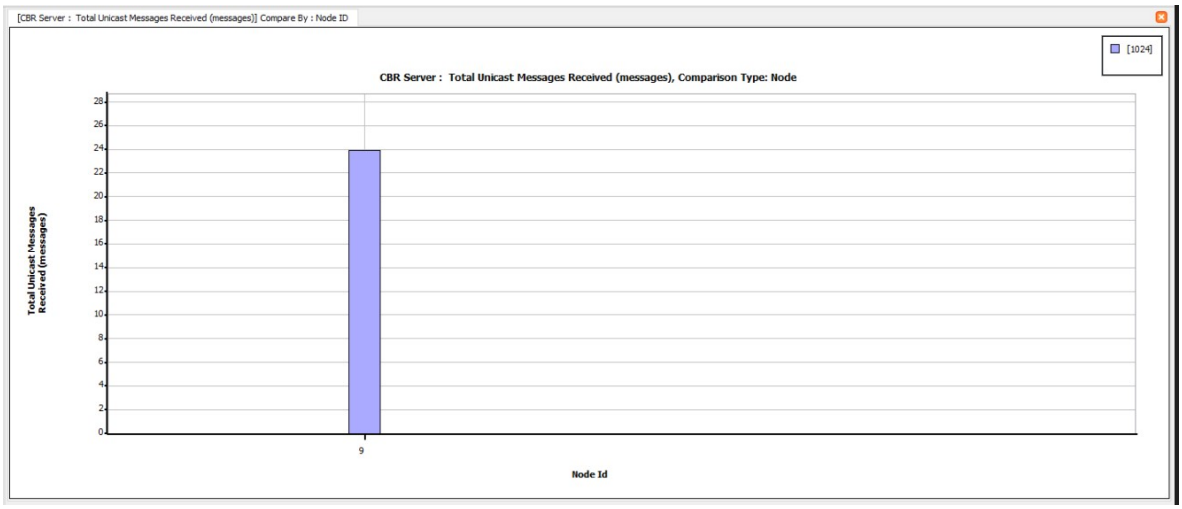
TELNET server rec



CBR client sent



CBR server received



RESULTS:

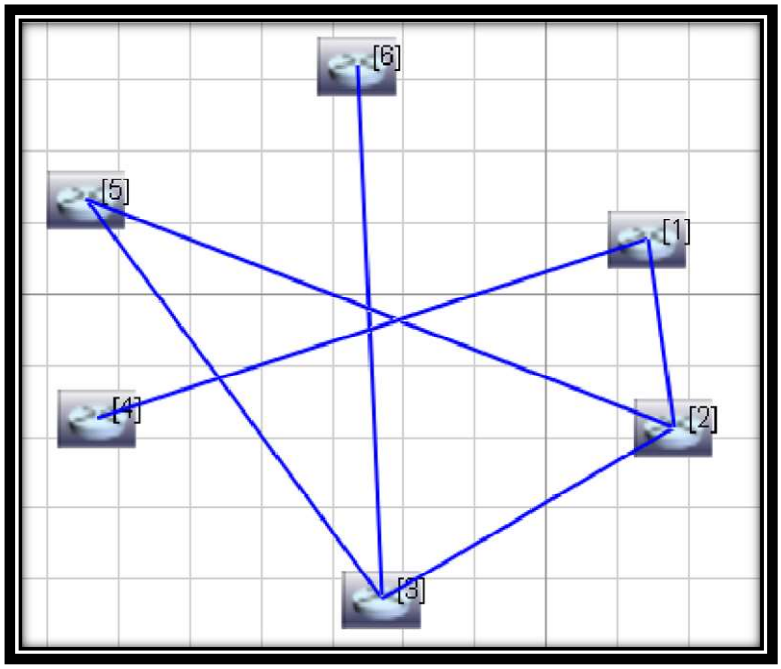
The above network is simulated and verified.

Experiment 9:

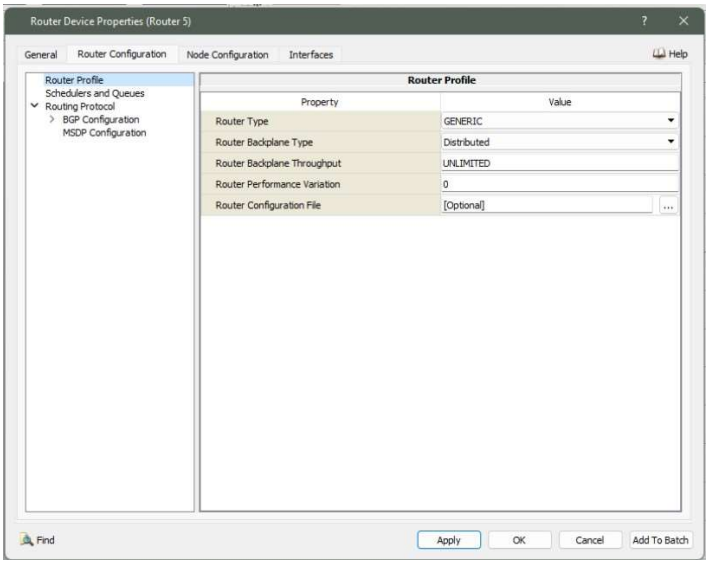
AIM: Configure a network of 5 routers with point to point connection. Apply RIP and OSPF routing algorithms and compare.

PROCEDURE:

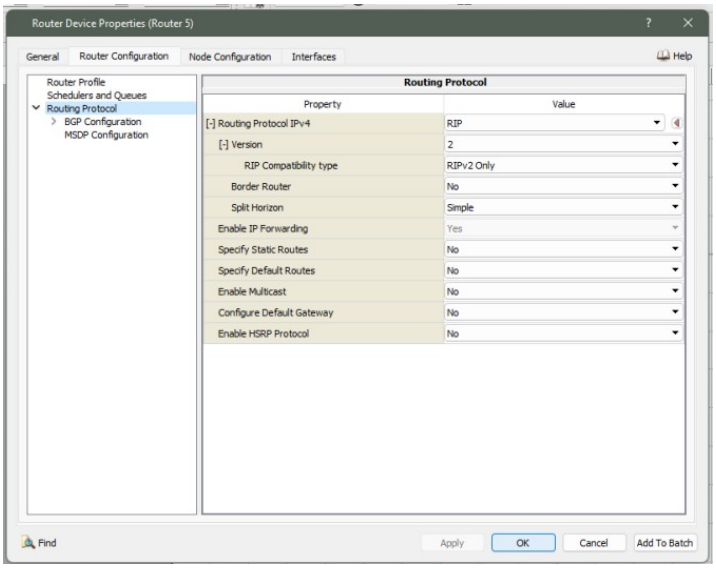
1. Make connections as shown in figure.



2. Change all the router device properties as follows



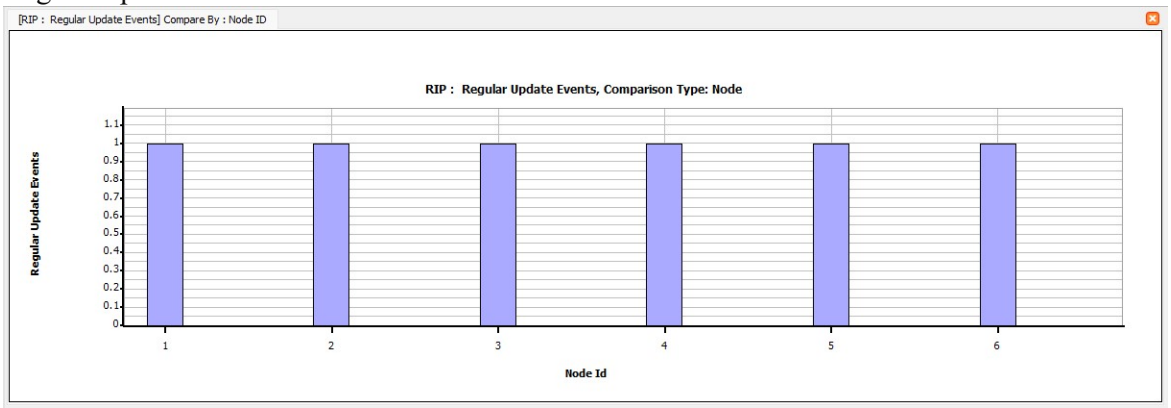
3. Configure routing protocol
   a. Select the node and under Node configuration choose routing protocol.
   b. Set the routing protocol field to the desired routing protocol.



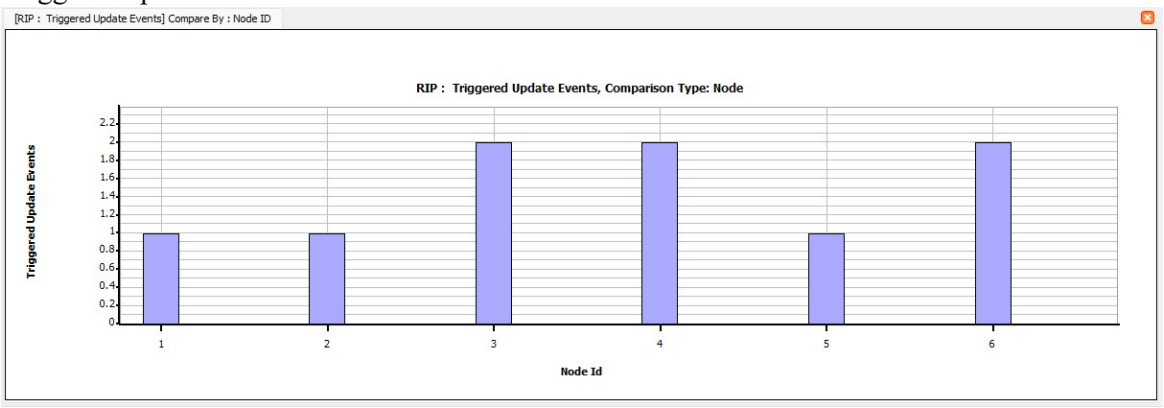
OBSERVATIONS:

RIP protocol

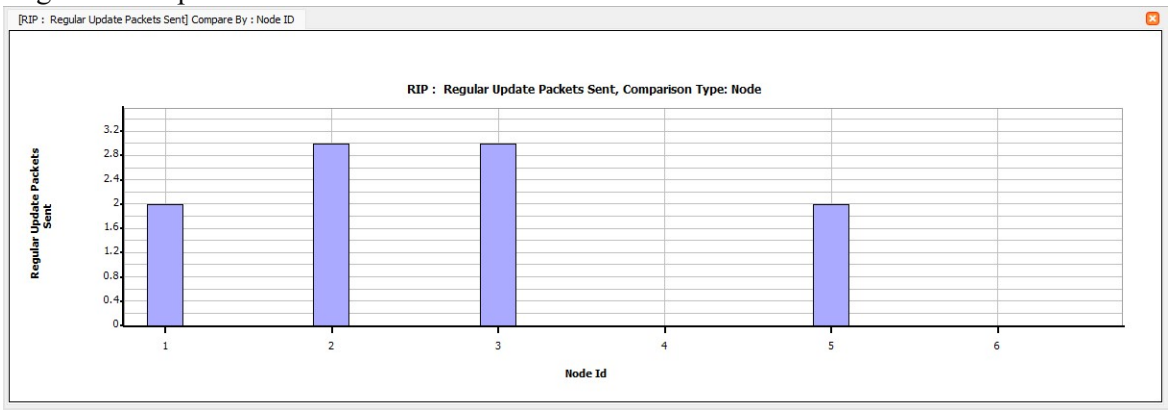
Regular update event



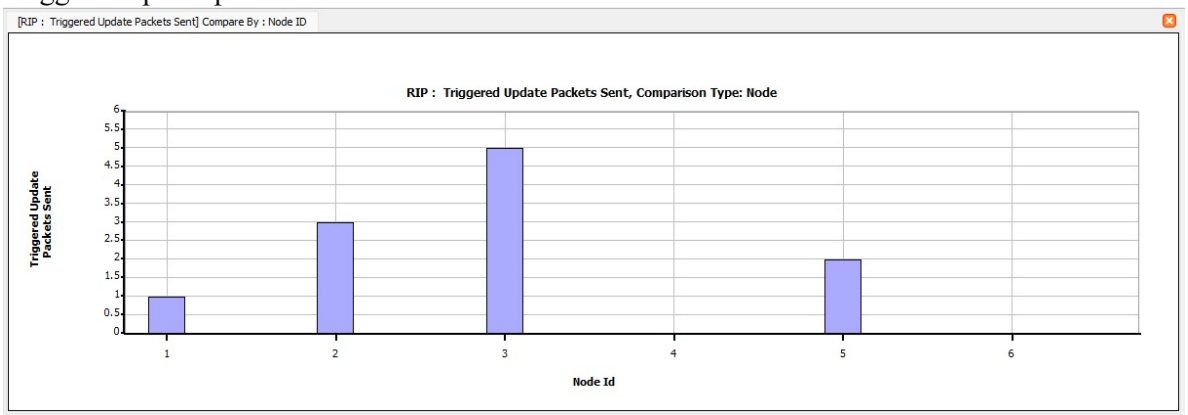
Triggered update event



Regular event packets sent

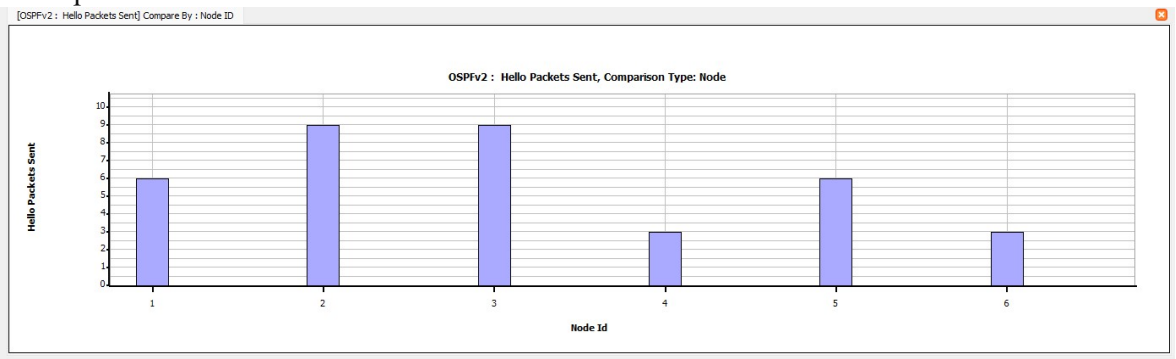


Triggered update packets sent

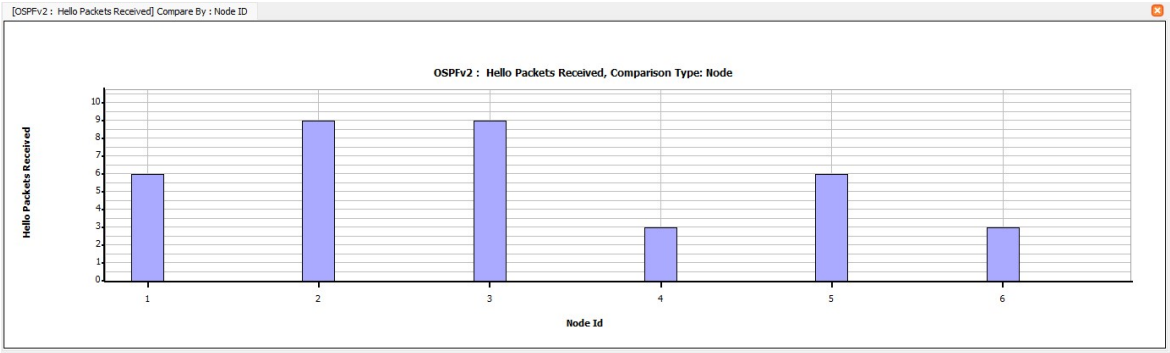


OSPF protocol

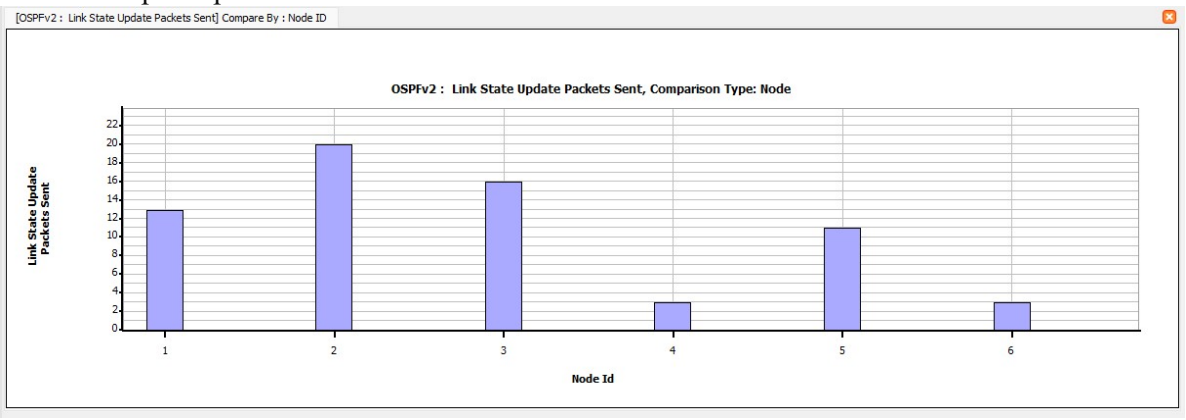
Hello packets sent



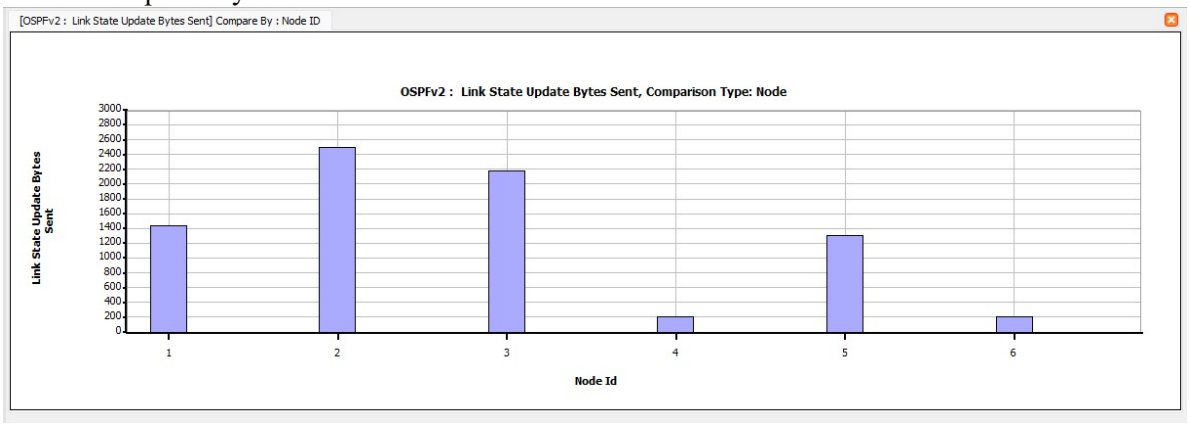
Hello packets received



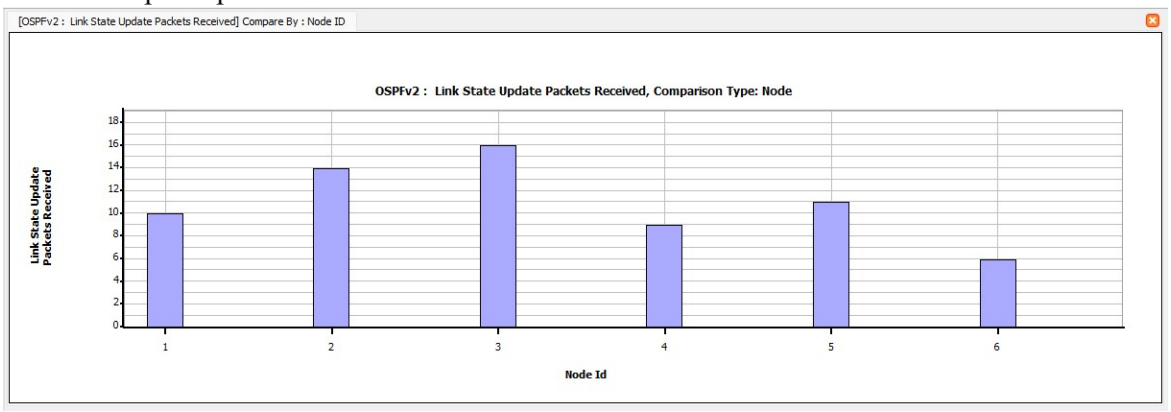
Link state update packet sent



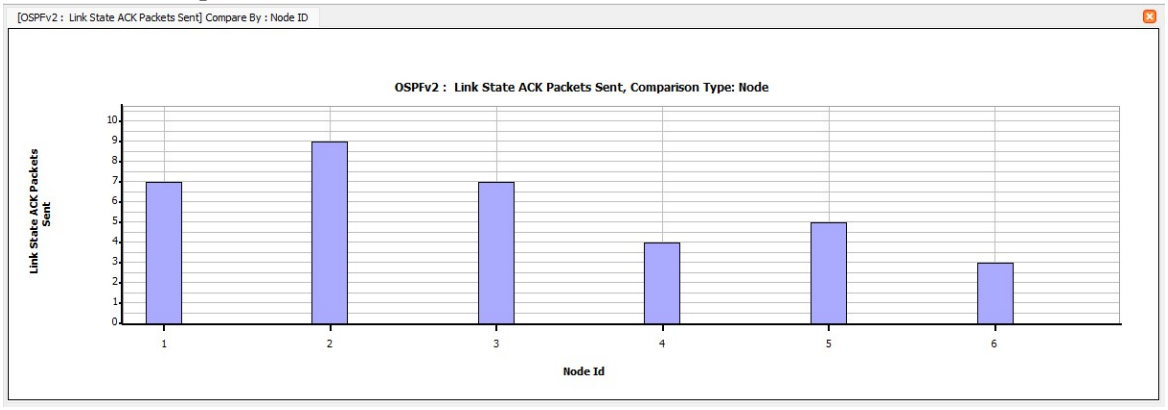
Link state update byte sent



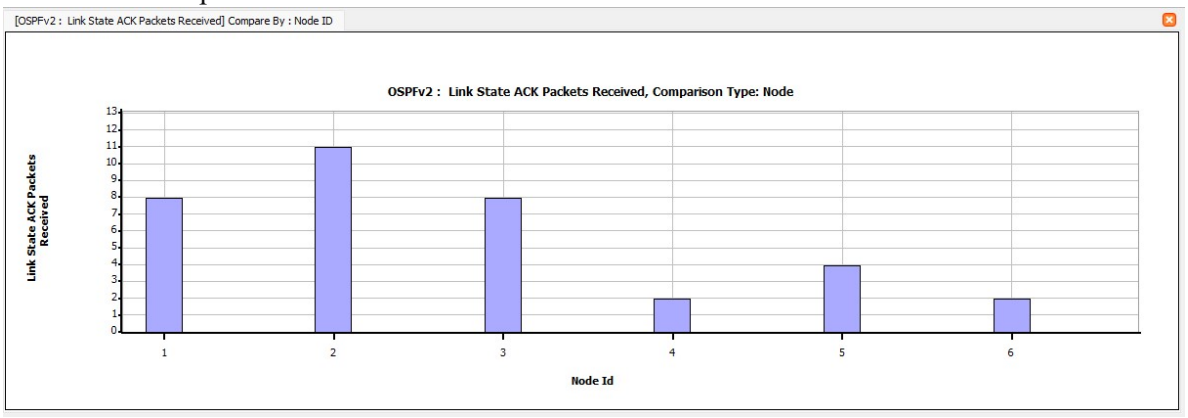
Link state update packet received



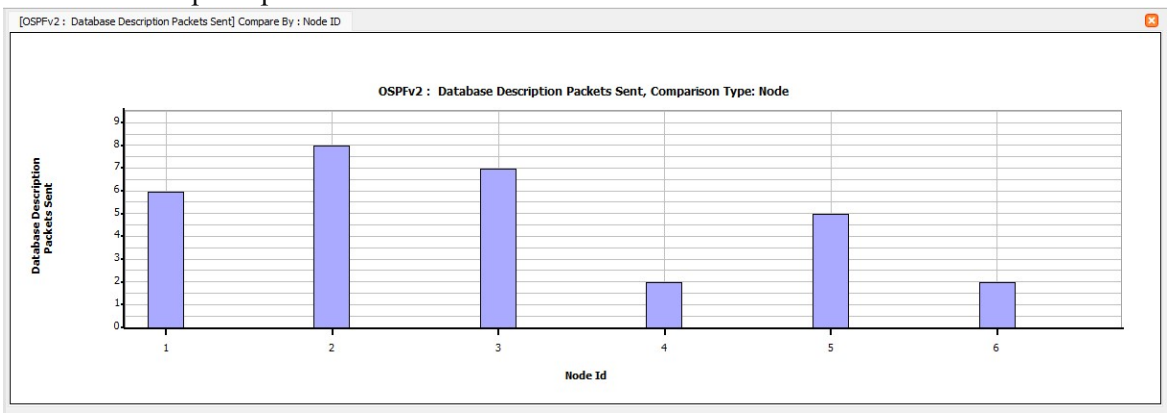
Link state ACK packets sent



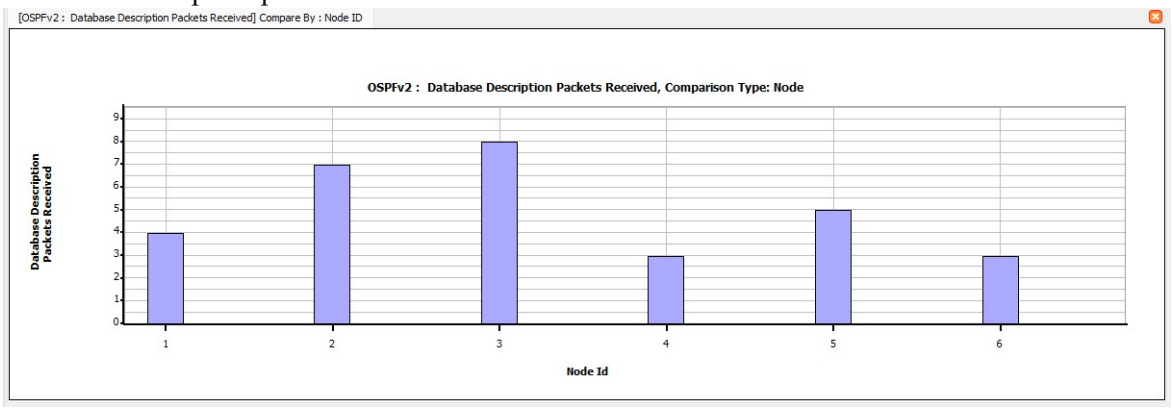
Link state ACK packets received



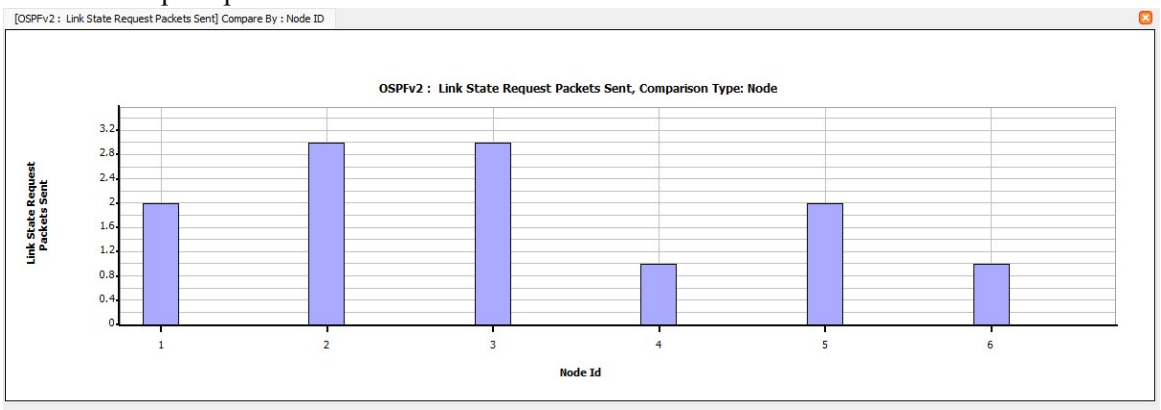
Database description packets sent



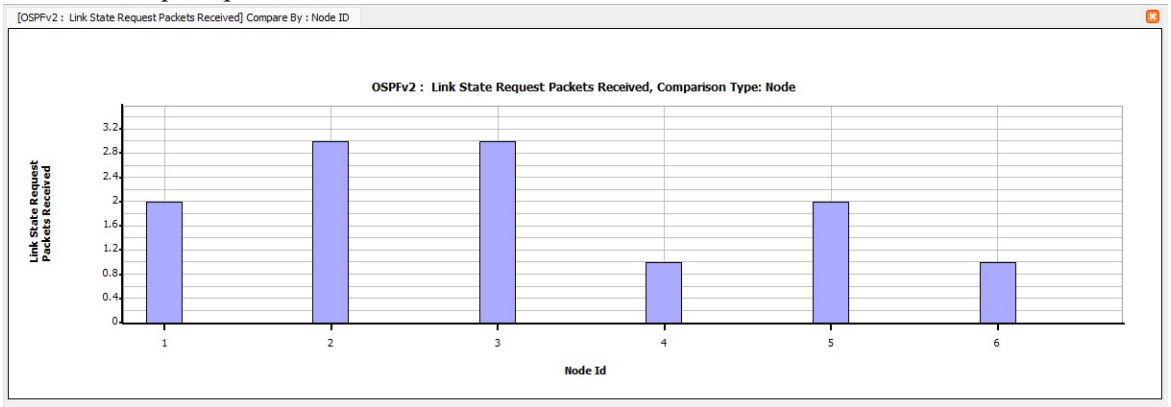
Database description packets received



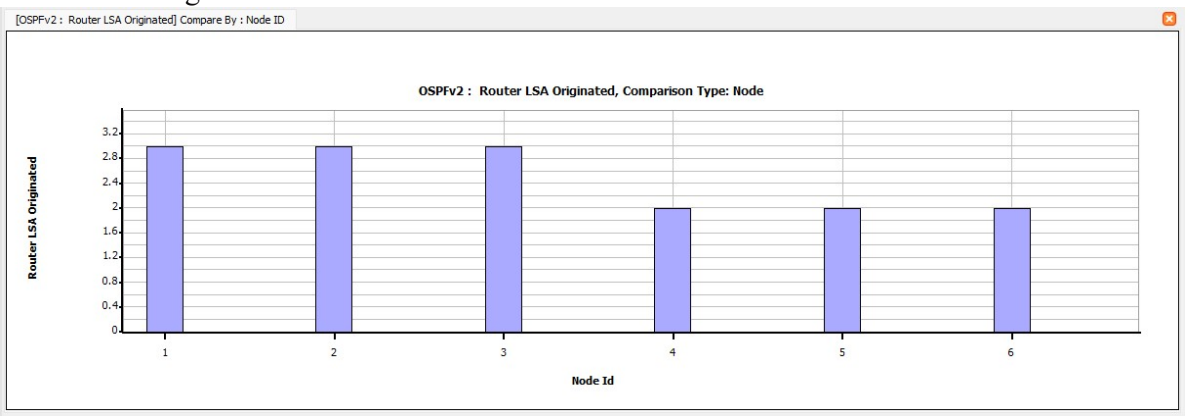
Link state request packets sent



Link state request packets received



Router LSA originated



RESULTS:

Regular update event

Node id	number of packets
1	1
2	1
3	1
4	1
5	1
6	1

Triggered update event

Node id	number of packets
1	1
2	1
3	2
4	2
5	1
6	2

Regular event packets sent

Node id	number of packets
1	2
2	3
3	3
4	0
5	2
6	0

Triggered update packets sent

Node id	number of packets
1	1
2	3
3	5
4	0
5	2
6	0

Hello packets sent

Node id	number of packets
1	6
2	9
3	9
4	3
5	6
6	3

Hello packets received

Node id	number of packets
1	6
2	9
3	9
4	3
5	6
6	3

Link state update packet sent

Node id	number of packets
1	13
2	20
3	16
4	3
5	12
6	3

Link state update byte sent

Node id	number of packets
1	1400
2	2500
3	2200
4	200
5	1300
6	200

Link state update packet received

Node id	number of packets
1	10
2	14
3	16
4	9
5	11
6	6

Link state ACK packets sent

Node id	number of packets
1	7
2	9
3	7
4	4
5	5
6	3

Link state ACK packets received

Node id	number of packets
1	8
2	11
3	8
4	2
5	4
6	2

Database description packets sent

Node id	number of packets
1	6
2	8
3	7
4	2
5	5
6	2

Database description packets received

Node id	number of packets
1	4
2	7
3	8
4	3
5	5
6	3

Link state request packets sent

Node id	number of packets
1	2
2	3
3	3
4	1
5	2
6	1

Link state request packets received

Node id	number of packets
1	2
2	3
3	3
4	1
5	2
6	1

Router LSA originated

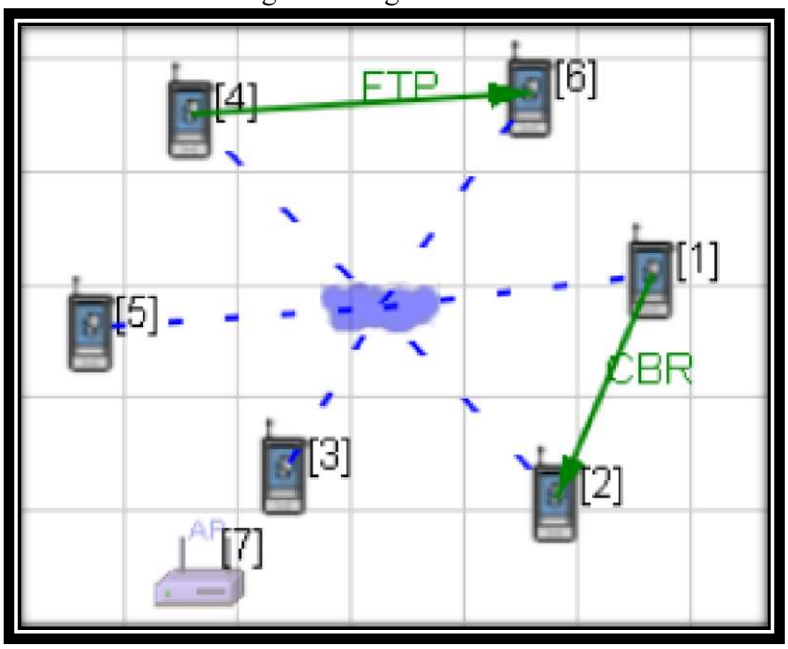
Node id	number of packets
1	3
2	3
3	3
4	2
5	2
6	2

Experiment 10:

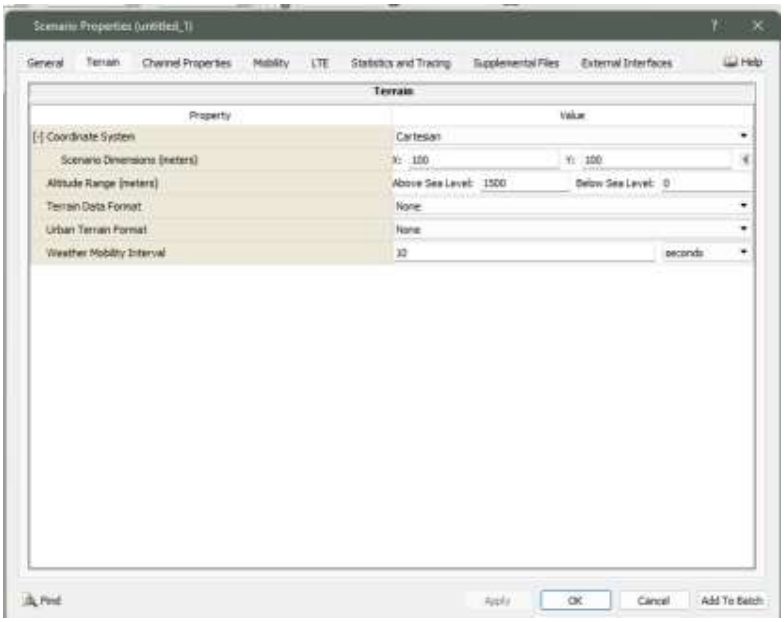
AIM: Simulate a wireless infrastructure network with 6 nodes and analyze.

PROCEDURE:

1. Make connections as given in figure

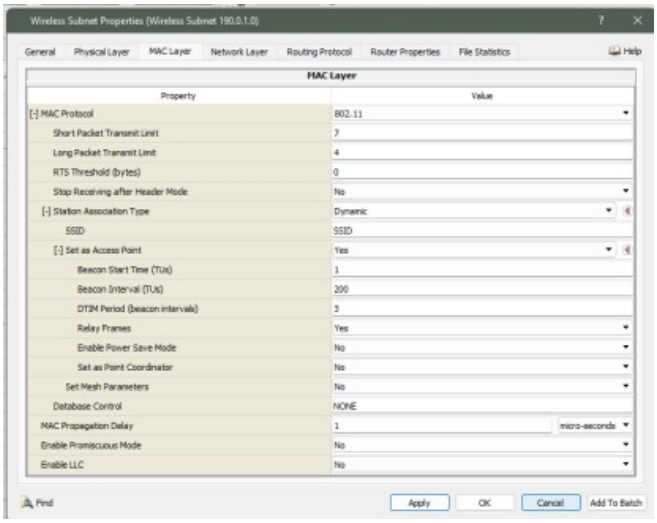


2. Change the terrain dimensions in scenario generation as shown



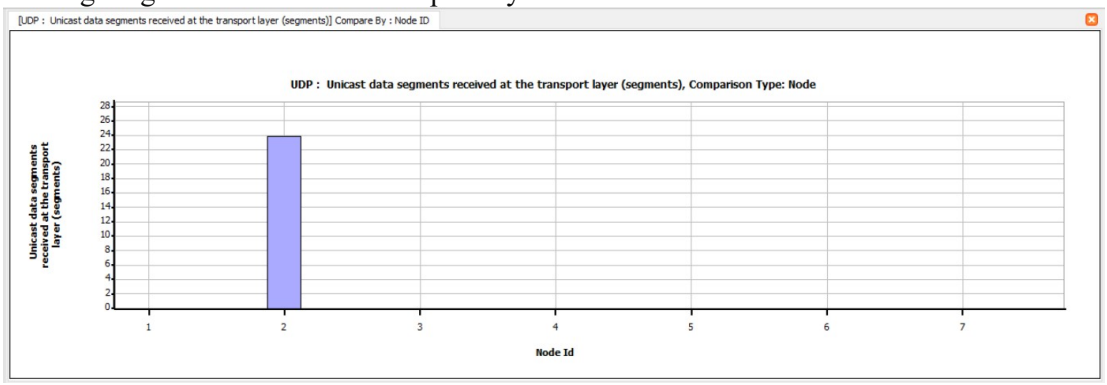
3. MAC layer properties of the cloud are changed as follows.

   a. Select the subnet Set station association type to Dynamic Station scan type to active
   b. To set a node as access point
   c. Select the node and from properties select interface0
   d. Select MAC layer and set access point field to yes

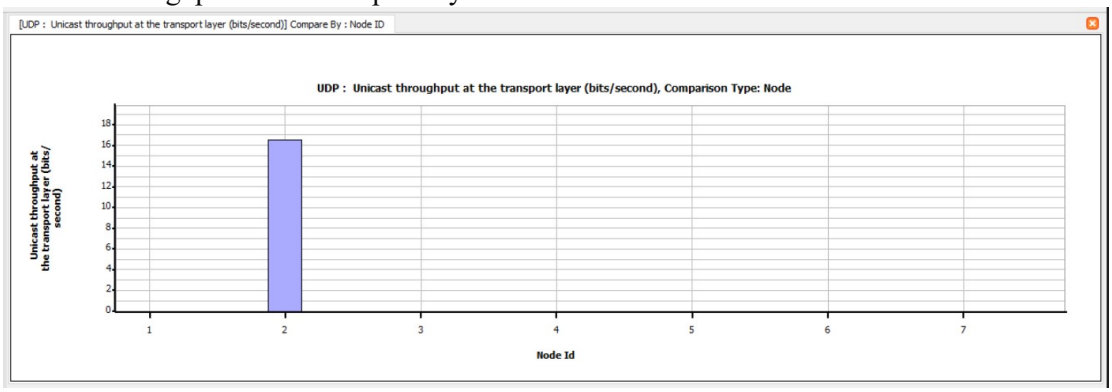


OBSERVATIONS:

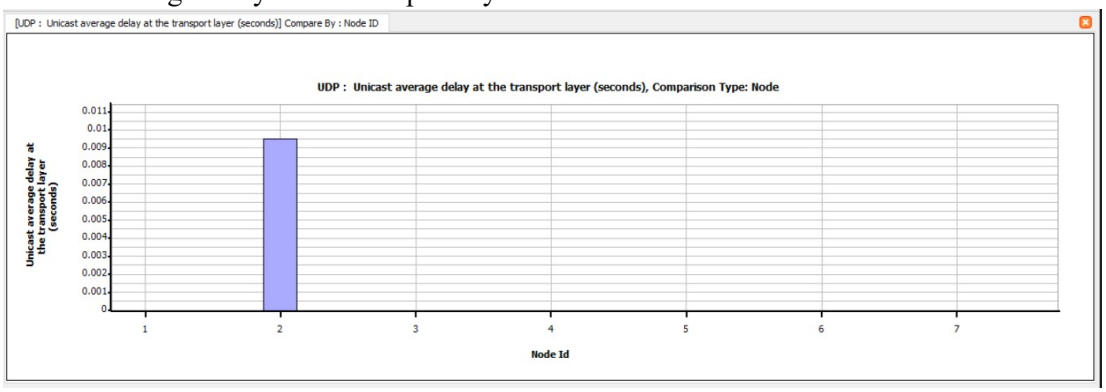
Message segments received at transport layer



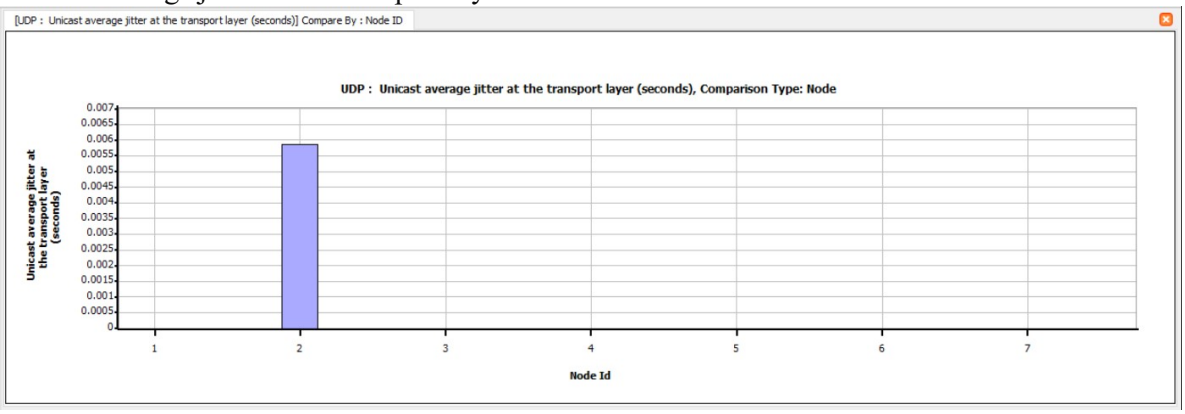
Unicast throughput at the transport layer



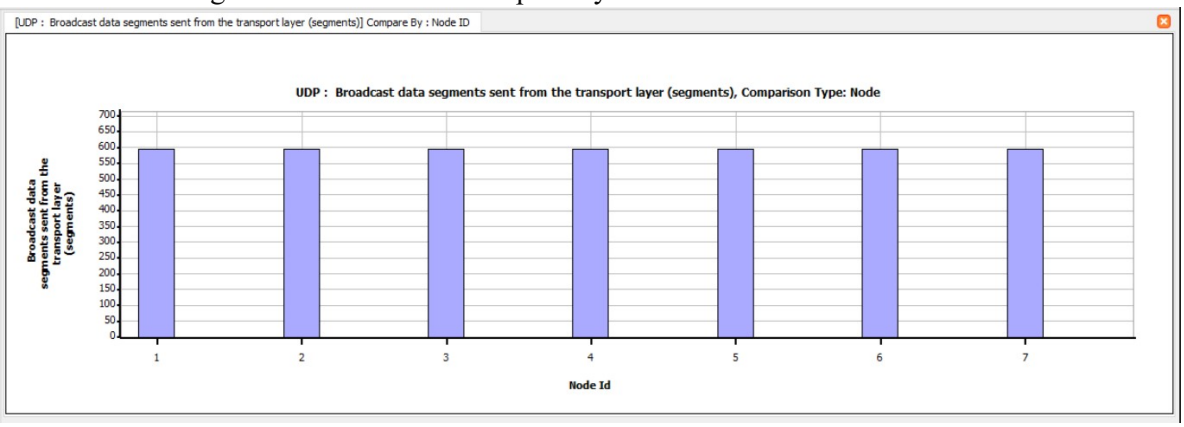
Unicast average delay at the transport layer



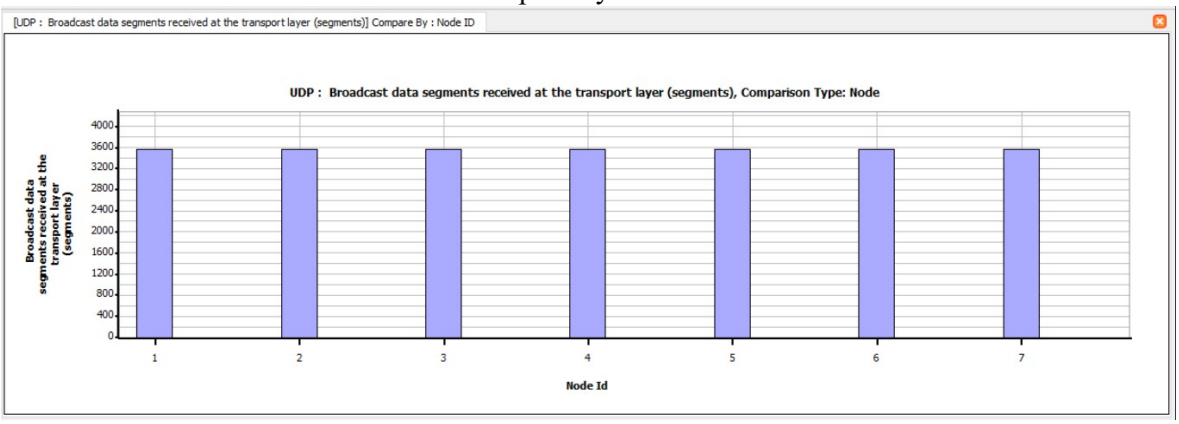
Unicast average jitter at the transport layer



Broadcast data segment sent from the transport layer



Broadcast data received sent from the transport layer



RESULTS:

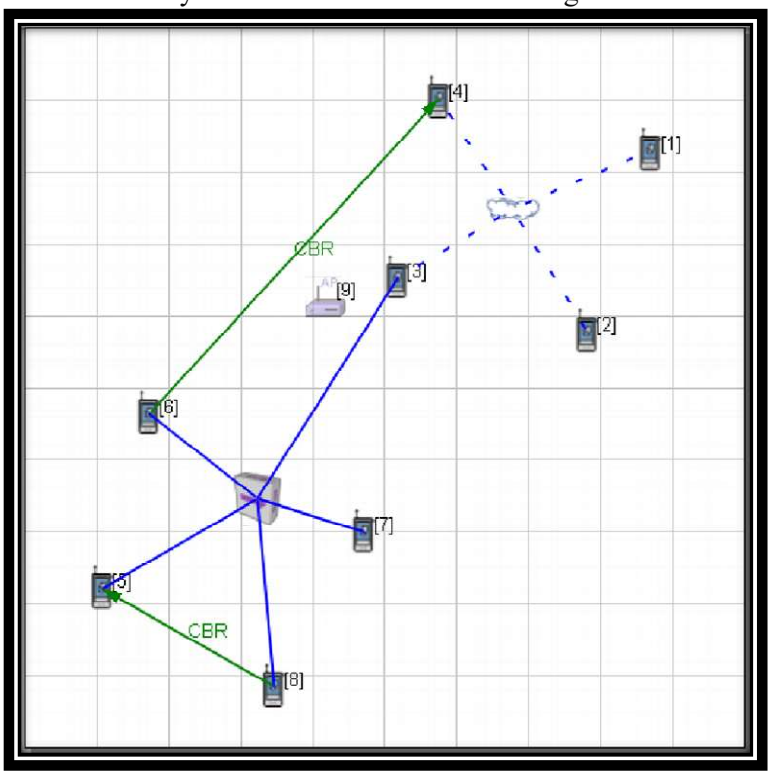
Hence, the wireless infrastructure network is simulated and it's various parameters is observed.

Experiment 11:

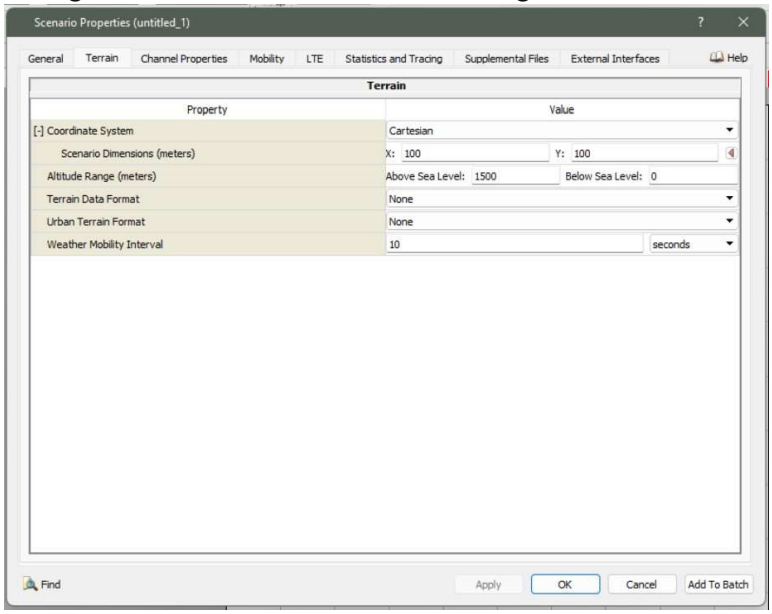
AIM: Configure a wired network with 4 nodes and wireless infrastructure network with 4 nodes apply relevant TCP and UDP applications from a node in wired network to a node in wireless network and analyze.

PROCEDURE:

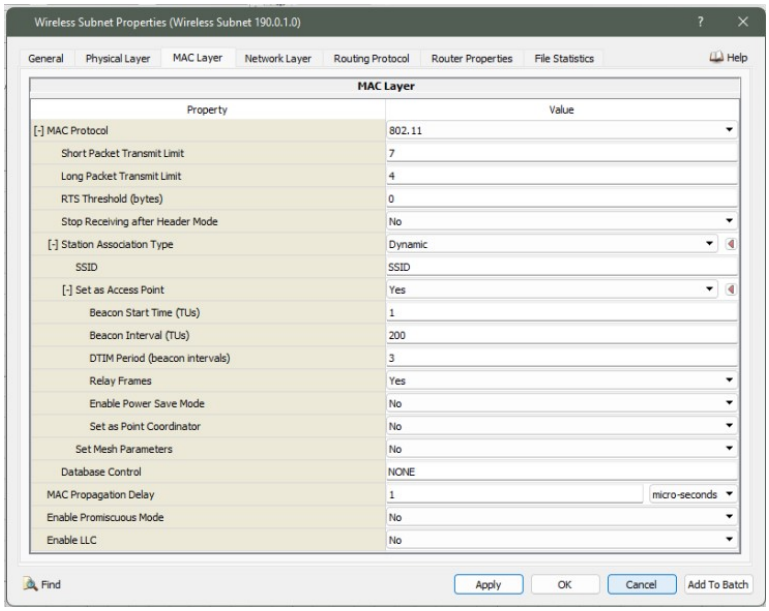
1. Make necessary connections as shown in the figure



2. Change the terrain dimensions in scenario generation as shown.



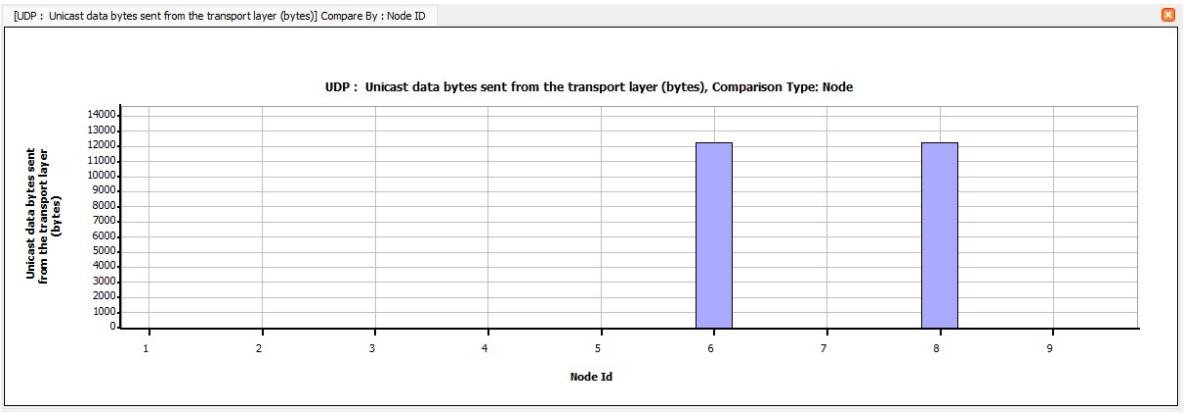
3. MAC layer properties of the cloud is changed as follows.
   a. Select the subnet Set station association type to Dynamic Station scan type to active
   b. To set a node as access point, Select the node and from properties select interface0
   c. Select MAC layer and set as access point field to yes



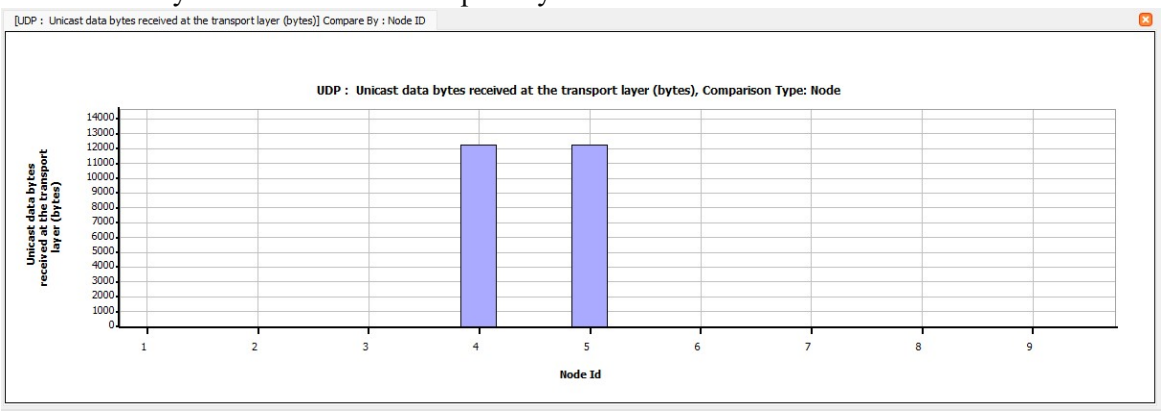
OBSERVATIONS:

UDP

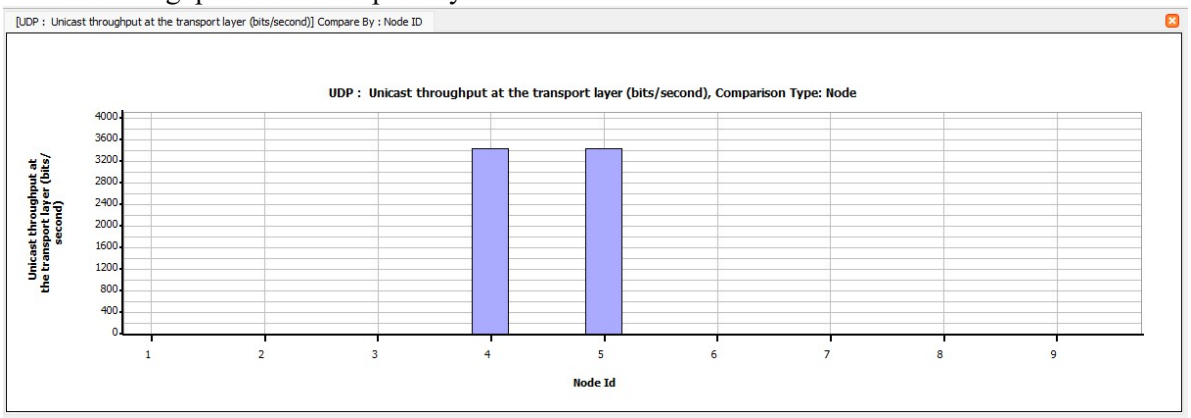
Unicast data bytes sent from transport layer



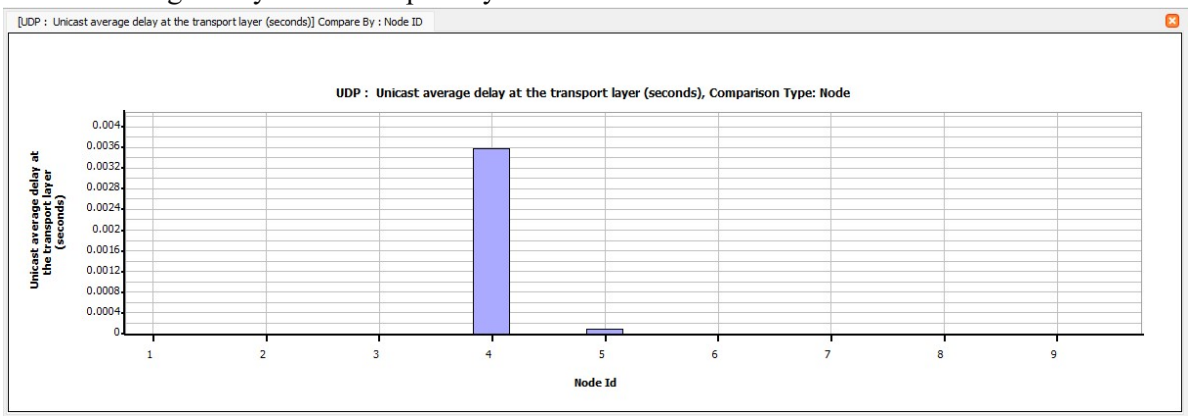
Unicast data bytes received from transport layer



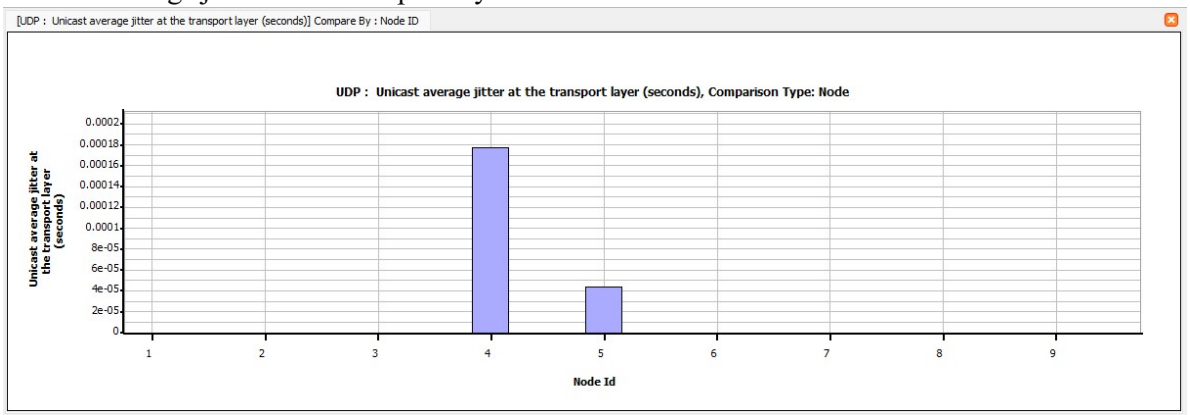
Unicast throughput at the transport layer



Unicast average delay at the transport layer

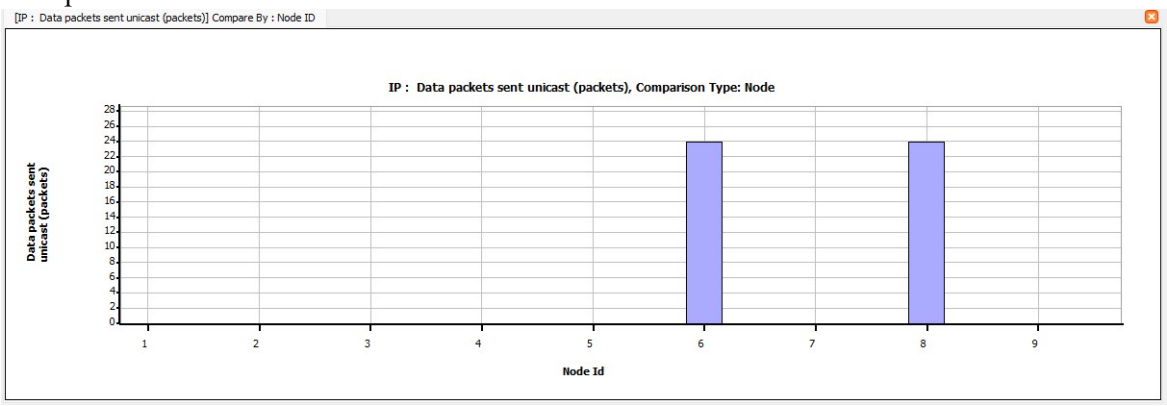


Unicast average jitter at the transport layer

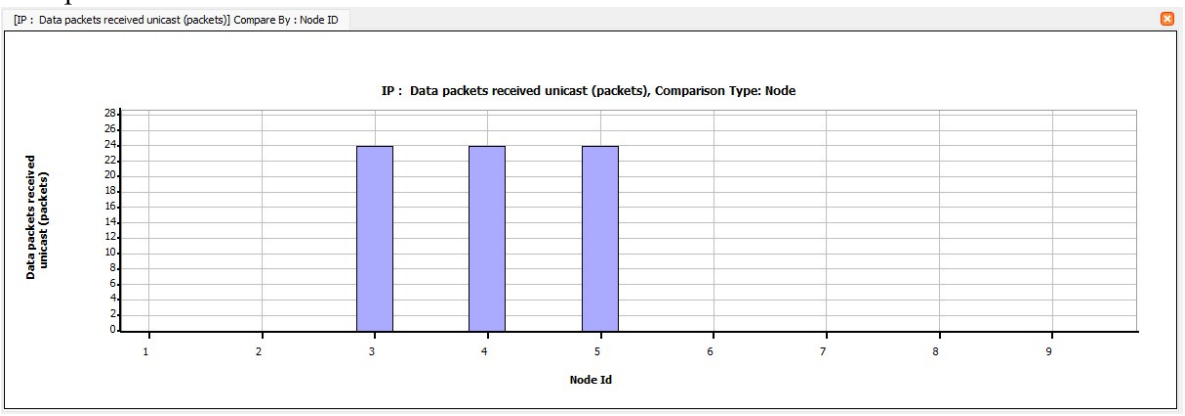


IP

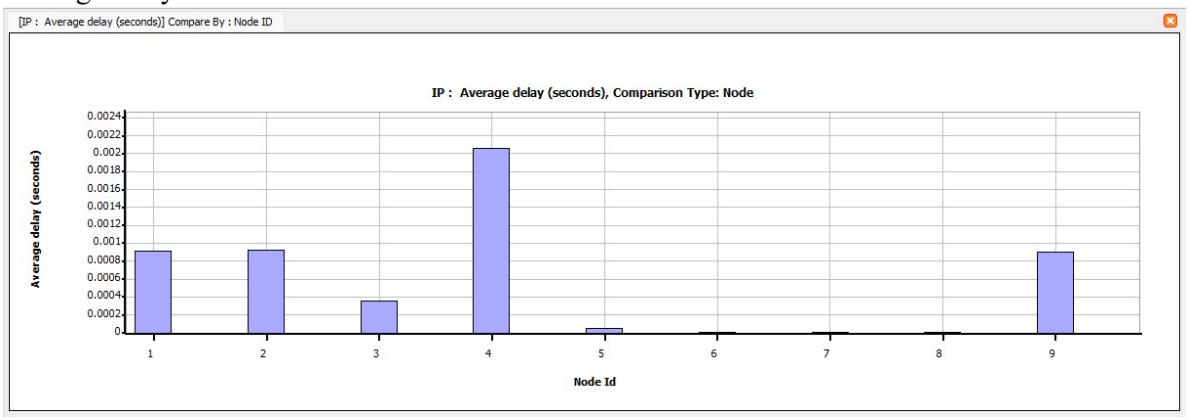
Data packets sent unicast



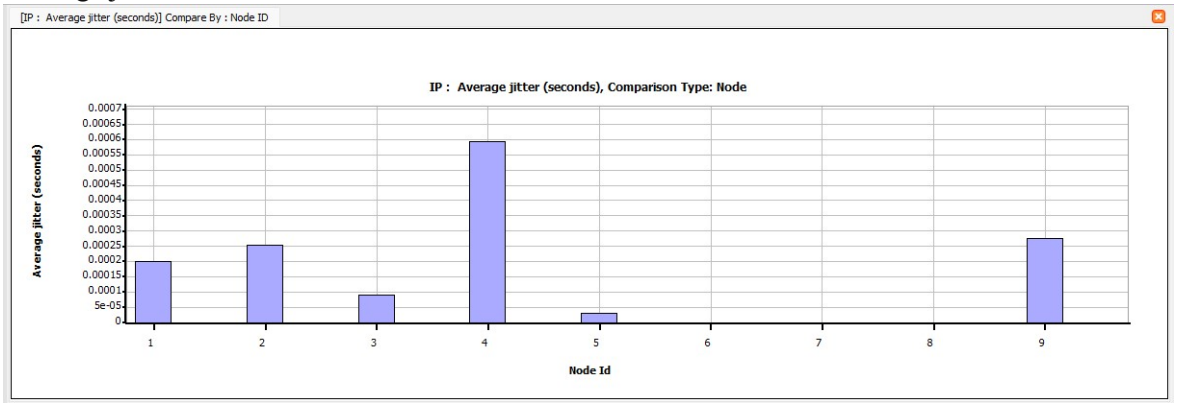
Data packets received unicast



Average delay



Average jitter



RESULTS:

The devices communicates with each other through an access point. Here node 3 is the AP. The access point communicates with LAN with the help of a switch. The required parameters are recorded above.

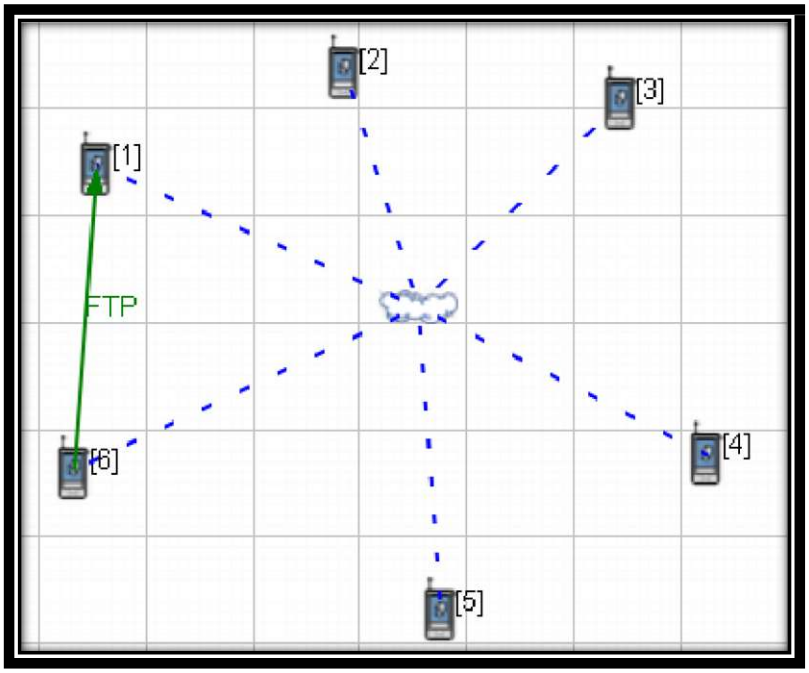
Experiment 12:

AIM:

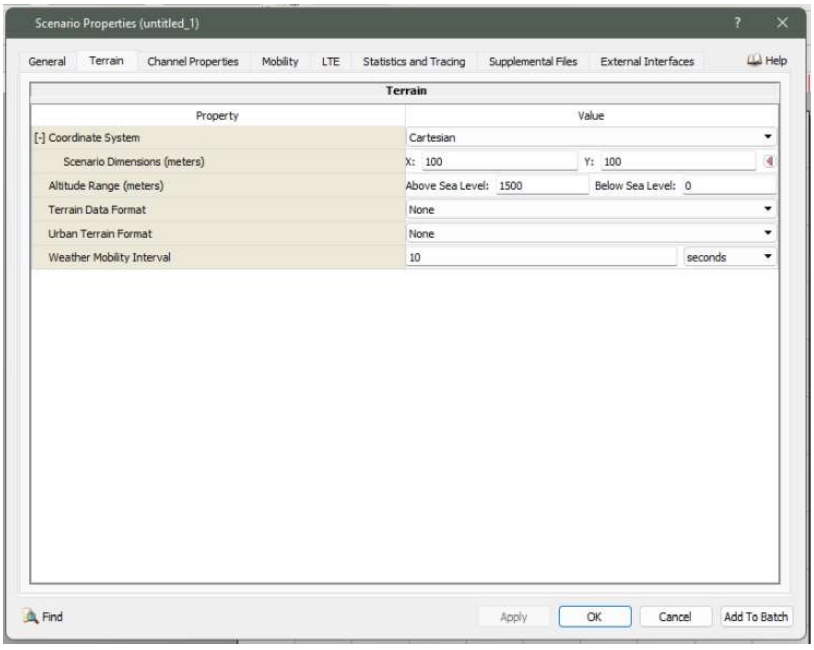
a. Simulate wireless ad hoc network with 6 nodes give mobility to a node and analyze.
b. give mobility to all the nodes.

PROCEDURE:

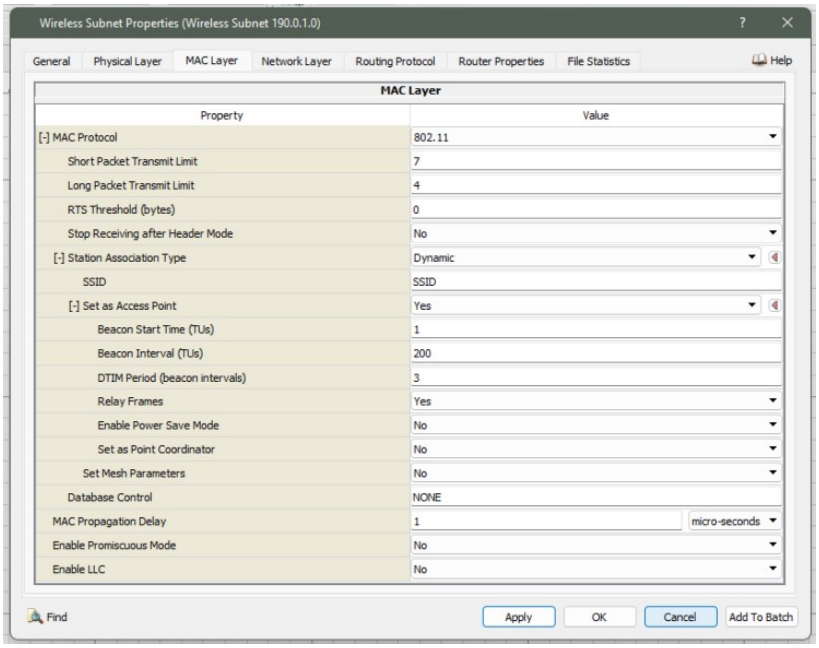
1. Make connection as given in figure below.



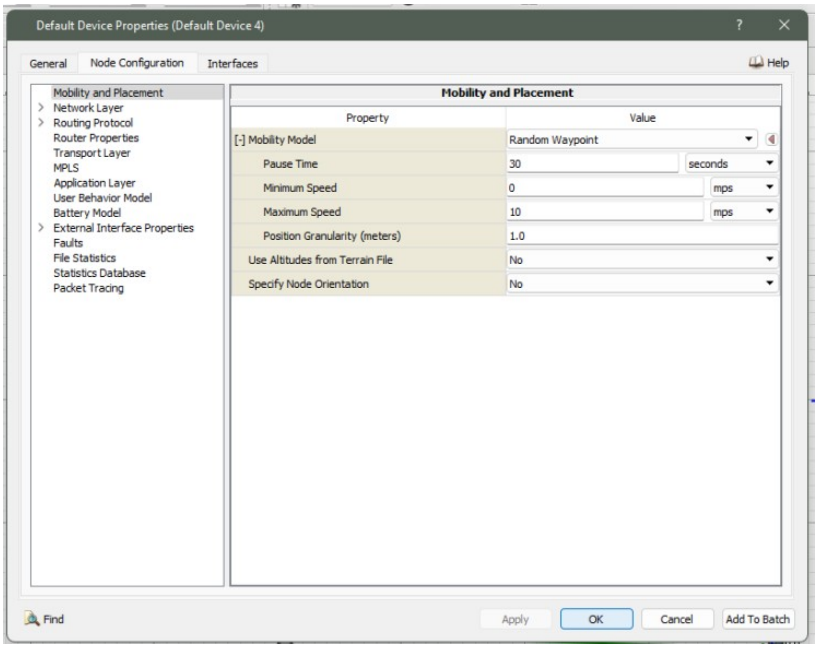
2. Change the terrain dimensions in scenario generation as shown



3. MAC layer properties of the cloud is changed as follows.
   a. Select the subnet Set station association type to Dynamic Station scan type to active
   b. To set a node as access point 1. Select the node 2. From properties select interface0 3. Select MAC layer 4. Set as access point field to yes

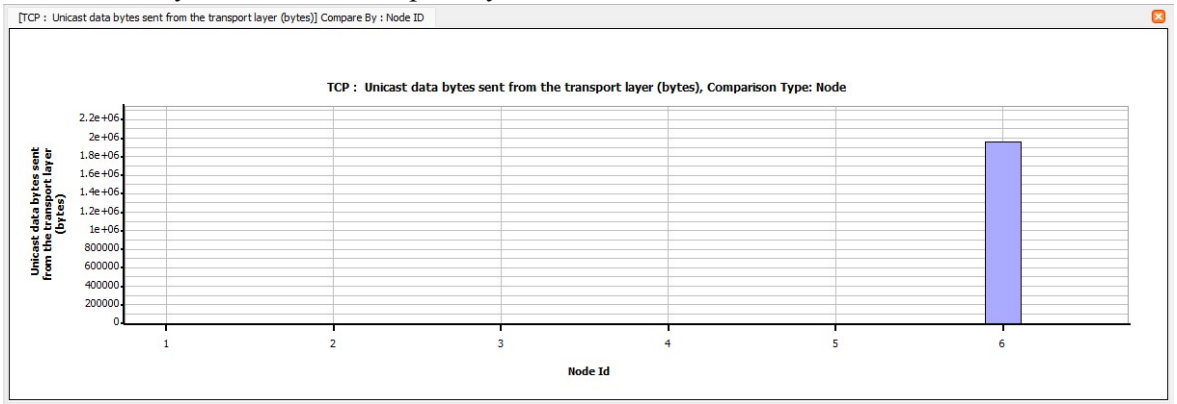


4. To give mobility to all the nodes Select the nodes Go to properties, node configuration choose the mobility model

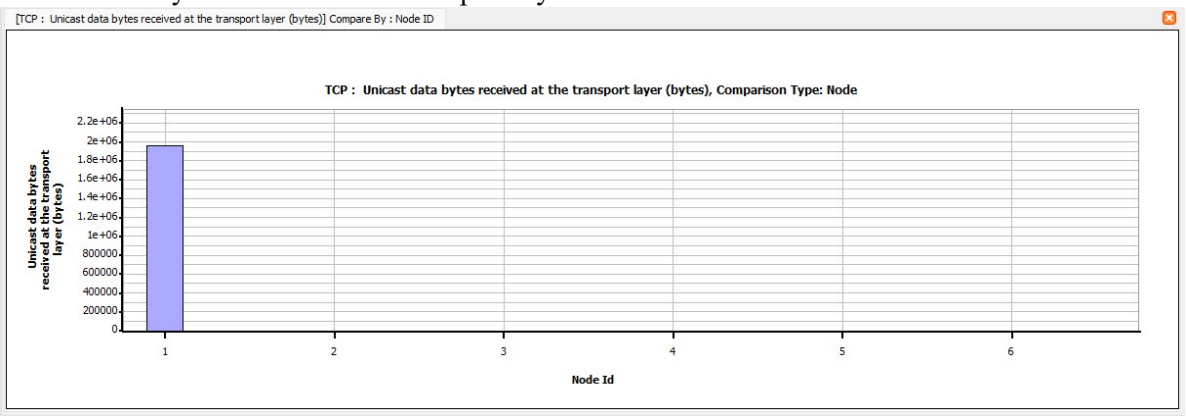


OBSERVATIONS:

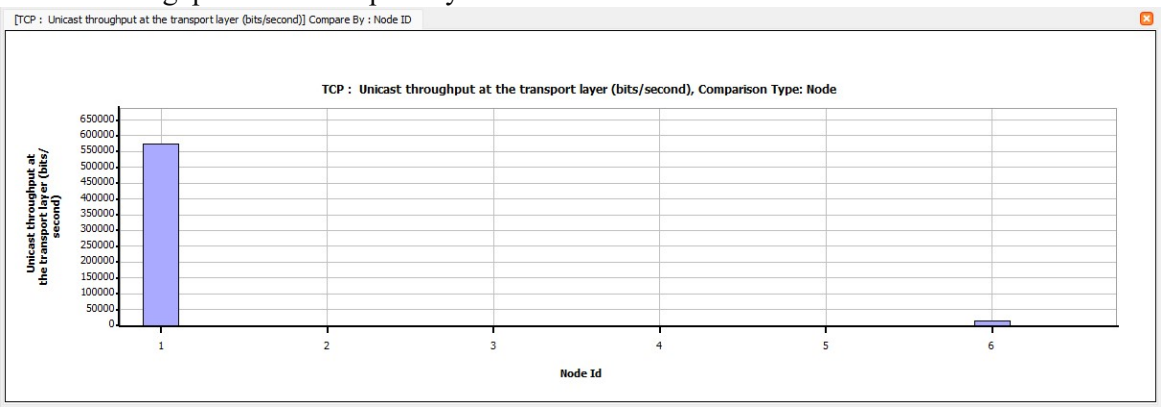
Unicast data bytes sent from transport layer



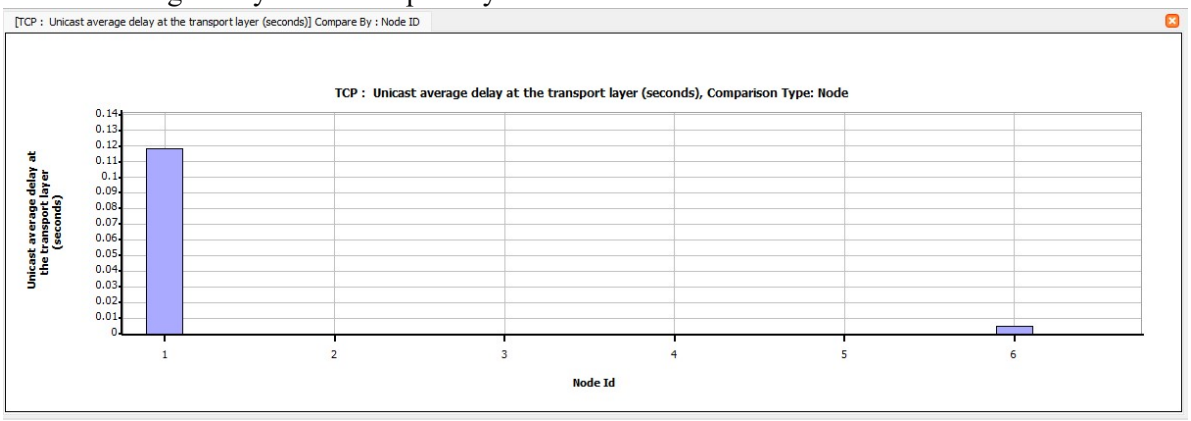
Unicast data bytes received from transport layer



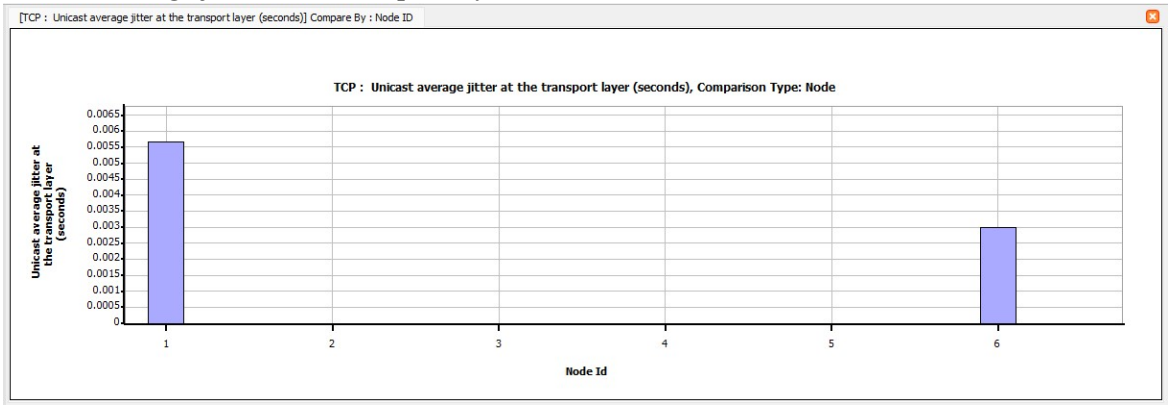
Unicast throughput at the transport layer



Unicast average delay at the transport layer



Unicast average jitter at the transport layer



RESULTS:

1. Here the mobility is given to only one node and observations were made.
2. Mobility to all nodes were given and all the nodes and data transfer is observed.