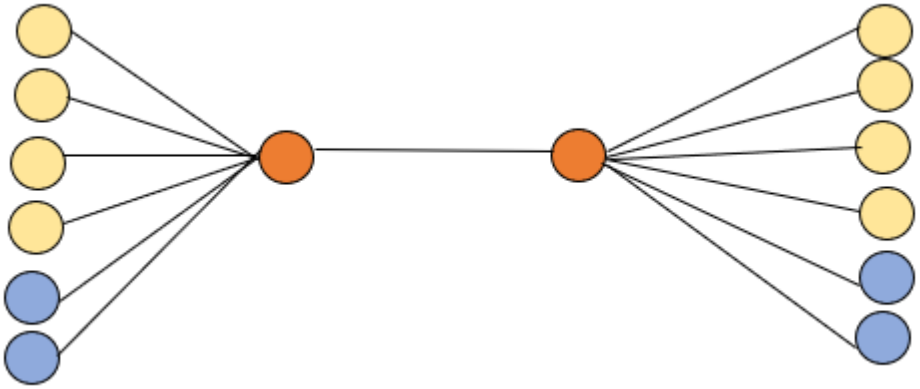
NAME - ABINASH GUPTA
ROLL NO. - 120CS0157
DCCN LAB 8

Q2 .

Write a Tcl script that forms a network consisting of 6 nodes, numbered from 1 to 6. Each of source and destination has bandwidth of 300 Mbps and delay of 20 ms. Set the bottleneck link bandwidth as 500 sec and delay 10ms. Set the routing protocol to Droptail. Define different colors for different data flows. Send TCP packet from node 1 to node 4 and UDP packet from node 5 to 6. Start the TCP data transmission at 1 sec and UDP at 15 sec. Finish the transmission at 100 sec. Then run nam to view the
results.



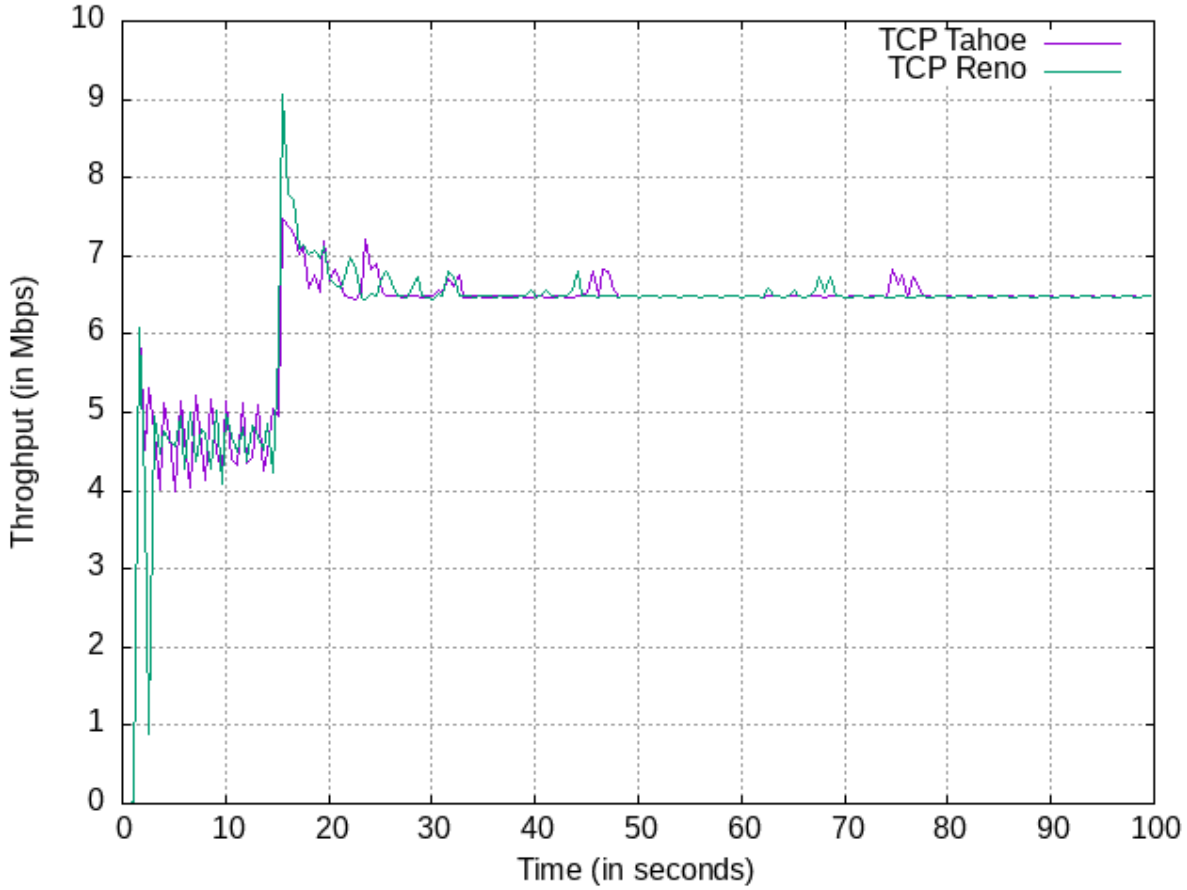
Calculate the following performance metrics using awk script and plot the graph and make the table:

a) Plot Throughput Graph (Tahoe vs Reno)
b) Plot Delay Graph (Tahoe vs Reno)
c) Compare the Packet loss ratio using table (Tahoe vs Reno)
d) Plot Jain Fairness index graph using gnuplot
e) Plot graph for window size (Tahoe vs Reno)

Instantaneous Throughput Calculation

```
#!/usr/bin/awk -f

BEGIN {
    # Set default values for variables
    interval = 0.1  # Time interval in seconds
    time = 0.1
    tcp_byte_count = 0
    udp_byte_count = 0
    tcp_throughput = 0
    udp_throughput = 0
}
```

```
# Process each packet in the trace file
{
   # Check if the packet is TCP or UDP
   if ($5 == "tcp") {
       tcp_byte_count += $6
   } else if ($5 == "cbr") {
       udp_byte_count += $6
   }

   # Update time and calculate throughput every "interval" seconds
   if ($2 >= time) {
       BC = tcp_byte_count+udp_byte_count
       THR = BC * 8 / interval / 1000
       tcp_throughput = tcp_byte_count * 8 / interval / 1000
       udp_throughput = udp_byte_count * 8 / interval / 1000
       printf("%.2f %.2f\n", time, THR)
       time += interval
       tcp_byte_count = 0
       udp_byte_count = 0
   }
}

# Print final throughput if needed
END {
  ; #if (tcp_byte_count > 0 || udp_byte_count > 0) {
    #   tcp_throughput = tcp_byte_count * 8 / interval / 1000000
     #  udp_throughput = udp_byte_count * 8 / interval / 1000000
      # printf("%.2f %.2f %.2f\n", time, tcp_byte_count,
udp_byte_count)
   #}
}
```



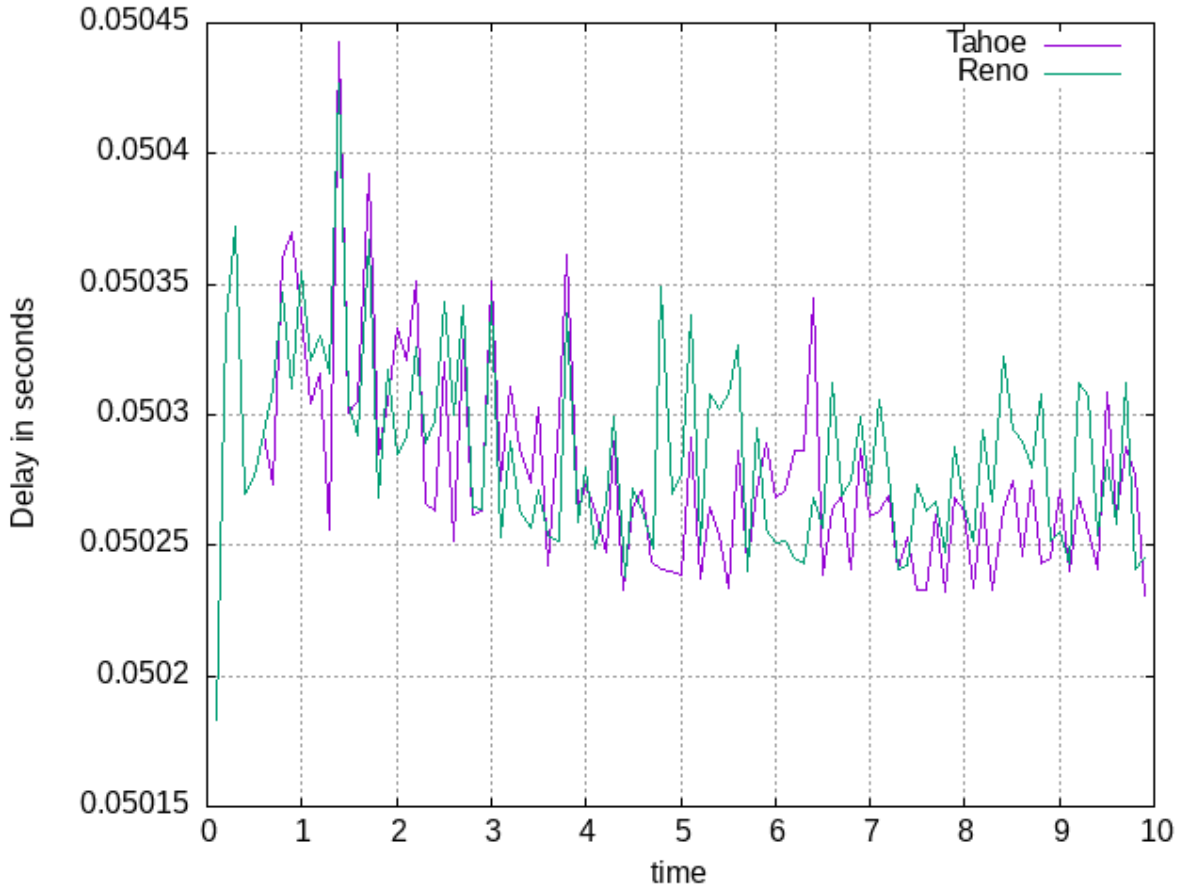
Instantaneous Delay Calculation

```
BEGIN{
    receiveNum = 0;
    interval=0.1;
    start=0.1;

}

{
    event = $1
    time = $2
    from_node = $3;
    to_node = $4;
    pkt_type = $5;
    src_addr = $9;
    dest_addr = $10;
    pkt_id = $12

    if (event == "+" && pkt_type != "ack")
    {
        fro = int(from_node);
```

```
        src = int(src_addr);
        if (fro == src)
        {
            sendTime[pkt_id] = time
        }
    }


    if (event == "r" && pkt_type != "ack")
    {
        to = int(to_node);
        dst = int(dest_addr);
        if (to == dst)
        {
            receiveNum++
            recvTime[pkt_id] = time
            delay[pkt_id] = recvTime[pkt_id] - sendTime[pkt_id];
            tot_delay += delay[pkt_id];
        }
    }
    if($2>=start){
    if (receiveNum != 0)
    {
        avg_delay = tot_delay / receiveNum
    }
    else
    {
        avg_delay = 0
    }
    printf("%.2f %f s\n",time, avg_delay);
    start+=interval;
    avg_delay=0;
    receiveNum=0;
    tot_delay=0;
    for( i in recvTime){
        recvTime[i]=0;
        sendTime[i]=0;
        delay[i]=0;
    }

    }
}
```

```
END{
    ;
}
```

OUTPUT



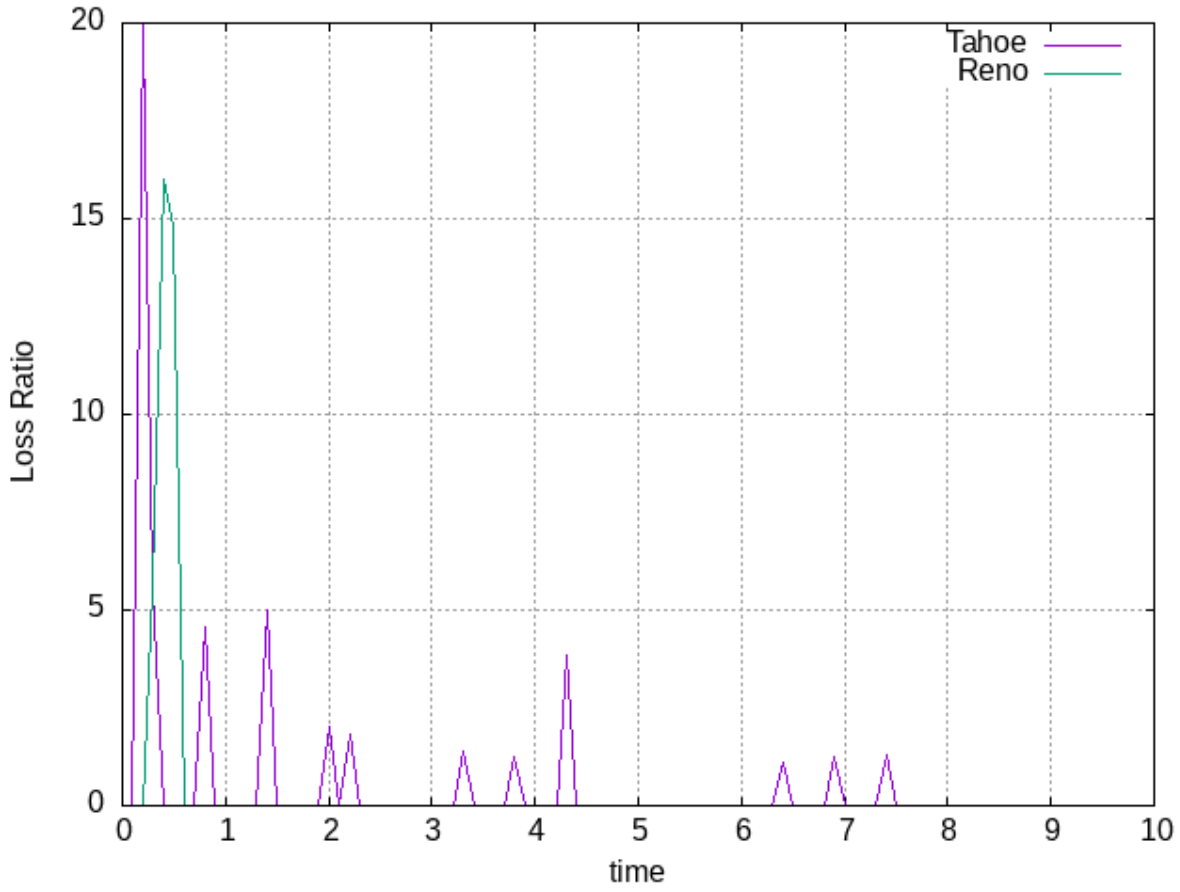
Packet Loss

```
BEGIN {
    send=0;
    received=0;
    dropped=0;
    interval=0.1;
    start=0.1;
}

{
# Trace line format: normal
    event = $1;
    time = $2;
```

```
    from_node = $3;
    to_node = $4;
    pkt_type = $5;
    pkt_size = $6;
    flgs = $7;
    f_id = $8;
    src_addr = $9;
    dest_addr = $10;
    seq_no = $11;
    pkt_id = $12;
#packet delivery ratio
    if (event == "+" && pkt_type != "ack")
    {
        fro = int(from_node);
        src = int(src_addr);
        if (fro == src)
        {
            send++
        }
    }

    if (event == "r" && pkt_type != "ack")
    {
        to = int(to_node);
        dst = int(dest_addr);
        if (to == dst)
        {
            received++
        }
    }

    if (event == "d")
    {
        dropped++;
    }

    if($2>=start){
        printf("%.2f %f\n",time , (dropped/send)*100);
        start+=interval;
        send=0;
        received=0;
        dropped=0;
    }
```

```
}
 END{ ;
    # print "\nGeneratedPackets = " send;
    # print "ReceivedPackets = " received;
    # print "Total Dropped Packets = " dropped;
    # print ("\nPacket Delivery Ratio = ", (received/send)*100" %");
    # print ("Packet Loss Ratio = ", (dropped/send)*100" %");
}
```

OUTPUT-



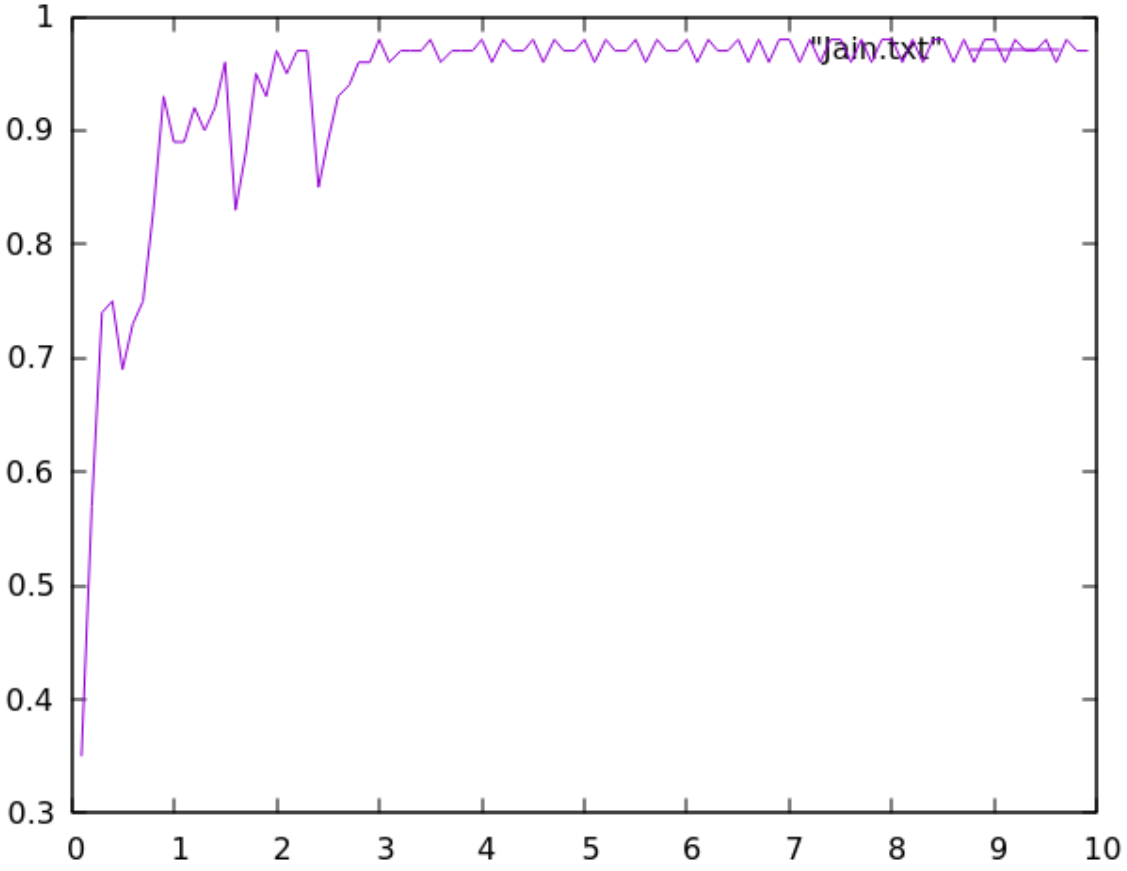
Jain Fairness index graph using gnuplot

```
BEGIN{
    interval = 0.1
    start = 0.1
    sum = 0;
    Sq_sum = 0;
    cnt = 0;
    JI = 0;
}
```

```
{
   event = $1
   time = $2
   from_node = $3;
   to_node = $4;
   pkt_type = $5;
   pkt_size = $6;
   f_id = $8;
   src_addr = $9;
   dest_addr = $10;
   pkt_id = $12

   if (event == "r" && pkt_type != "ack")
   {
       to = int(to_node);
       dst = int(dest_addr);
       if (to == dst)
       {
           node_thr[f_id] += pkt_size;
       }
   }

   if (time >= start)
   {
       for (i in node_thr)
       {
           Th = (node_thr[i]/interval)*(8/1000);
           sum += Th;
           Sq_sum += (Th*Th);
           cnt++;
           node_thr[i] = 0;
       }
       JI = ((sum*sum)/(cnt*Sq_sum));
       printf("%.2f %.2f\n",time, JI);
       start += interval
       sum = 0;
       Sq_sum = 0;
       cnt = 0;
   }
}

END{
   ;
```

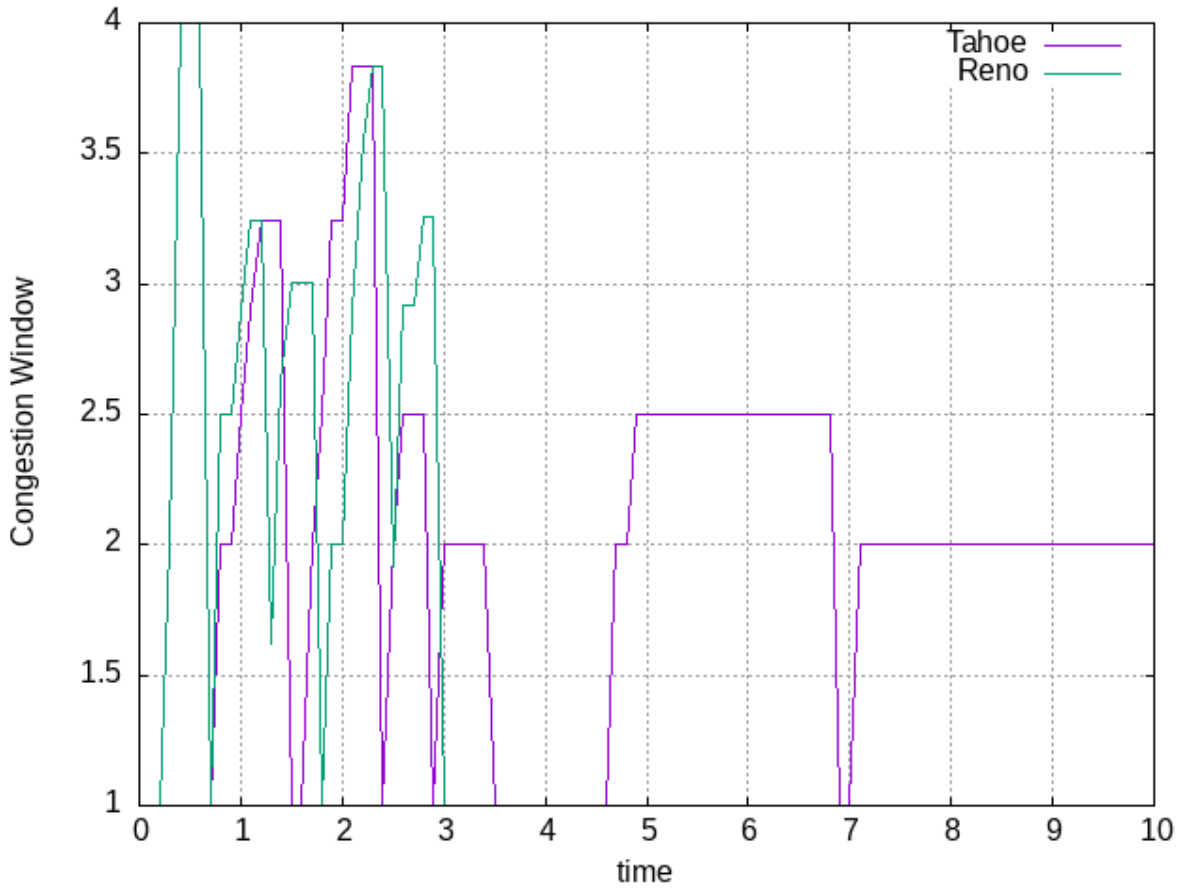
```
}
```

OUTPUT -



Congestion Window Size

```
proc Record {} {
   global f_cwnd tcp1 tcp2 tcp3 tcp4 ns
   set intval 0.1
   set now [$ns now]
   set cwnd1 [$tcp1 set cwnd_]
   set cwnd2 [$tcp2 set cwnd_]
   set cwnd3 [$tcp3 set cwnd_]
   set cwnd4 [$tcp4 set cwnd_]



   puts $f_cwnd "$now $cwnd1  $cwnd2  $cwnd3 $cwnd4"
   $ns at [expr $now + $intval] "Record"
}
```


Tahoe
Reno
Congestion Window
time
4
3.5
3
2.5
2
1.5
1
0
1
2
3
4
5
6
7
8
9
10