REPORT ON NS3 TERM PROJECT

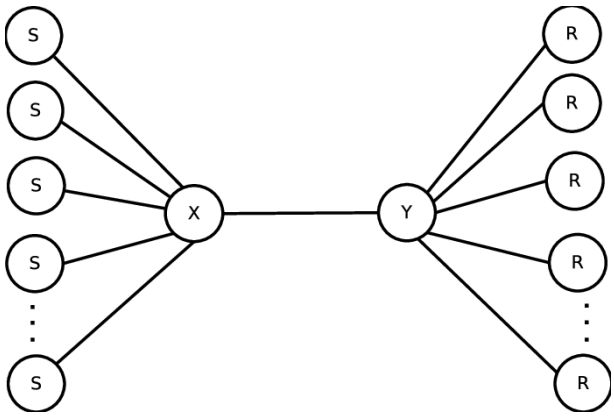
TCP Congestion Control Scheme Suited for Bandwidth Reservation Network

Submitted By:

Student ID: 1705031.
Name: Md.Mahedi Hasan Rigan
Student ID: 23 February, 2022

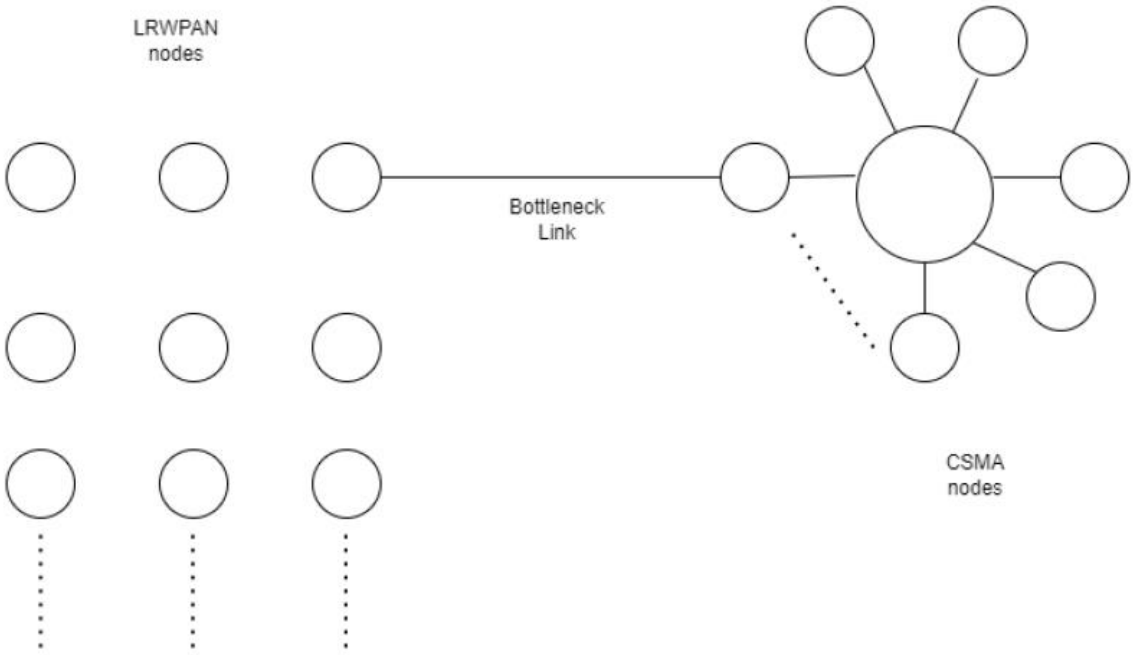
NETWORK TOPOLOGIES

Wired:



The dumbbell topology consists of a number of TCP senders (S), TCP receivers (R), routers (X and Y) and several links. The link in the middle work as bottleneck link as it's bandwidth is set to low. We can vary the number of nodes on each side.

Wireless:



In this topology, right sided nodes are CSMA nodes and left sided nodes are LRWPAN nodes. They are connected by a point-to-point network which is the bottleneck link here. Here, client applications are set-up at wireless nodes and the CSMA nodes contain packet sink applications. Here, as the wireless nodes are static in place, as mobility module, “ConstantPositionMobilityModel” is used. In the network layer, IPv6 is used instead of IPv4.

Parameters under variation:

Here, the parameters which are varied :

1. **Number of Nodes:** In both wired and wireless topologies, the number of nodes on each side of the bottleneck link can be varied with the “numberOfNodes” parameter. For example: if numberOfNodes= 10 , there are total 22 nodes, 20 on each side and 2 in the middle.

2. **Number of Flows:** In both wired and wireless topologies, the number of flows can be varied with the “numberOfFlows” parameter. Each time a flow can pass from sender to receiver.So, there can be maximum of 2*numberOfFlows in the network.

3. **Packet per Second:** We can also vary the data-rate of applications by varying the “packetsPerSecond” parameter. For example: if we set packetsPerSecond = 1000, then the dataRate will become = 1000*packet_size and which will be in bps unit.We can the covert it to “Mbps” by dividing by 1024 twice.

4. **Coverage Area:** In the wireless topology, as the nodes are static, we can configure the range or coverage area of the WiFi channel. We can vary this with help of “coverageArea” parameter. For example: if coverageArea=2, the coverage area will be doubled of the initially set coverage area.

Overview of the proposed algorithm:

We propose the idea of setting the congestion window and the slow start threshold by using the reserved bandwidth once congestion occurs. One goal of the proposed system is to minimize variation in the congestion window.

Algorithm:

1. When a TCP connection is established, the sender initializes the congestion window to the size of the maximum segment.
2. When packet loss is detected, the congestion window is set to the slow start threshold, and the algorithm moves into the congestion avoidance mode (the congestion window size is increased by one packet).
3. When the congestion window size exceeds the slow start threshold, the algorithm also moves into the congestion avoidance mode (the congestion window size is increased by one packet). If a timeout occurs, the congestion window is reset to one minimum segment.
4. The slow start threshold is set to a window size that suits bandwidth reservation, and how to decide the slow start threshold is outlined below.

The equation of finding the value of “ssthresh”-

1. After receipt of 3 duplicate acknowledgments: Set slow start threshold, *ssthresh*, as follows-

$$ssthresh = \frac{r}{size \times 8} \times RTT$$

$$cwnd = ssthresh.$$

2. After timeout: Set ssthresh as follows-

$$ssthresh = \frac{r}{size \times 8} \times RTT$$

$$cwnd = 1.$$

Modifications made in NS3:

Firstly we created 2 new file in “internet” model of ns3.They are

i) Tcp-br.h
ii) Tcp-br.cc

So, we need to write this file name in wscript file.Tcp-br.h inherited from tcp-new reno.In the “tcp-br.h” file we have to first declare all the virtual method which is in “tcp-congestion-ops.h” as tcp-new-reno is also inherited from this file.As our algorithm needs to update the value of ssthresh, we do it in “tcp-br.cc” file.Then to update the value while in timeout and 3 duplicate acknowledge we then look for GetThresh() method in “tcp-socket-base.cc” file and update the value of “m_ssthresh” and “m_cwnd” variable.

The corresponding files in our simulation are given below-

```cpp
class TcpBR : public TcpNewReno
{
public:
    //
    static TypeId GetTypeId (void);

    //
    TcpBR (void);

    // Init (tcpnv_init)
    TcpBR (const TcpBR& sock);

    //
    virtual ~TcpBR (void);

    //
    virtual std::string GetName () const;

    virtual void IncreaseWindow (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked);


    virtual uint32_t GetSsThresh (Ptr<const TcpSocketState> tcb, uint32_t bytesInFlight);

    virtual Ptr<TcpCongestionOps> Fork ();
protected:
    virtual uint32_t SlowStart (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked);
    virtual void CongestionAvoidance (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked);
private:
    // TcpBR parameters
    uint32_t m_Reserved_Band;

};
} // namespace ns3
#endif // TcpBR_H
```

Filename: tcp-br.h

```
uint32_t
TcpBR::GetSsThresh (Ptr<const TcpSocketState> state,
                    uint32_t bytesInFlight)
{
  NS_LOG_FUNCTION (this << state << bytesInFlight);
  Time tmp = state->m_lastRtt;
  uint32_t RTT = static_cast<uint32_t>(tmp.GetSeconds());
  uint32_t ans = RTT*m_Reserved_Band;
  ans = ans/(1.2*1024);
  ans = ans/8;
  return ans;


  // return std::max (2 * state->m_segmentSize, bytesInFlight / 2);
}
```

Fig: GetSsThresh method(inside tcp-br.cc)

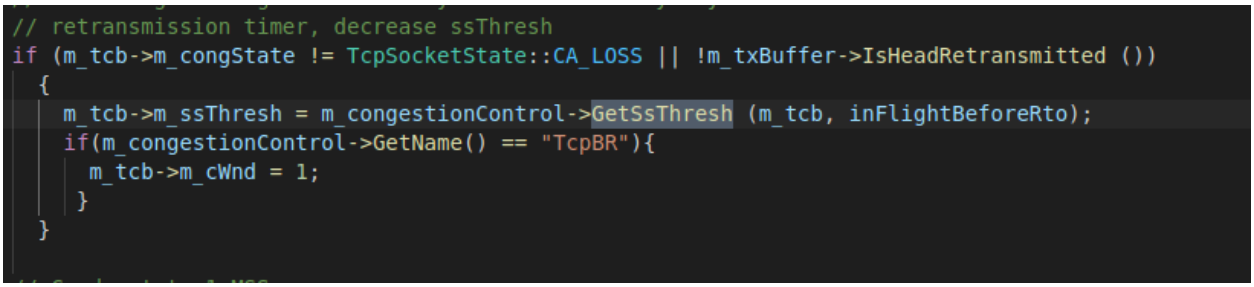
```
//     (such as datacenter network) whose sending rate is constrainted by
//     TCP socket buffer size at receiver side.
if ((m_dupAckCount == m_retxThresh) && ((m_highRxAckMark >= m_recover) || (!m_recoverActive)))
  {
    if(m_congestionControl->GetName() == "TcpBR"){
      m_tcb->m_ssThresh = m_congestionControl->GetSsThresh (m_tcb, BytesInFlight());
      m_tcb->m_cWnd = m_tcb->m_ssThresh;
    }
    EnterRecovery (currentDelivered);
    NS_ASSERT (m_tcb->m_congState == TcpSocketState::CA_RECOVERY);

  }
```

Fig: DupAck() method (inside “tcp-socket-base.cc)

```
// Sender should reduce the Congestion Window as a response to receiver's
// ECN Echo notification only once per window
void
TcpSocketBase::EnterCwr (uint32_t currentDelivered)
{
  NS_LOG_FUNCTION (this << currentDelivered);
  m_tcb->m_ssThresh = m_congestionControl->GetSsThresh (m_tcb, BytesInFlight ());
  if(m_congestionControl->GetName() == "TcpBR"){
    m_tcb->m_cWnd = m_tcb->m_ssThresh;
  }
  NS_LOG_DEBUG ("Reduce ssThresh to " << m_tcb->m_ssThresh);
```

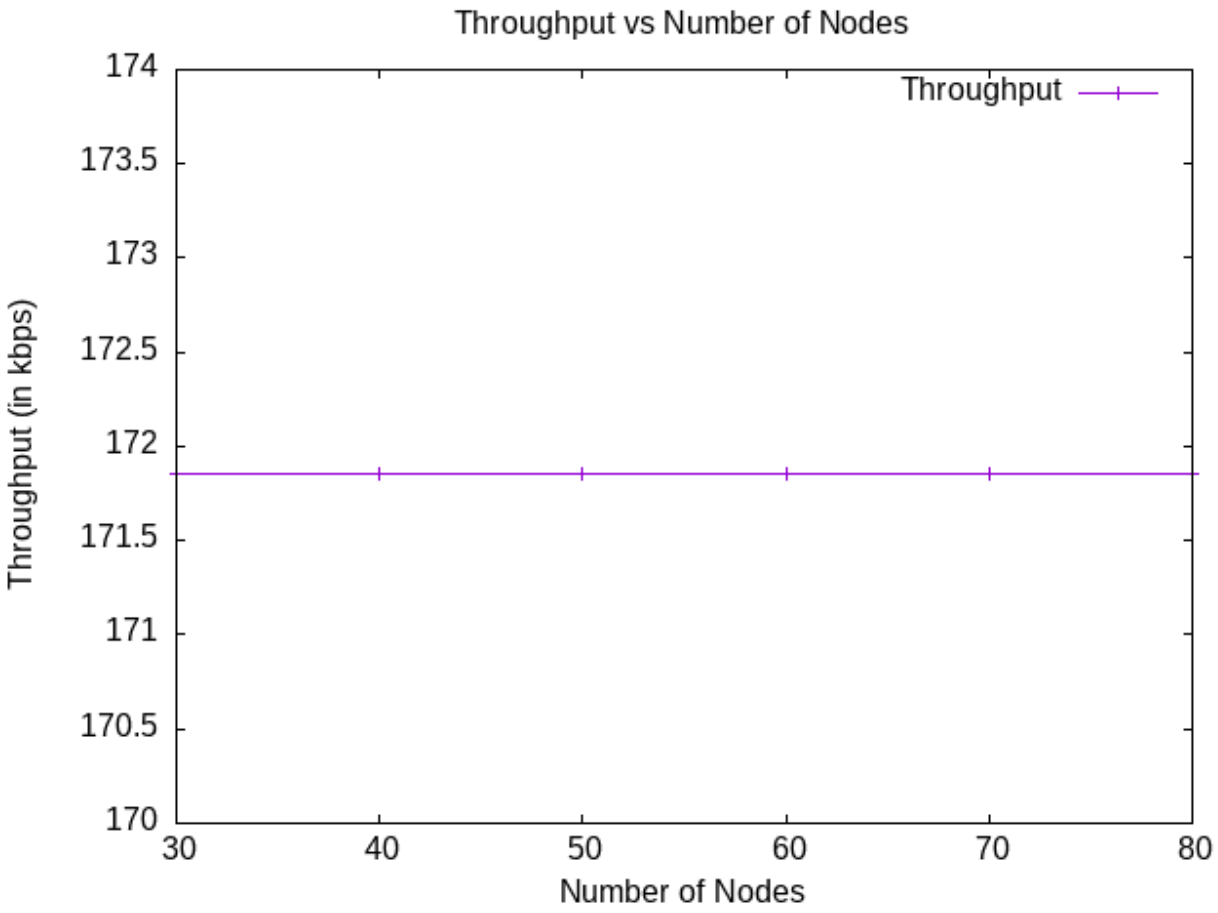
Fig: EnterCwr() method (inside “tcp-socket-base.cc)

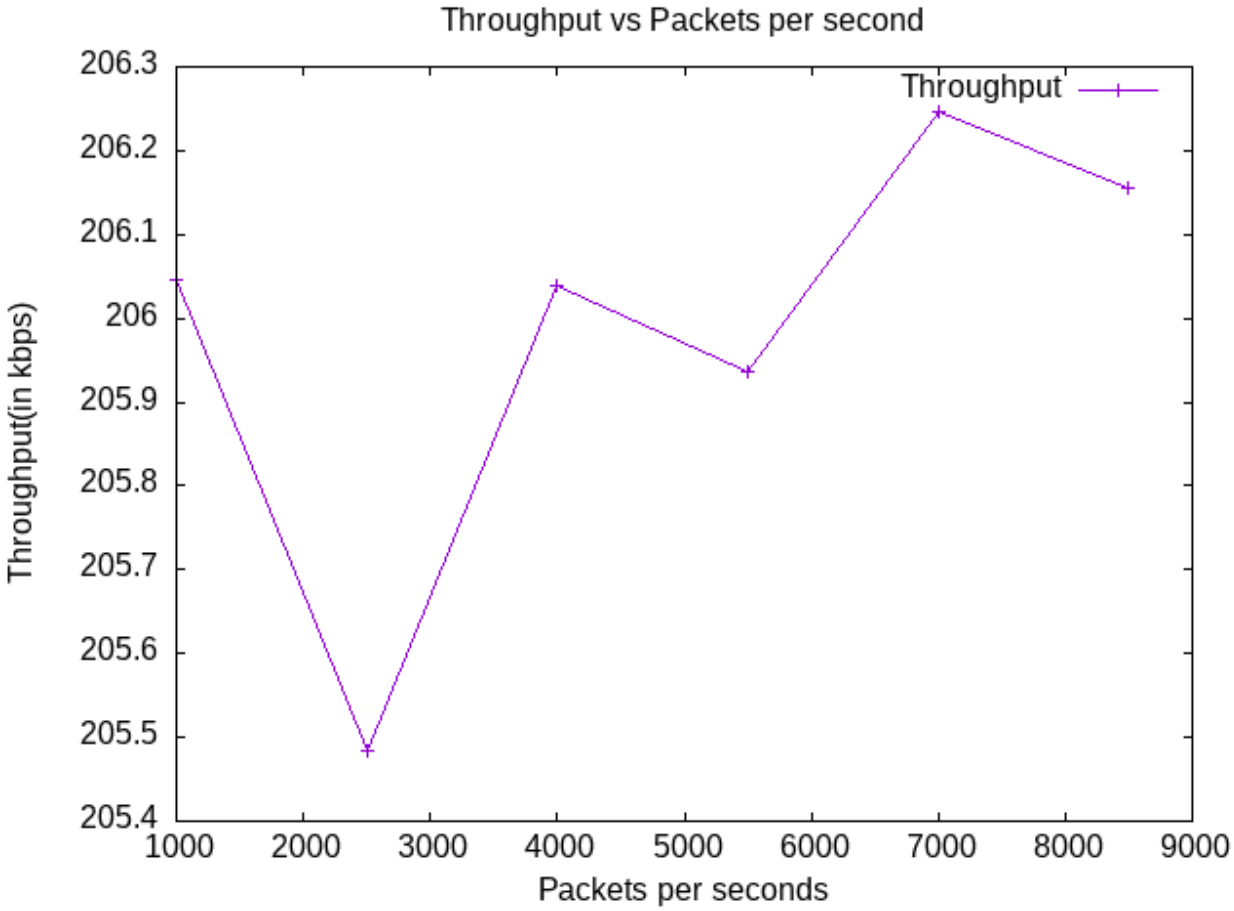


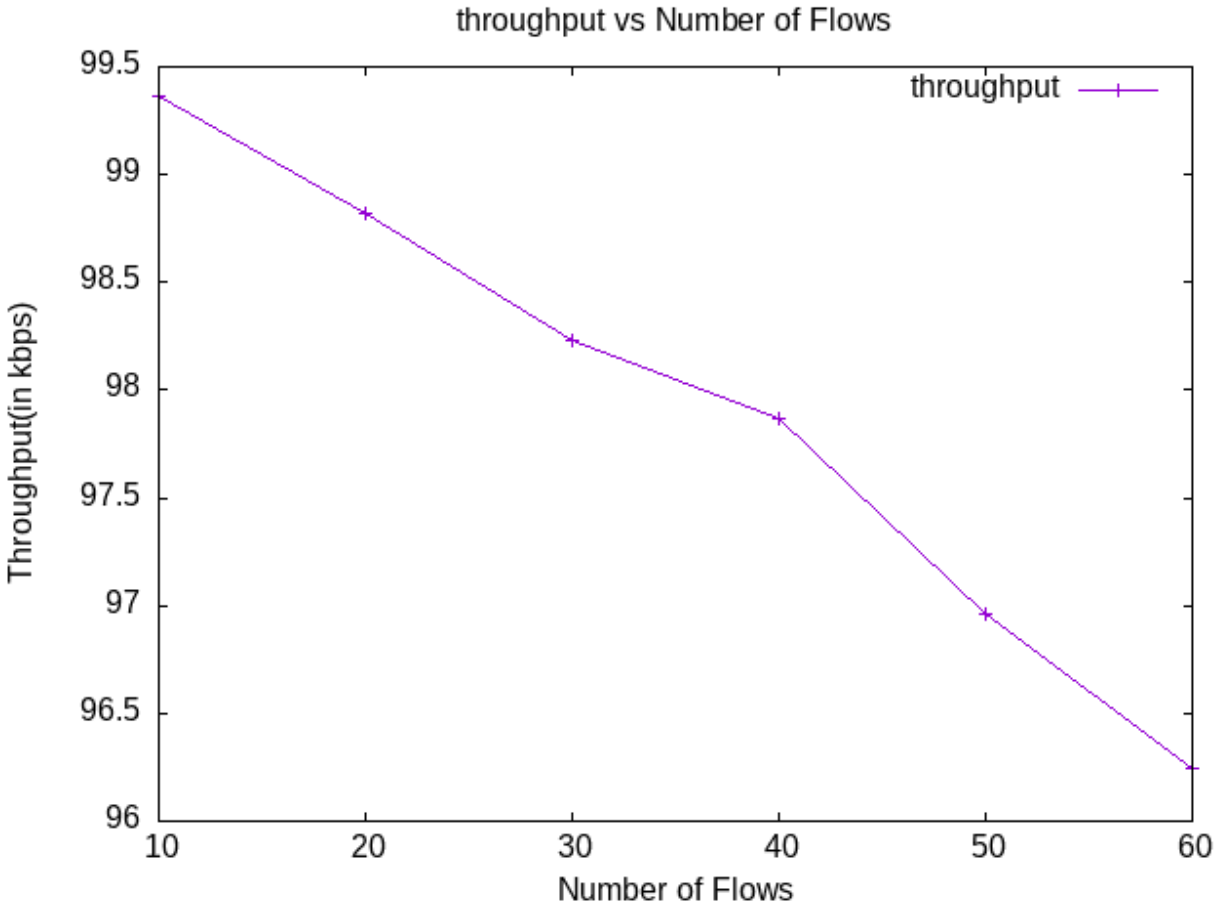
```
// retransmission timer, decrease ssThresh
if (m_tcb->m_congState != TcpSocketState::CA_LOSS || !m_txBuffer->IsHeadRetransmitted ())
  {
    m_tcb->m_ssThresh = m_congestionControl->GetSsThresh (m_tcb, inFlightBeforeRto);
    if(m_congestionControl->GetName() == "TcpBR"){
      m_tcb->m_cWnd = 1;
     }
  }
```

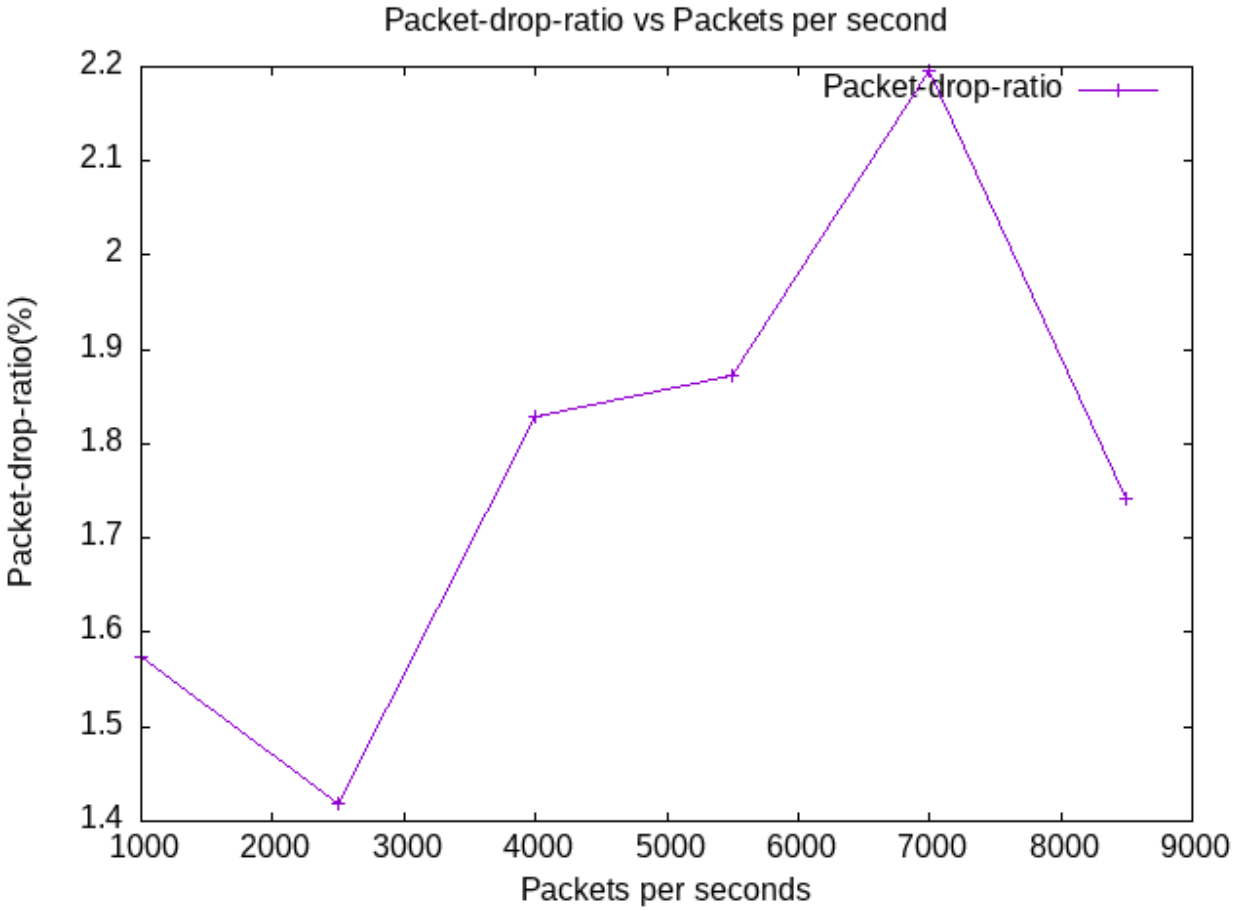
Fig: ReTxTimeout () method (inside “tcp-socket-base.cc)

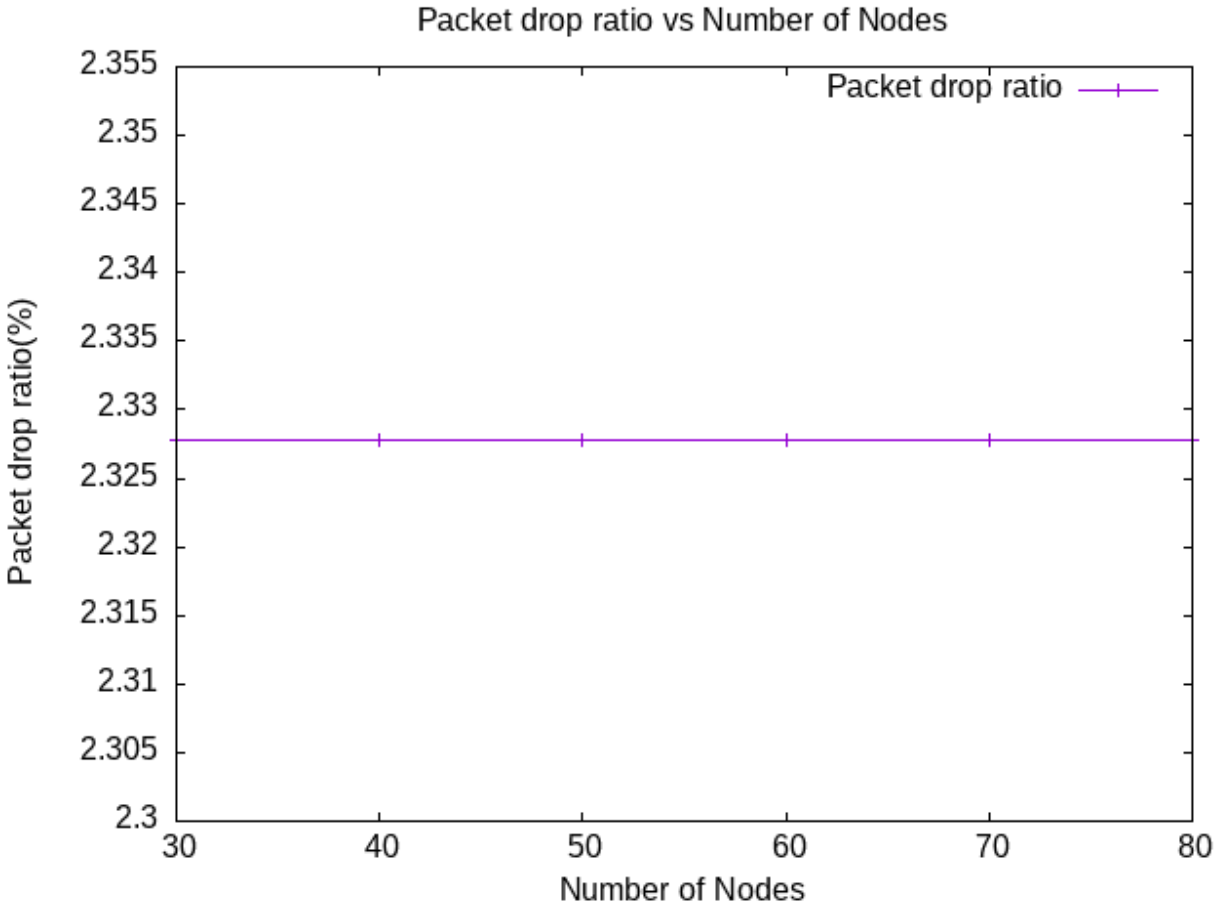
Results with Graphs (TASK A):
Wired:

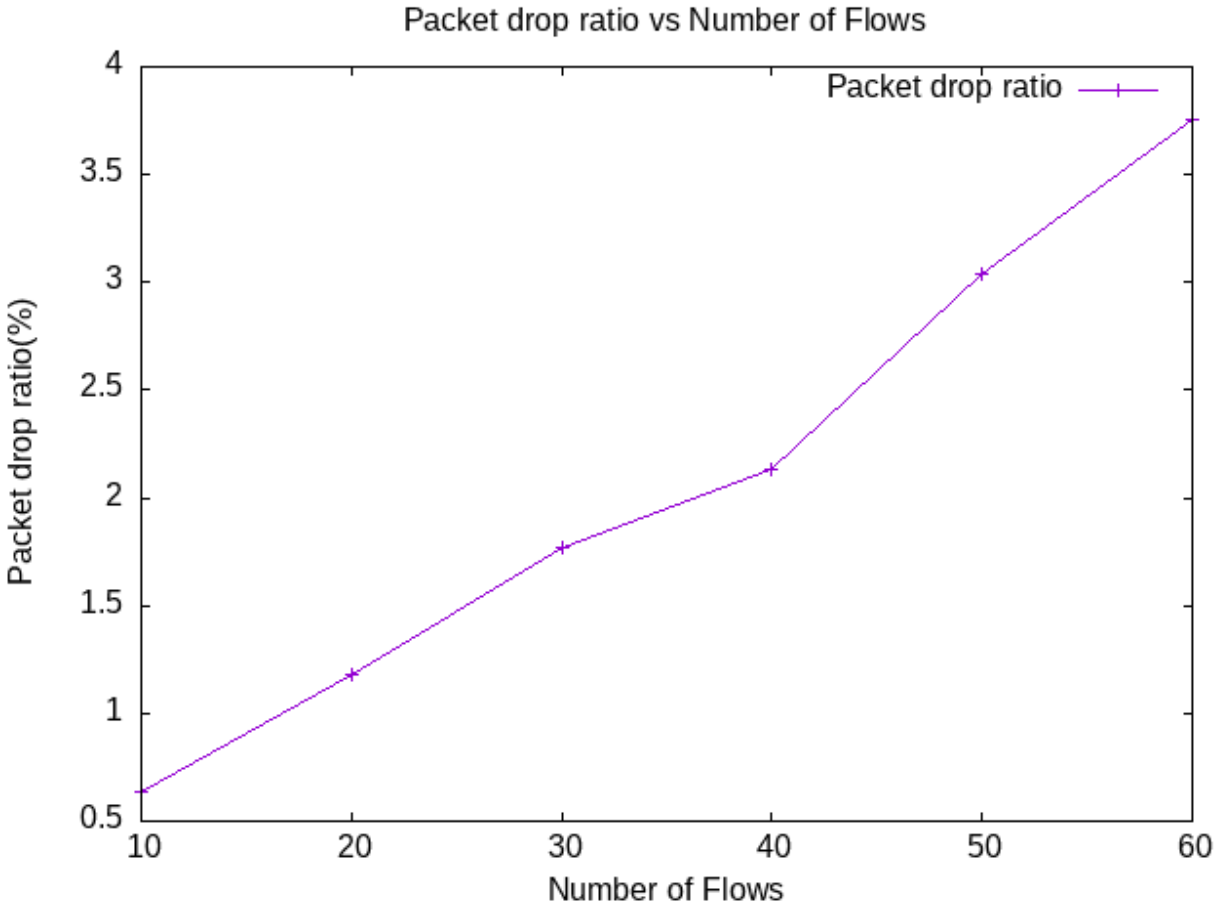


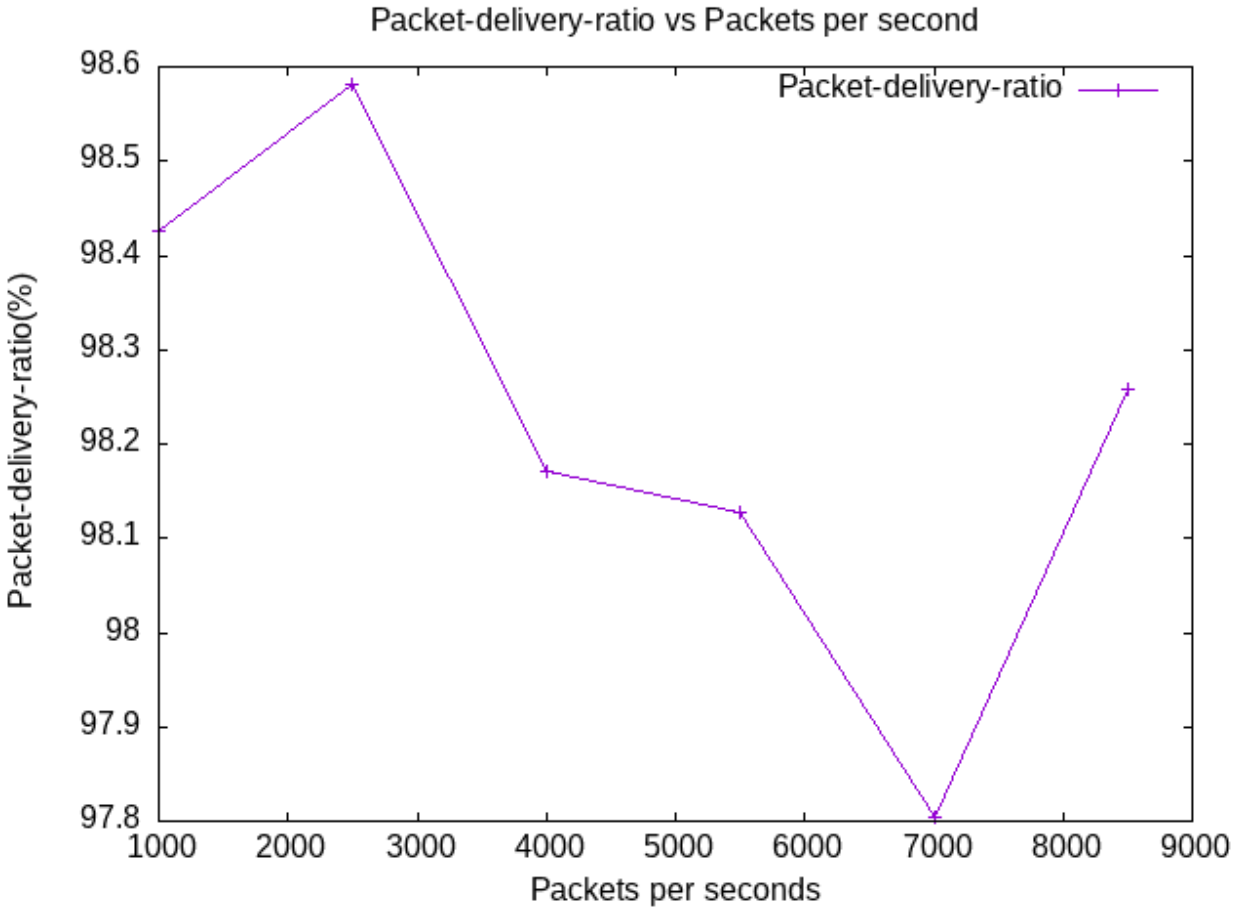

Throughput vs Packets per second
Throughput
Throughput(in kbps)
206.3
206.2
206.1
206
205.9
205.8
205.7
205.6
205.5
205.4
1000
2000
3000
4000
5000
6000
7000
8000
9000
Packets per seconds

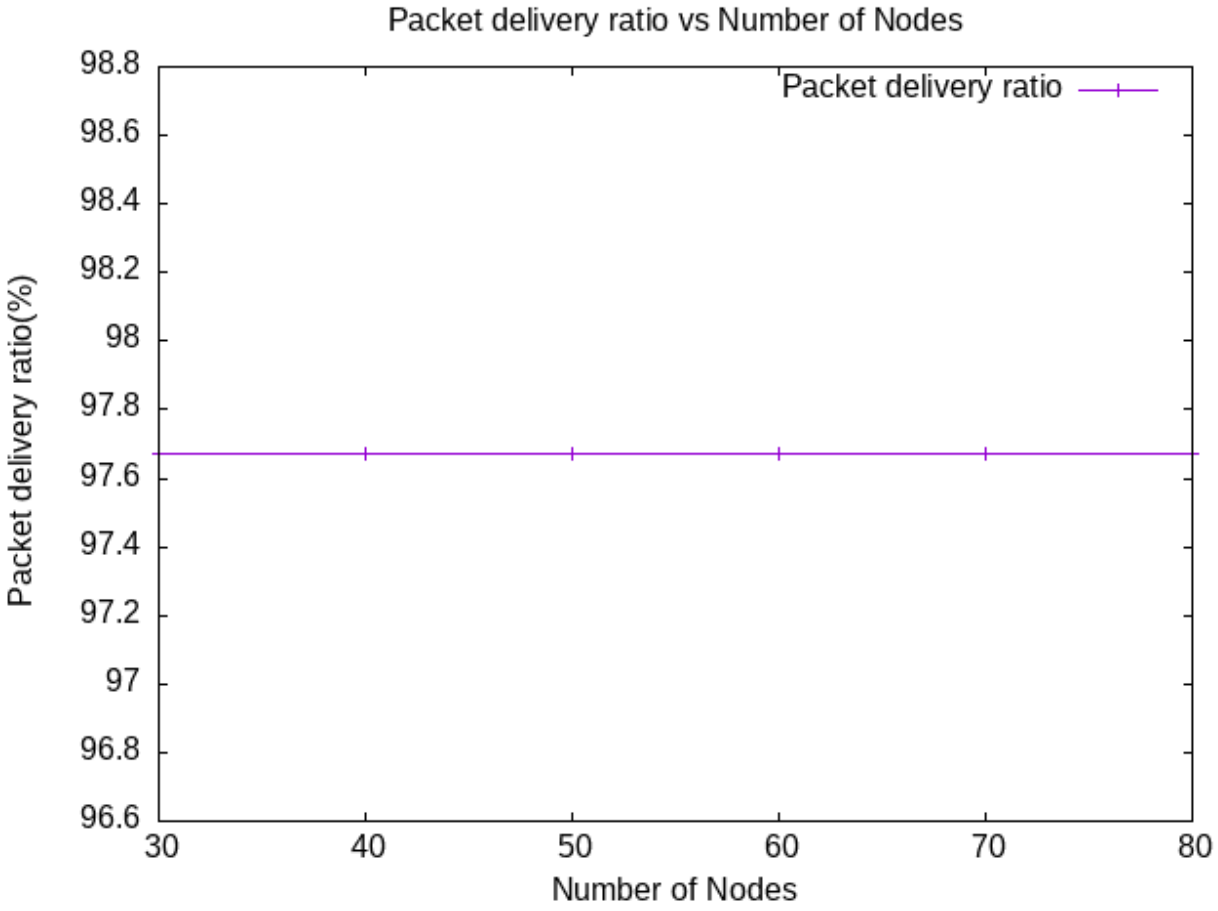

throughput vs Number of Flows
throughput
Throughput(in kbps)
Number of Flows
99.5
99
98.5
98
97.5
97
96.5
96
10
20
30
40
50
60

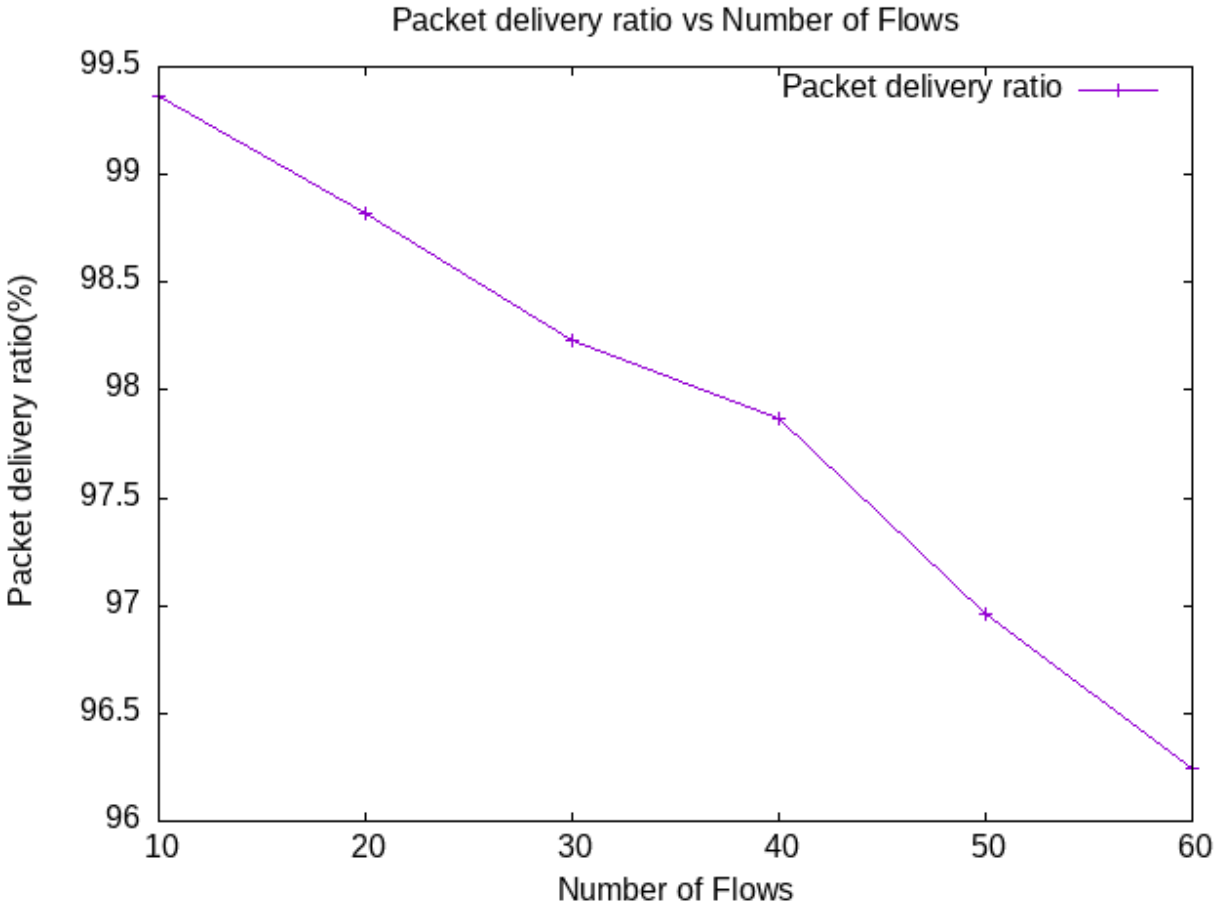

Packet-drop-ratio vs Packets per second
Packet-drop-ratio
Packet-drop-ratio(%)
2.2
2.1
2
1.9
1.8
1.7
1.6
1.5
1.4
1000
2000
3000
4000
5000
6000
7000
8000
9000
Packets per seconds

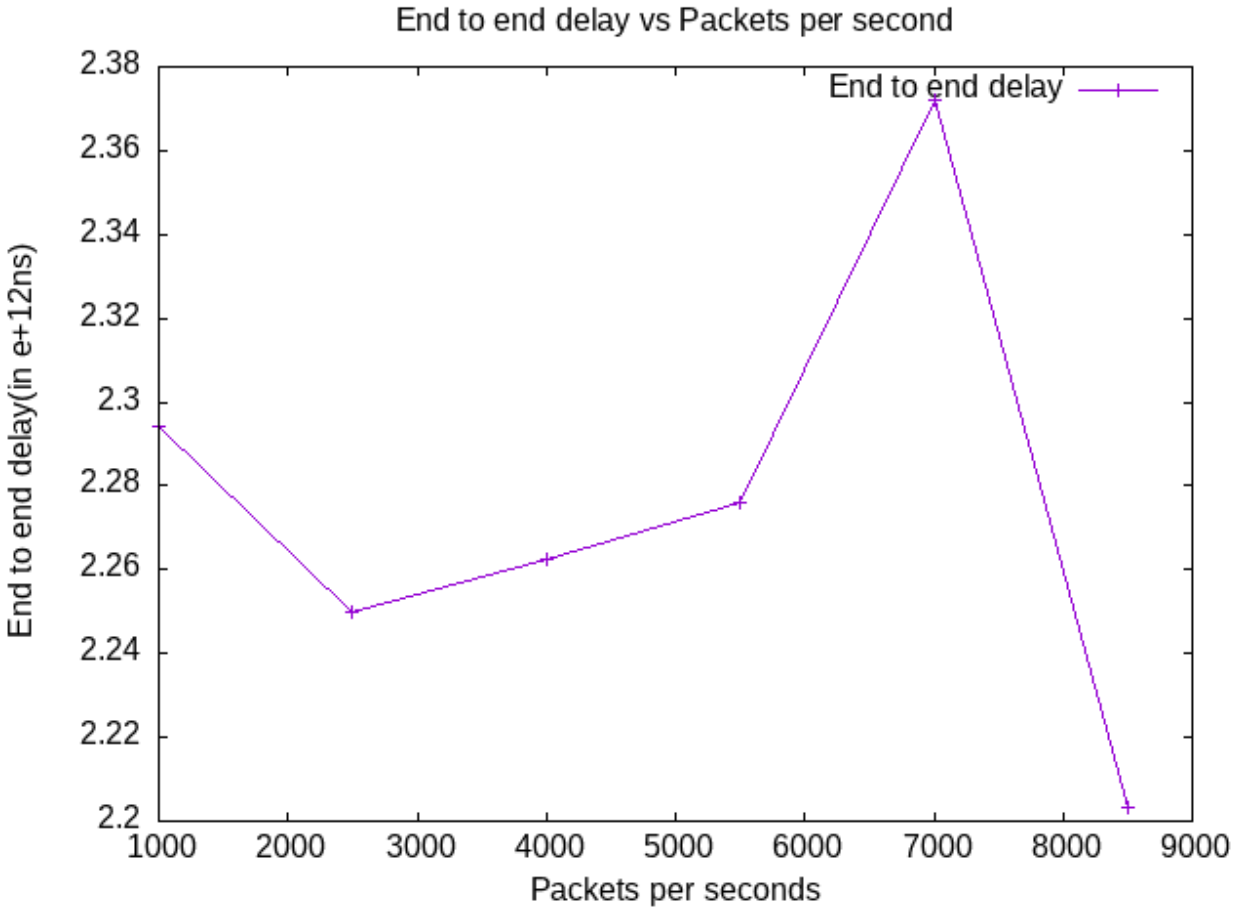

Packet drop ratio vs Number of Nodes
Packet drop ratio
Packet drop ratio(%)
2.355
2.35
2.345
2.34
2.335
2.33
2.325
2.32
2.315
2.31
2.305
2.3
30
40
50
60
70
80
Number of Nodes

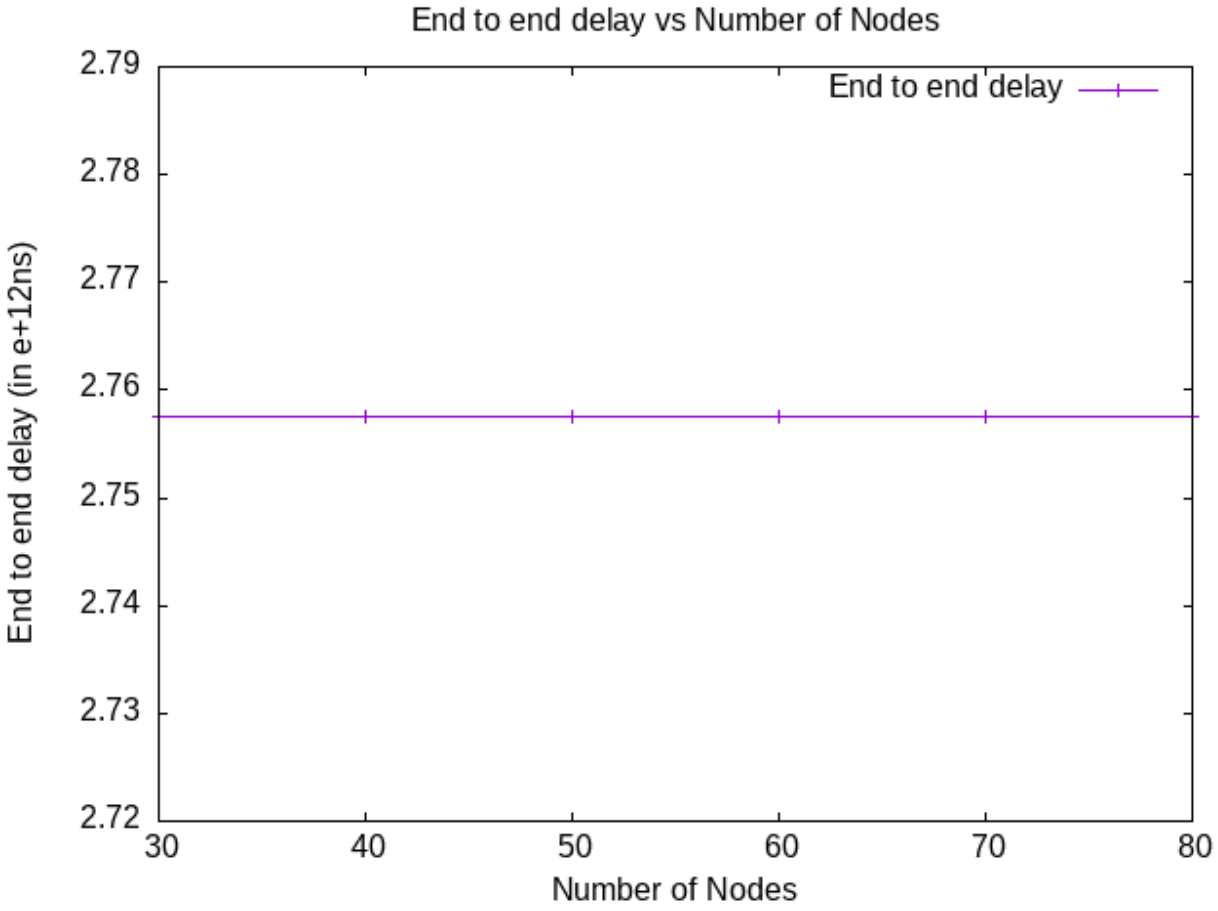

Packet drop ratio vs Number of Flows
Packet drop ratio
Packet drop ratio(%)
Number of Flows
4
3.5
3
2.5
2
1.5
1
0.5
10
20
30
40
50
60

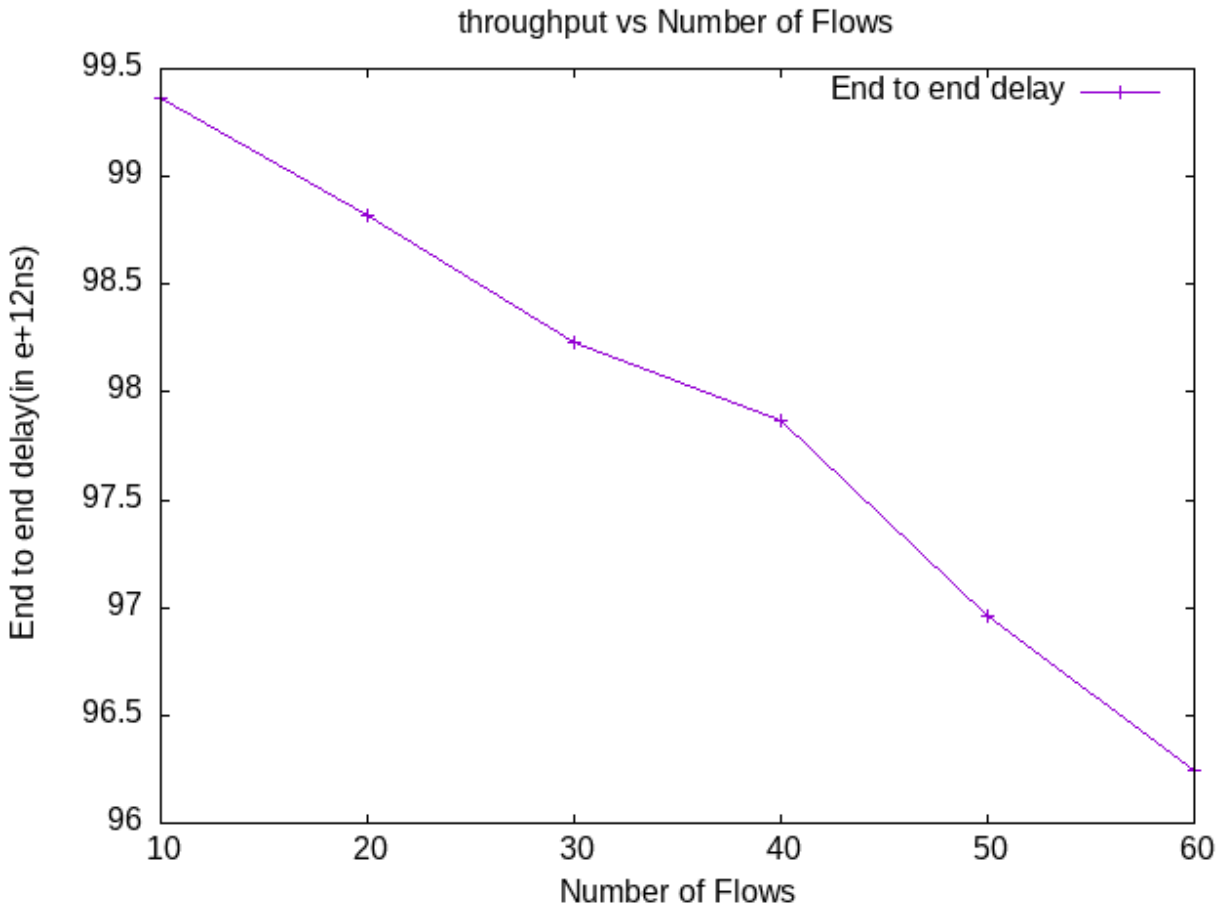

Packet-delivery-ratio vs Packets per second
Packet-delivery-ratio
Packet-delivery-ratio(%)
98.6
98.5
98.4
98.3
98.2
98.1
98
97.9
97.8
1000
2000
3000
4000
5000
6000
7000
8000
9000
Packets per seconds


Packet delivery ratio vs Number of Nodes
Packet delivery ratio
Packet delivery ratio(%)
98.8
98.6
98.4
98.2
98
97.8
97.6
97.4
97.2
97
96.8
96.6
30
40
50
60
70
80
Number of Nodes

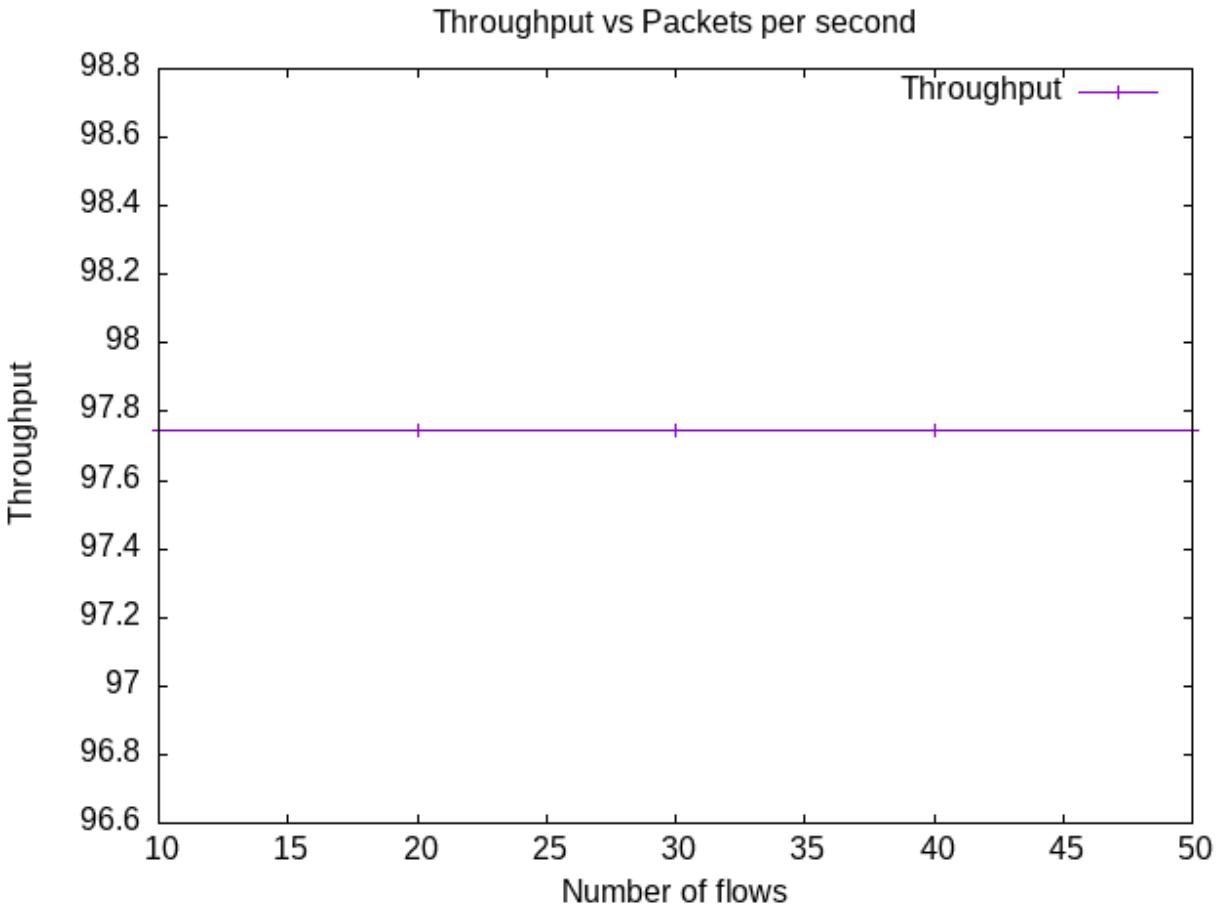

Packet delivery ratio vs Number of Flows
Packet delivery ratio
Packet delivery ratio(%)
99.5
99
98.5
98
97.5
97
96.5
96
10
20
30
40
50
60
Number of Flows

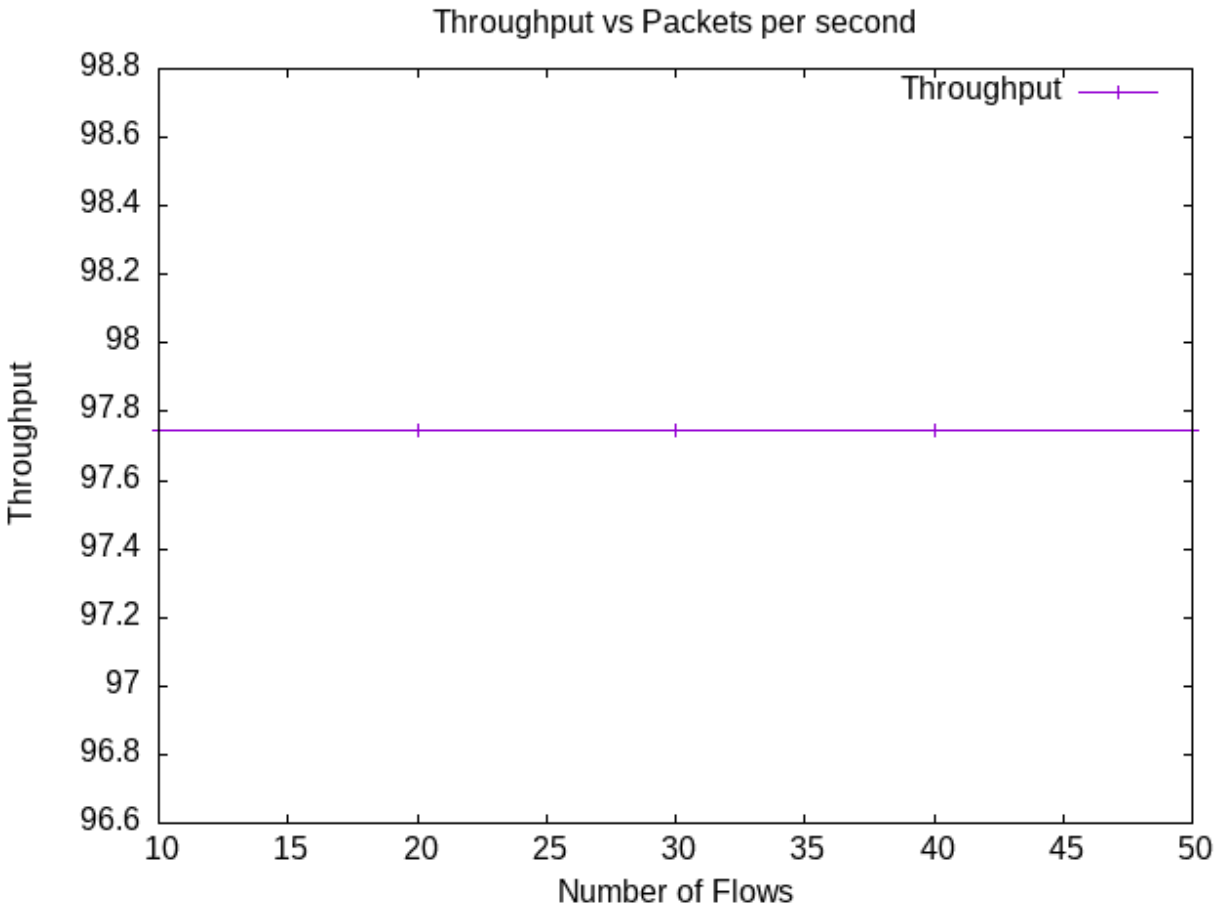

End to end delay vs Packets per second
End to end delay
End to end delay(in e+12ns)
Packets per seconds
2.38
2.36
2.34
2.32
2.3
2.28
2.26
2.24
2.22
2.2
1000
2000
3000
4000
5000
6000
7000
8000
9000

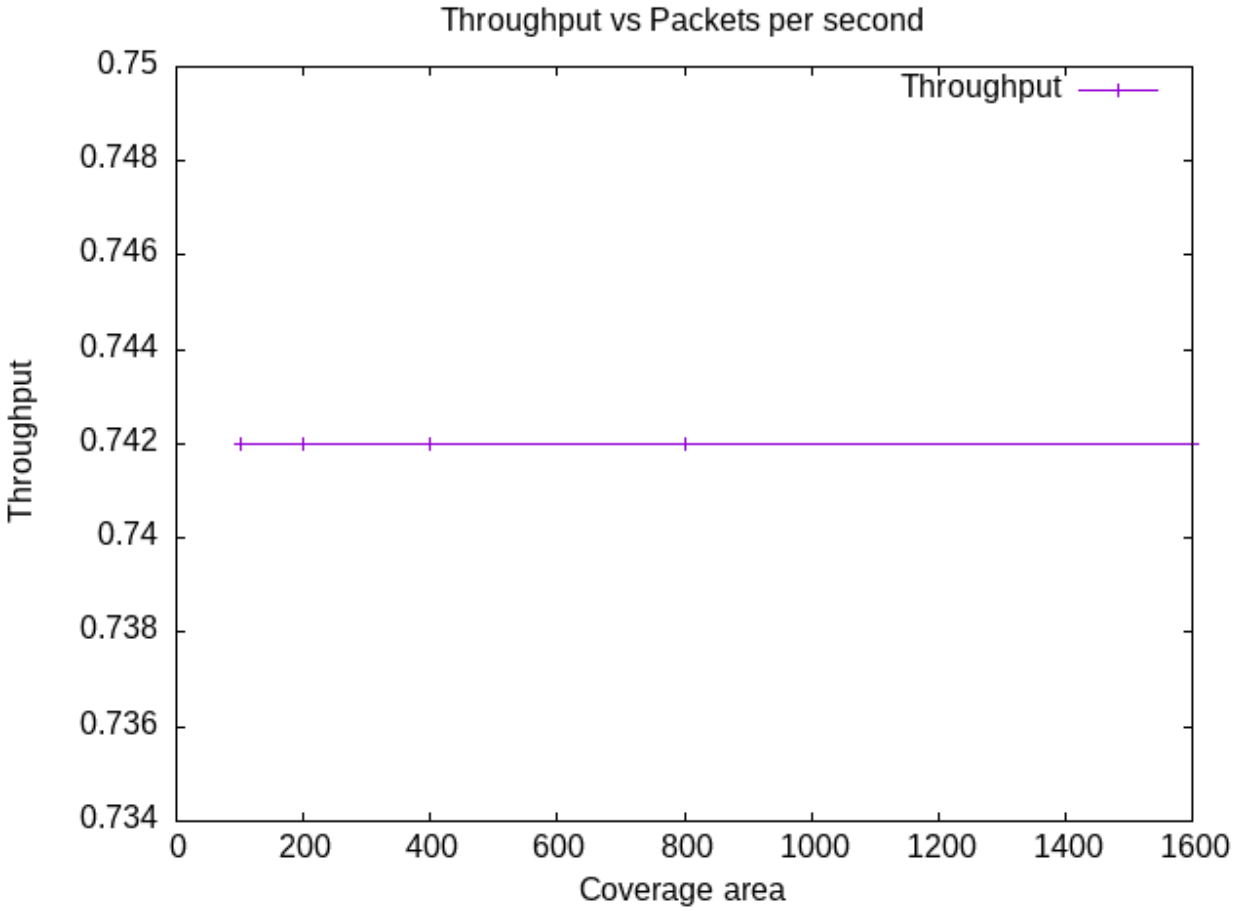

End to end delay vs Number of Nodes
End to end delay
End to end delay (in e+12ns)
2.79
2.78
2.77
2.76
2.75
2.74
2.73
2.72
30
40
50
60
70
80
Number of Nodes

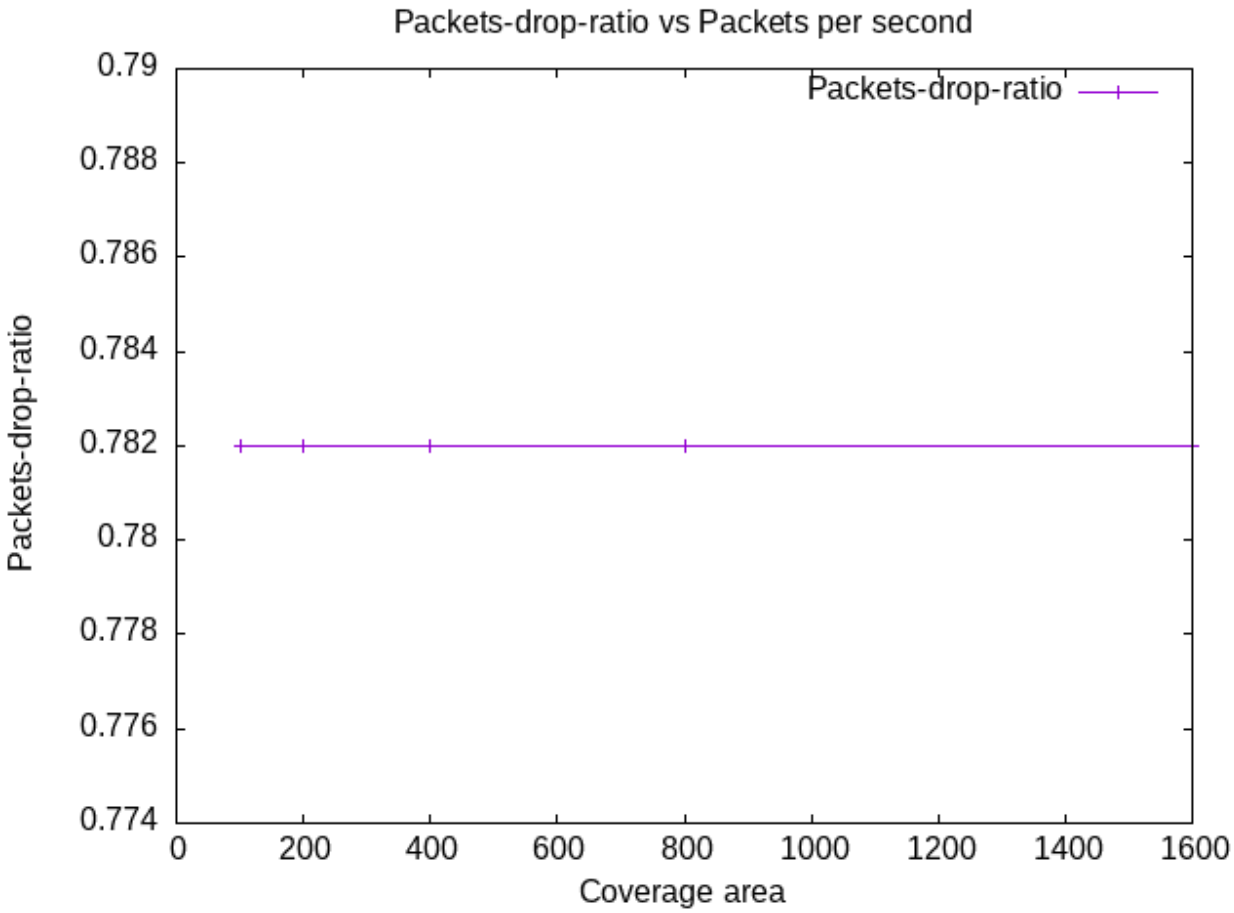

throughput vs Number of Flows
End to end delay
End to end delay(in e+12ns)
99.5
99
98.5
98
97.5
97
96.5
96
10
20
30
40
50
60
Number of Flows

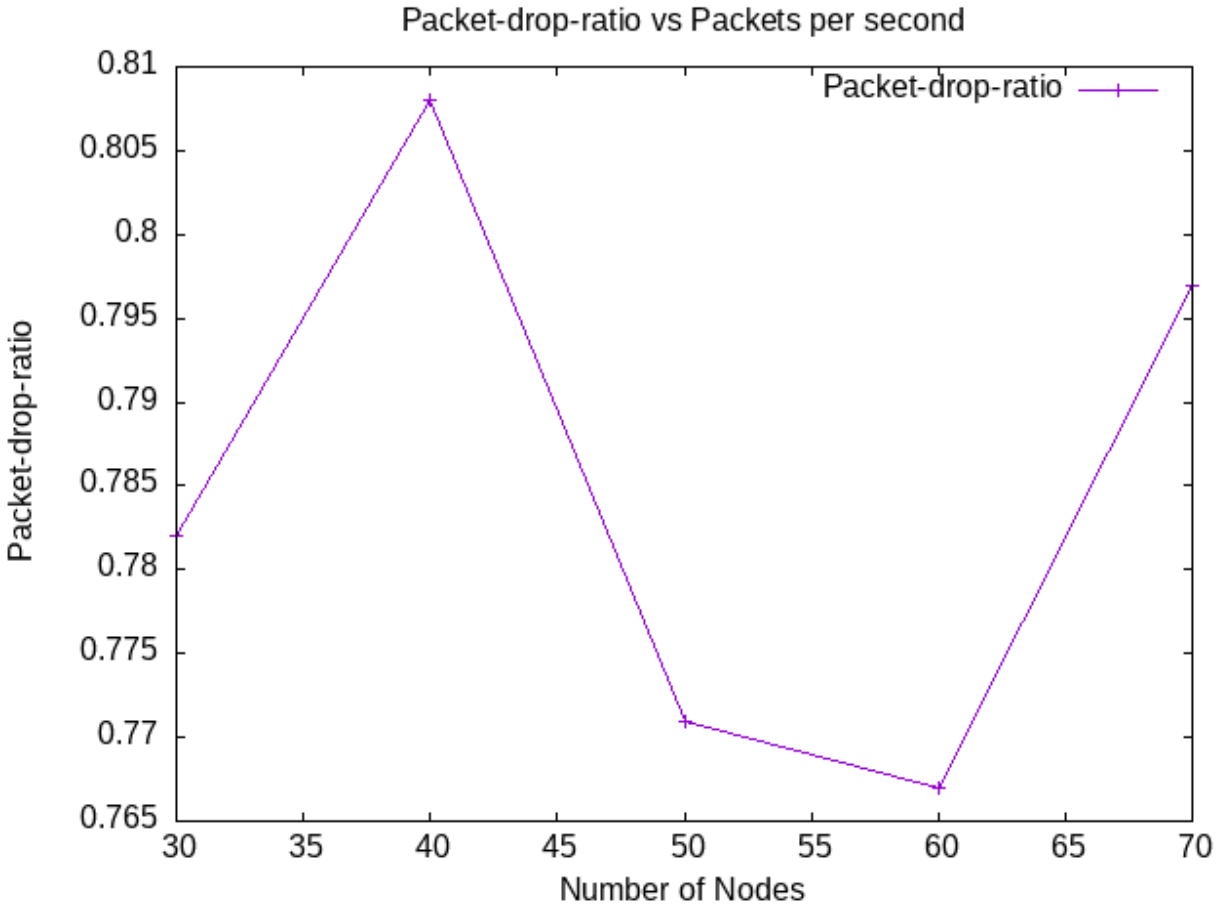
Wireless:

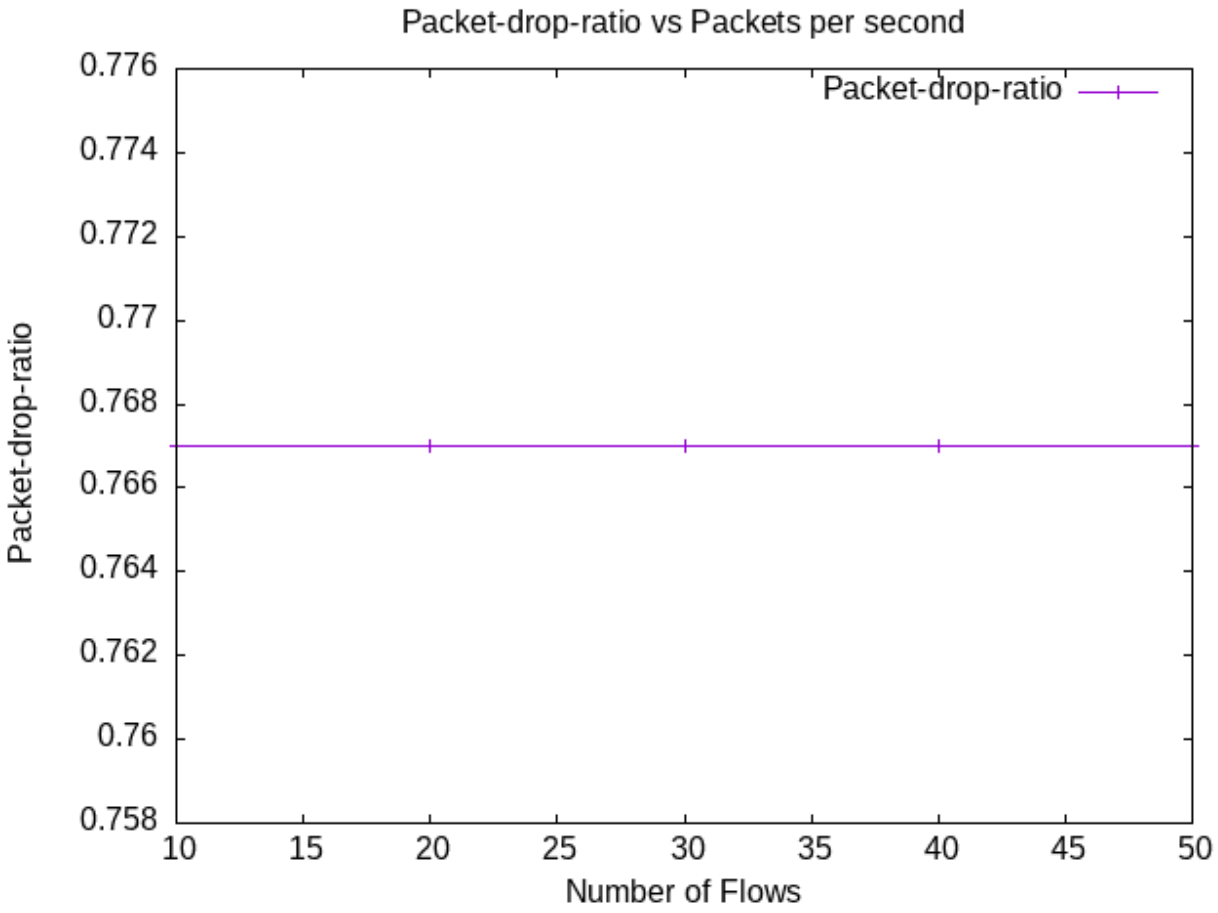

Throughput vs Packets per second
Throughput
98.8
98.6
98.4
98.2
98
97.8
97.6
97.4
97.2
97
96.8
96.6
Throughput
10
15
20
25
30
35
40
45
50
Number of flows

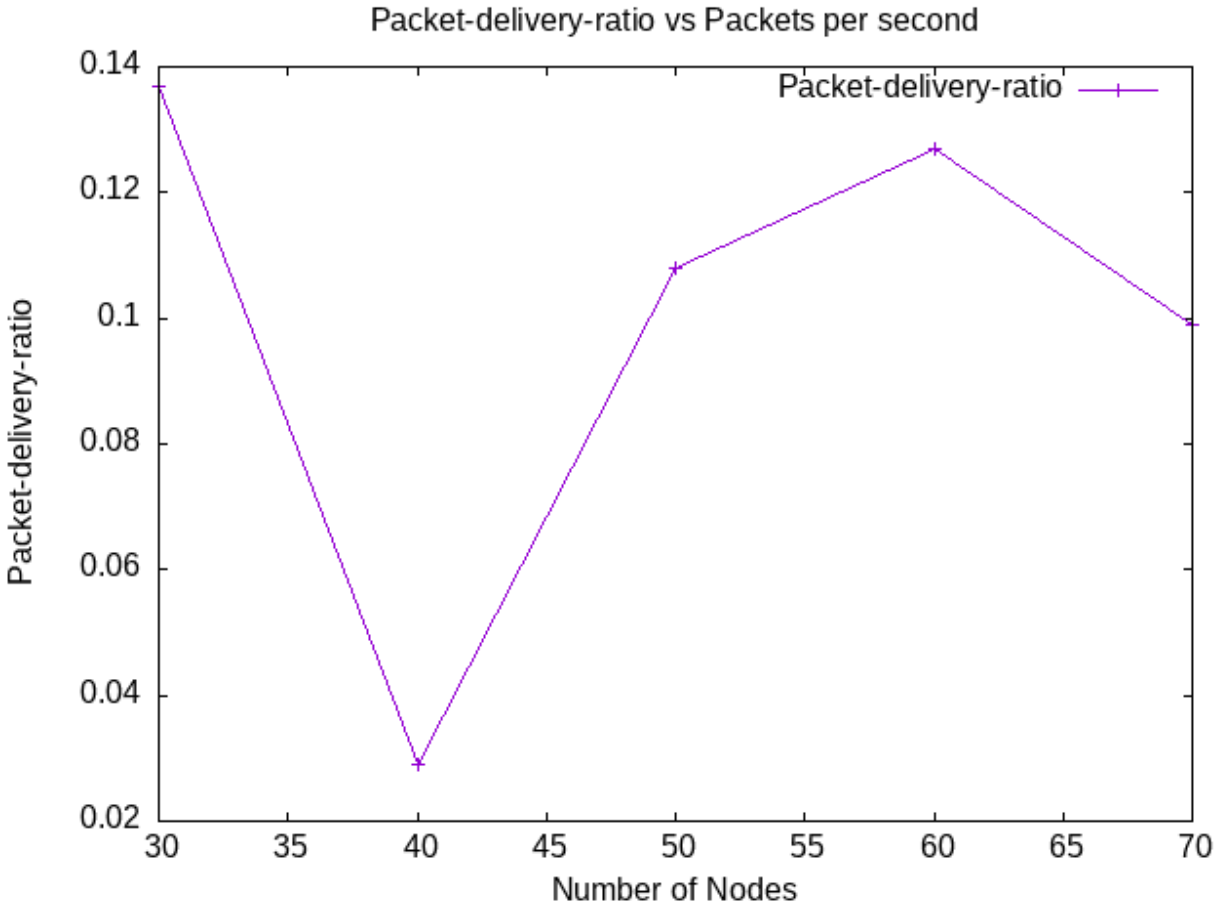

Throughput vs Packets per second
Throughput
98.8
98.6
98.4
98.2
98
97.8
97.6
97.4
97.2
97
96.8
96.6
Throughput
10
15
20
25
30
35
40
45
50
Number of Flows

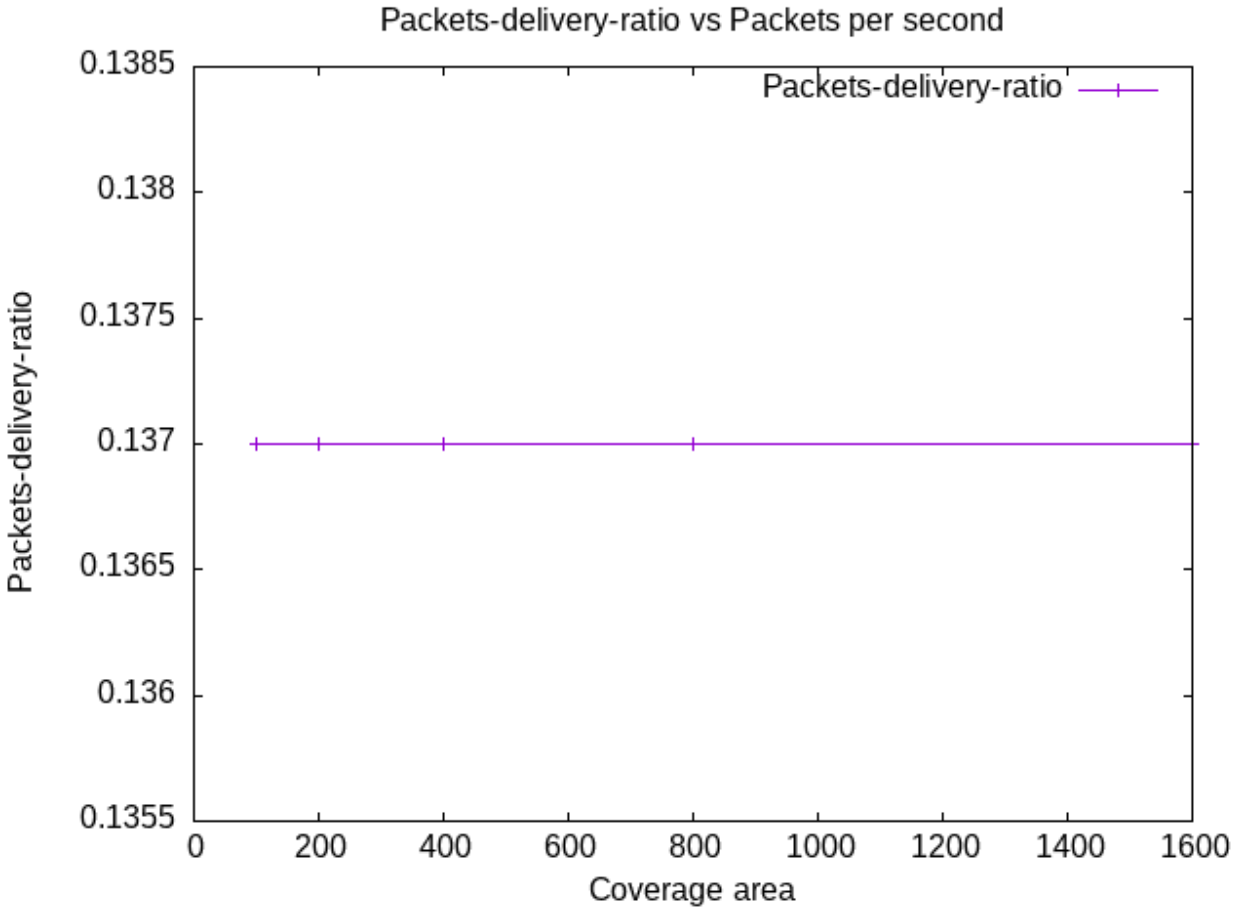

Throughput vs Packets per second
Throughput
0.75
0.748
0.746
0.744
0.742
0.74
0.738
0.736
0.734
Throughput
0
200
400
600
800
1000
1200
1400
1600
Coverage area

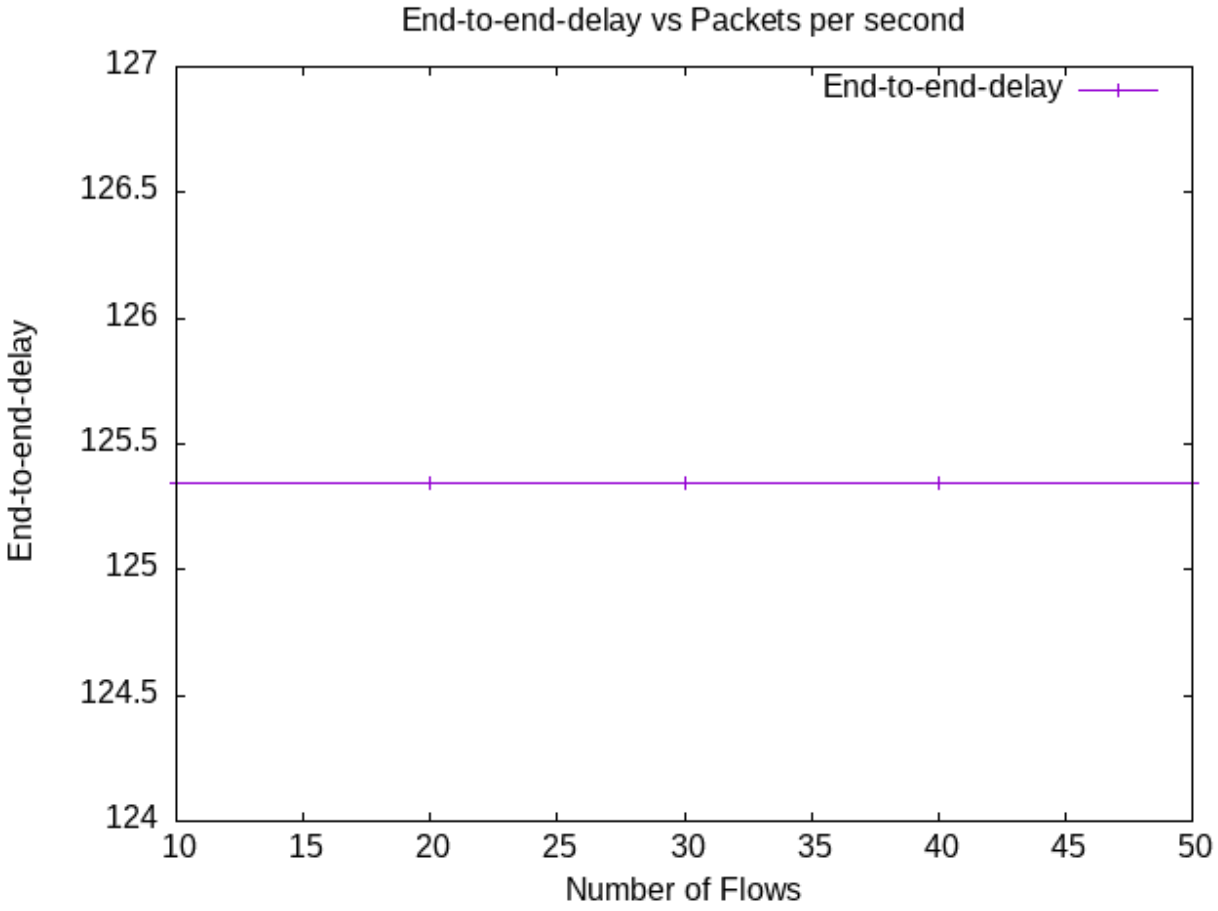

Packets-drop-ratio vs Packets per second
Packets-drop-ratio
Packets-drop-ratio
0.79
0.788
0.786
0.784
0.782
0.78
0.778
0.776
0.774
0
200
400
600
800
1000
1200
1400
1600
Coverage area

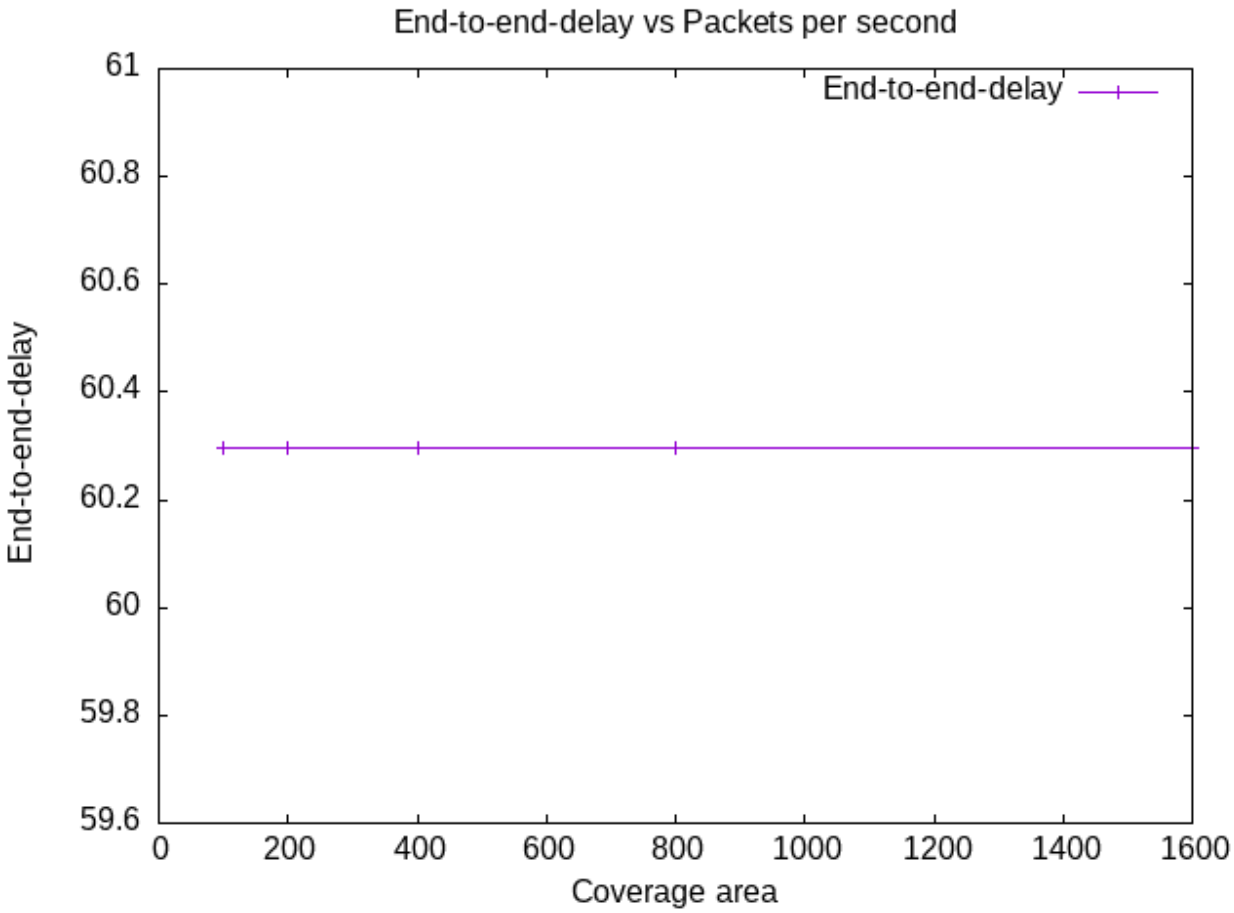

Packet-drop-ratio vs Packets per second
Packet-drop-ratio
Packet-drop-ratio
Number of Nodes
0.81
0.805
0.8
0.795
0.79
0.785
0.78
0.775
0.77
0.765
30
35
40
45
50
55
60
65
70

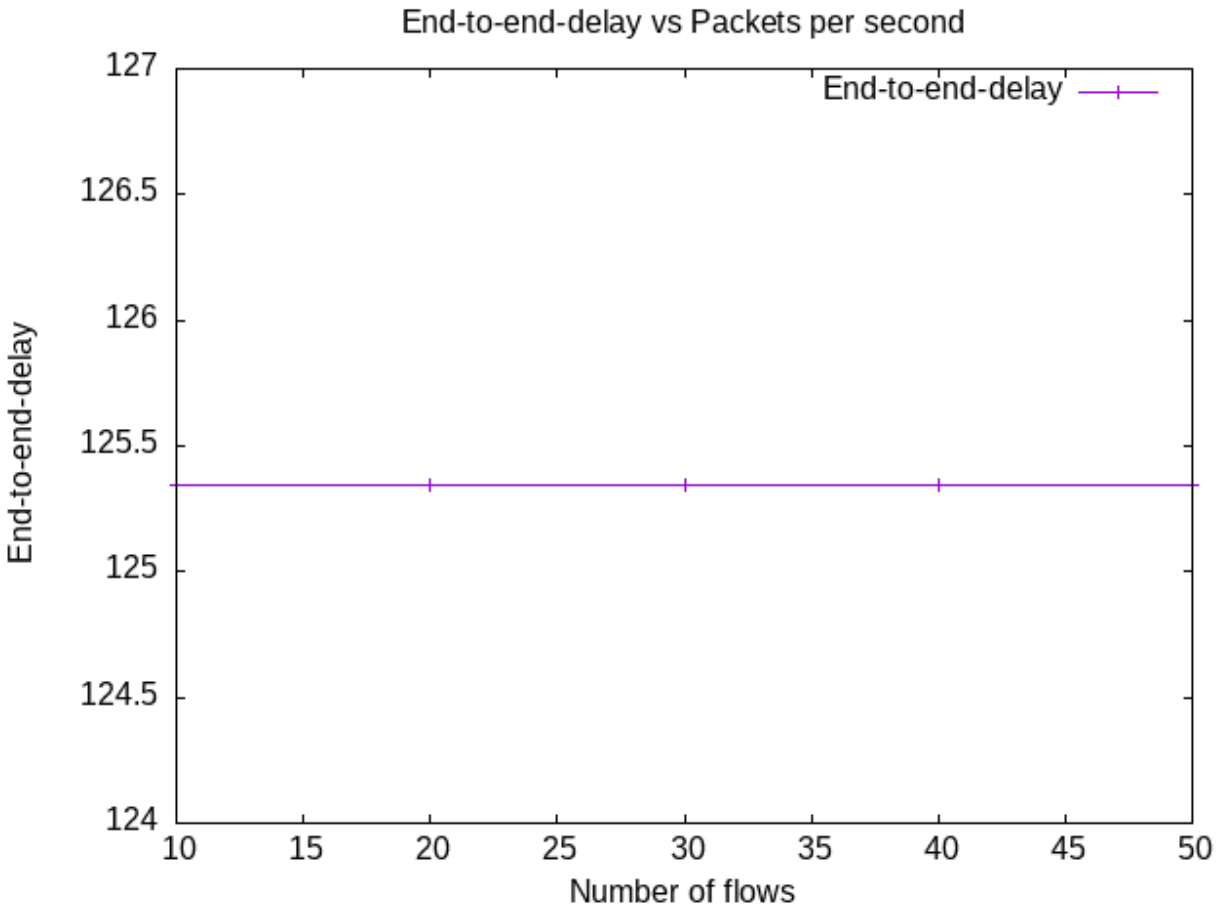

Packet-drop-ratio vs Packets per second
Packet-drop-ratio
0.776
0.774
0.772
0.77
0.768
0.766
0.764
0.762
0.76
0.758
Packet-drop-ratio
10
15
20
25
30
35
40
45
50
Number of Flows


Packet-delivery-ratio vs Packets per second
Packet-delivery-ratio
Packet-delivery-ratio
0.14
0.12
0.1
0.08
0.06
0.04
0.02
30
35
40
45
50
55
60
65
70
Number of Nodes

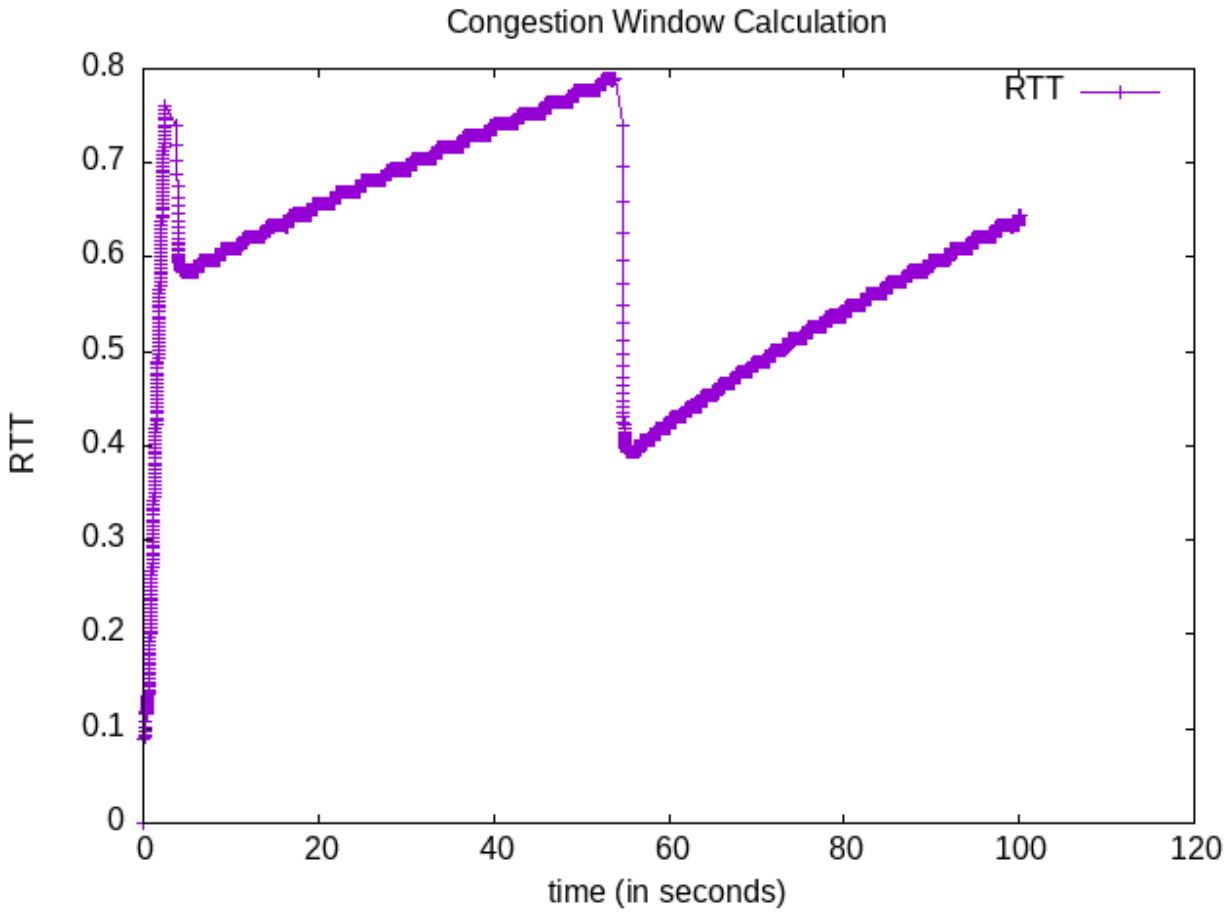

Packets-delivery-ratio vs Packets per second
Packets-delivery-ratio
Packets-delivery-ratio
0.1385
0.138
0.1375
0.137
0.1365
0.136
0.1355
0
200
400
600
800
1000
1200
1400
1600
Coverage area

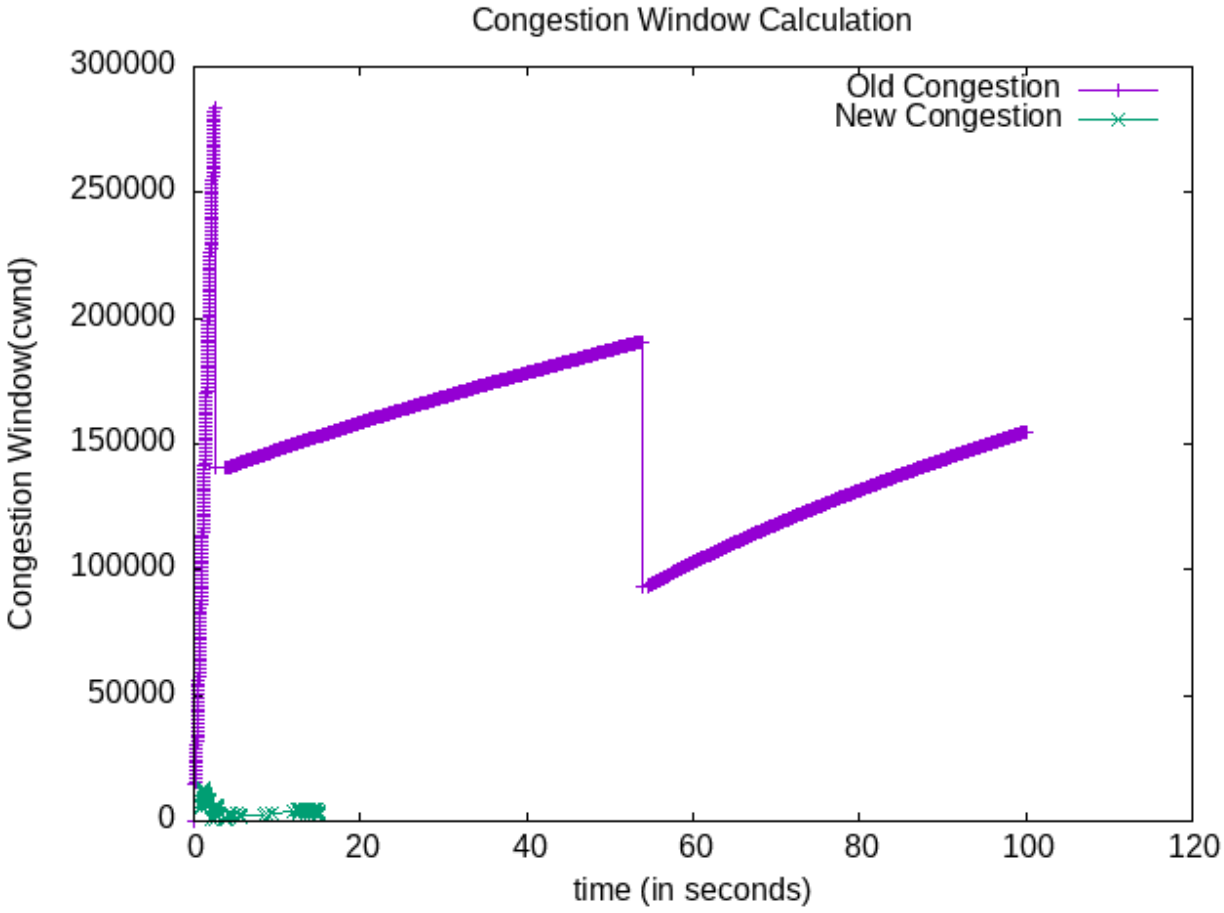

End-to-end-delay vs Packets per second
End-to-end-delay
127
126.5
126
125.5
125
124.5
124
End-to-end-delay
10
15
20
25
30
35
40
45
50
Number of Flows

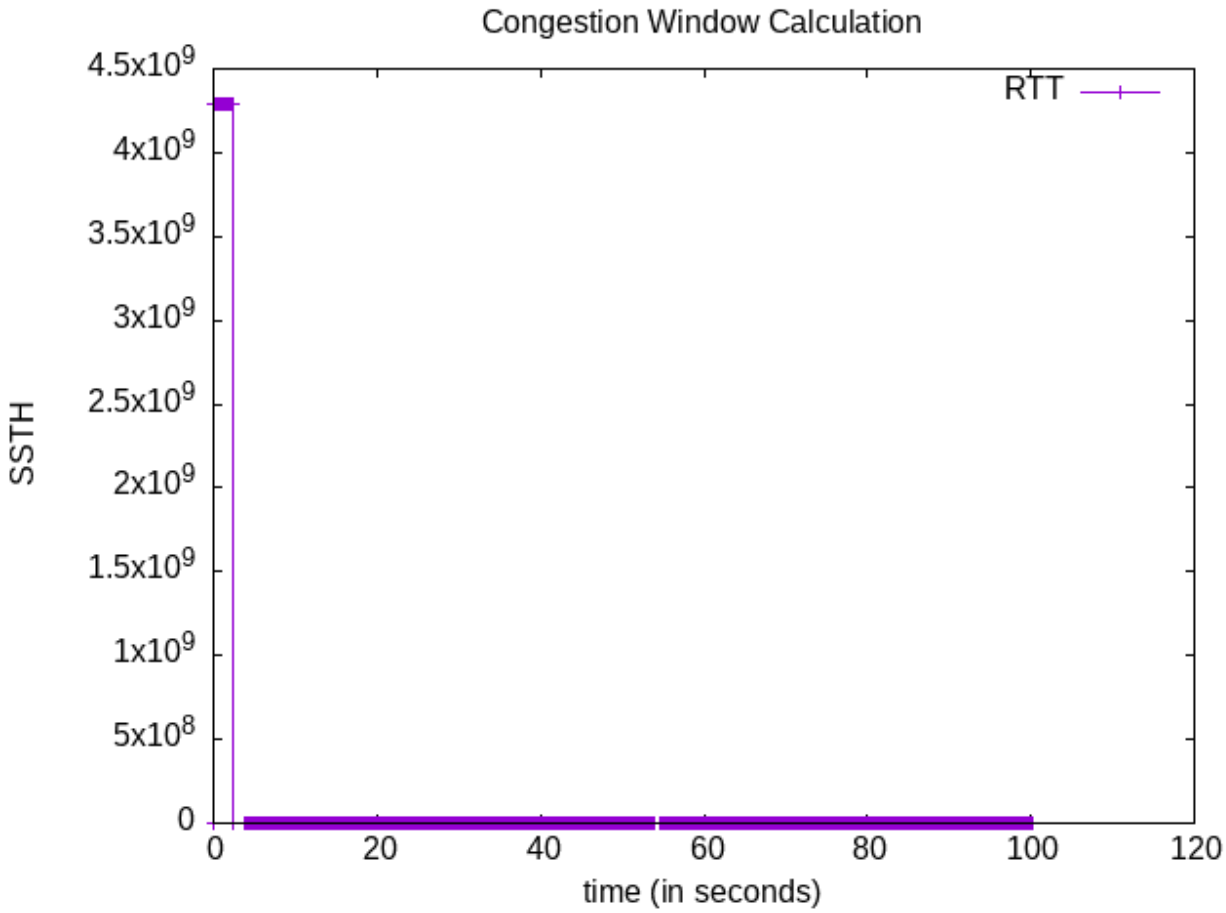

End-to-end-delay vs Packets per second
End-to-end-delay
End-to-end-delay
61
60.8
60.6
60.4
60.2
60
59.8
59.6
0
200
400
600
800
1000
1200
1400
1600
Coverage area



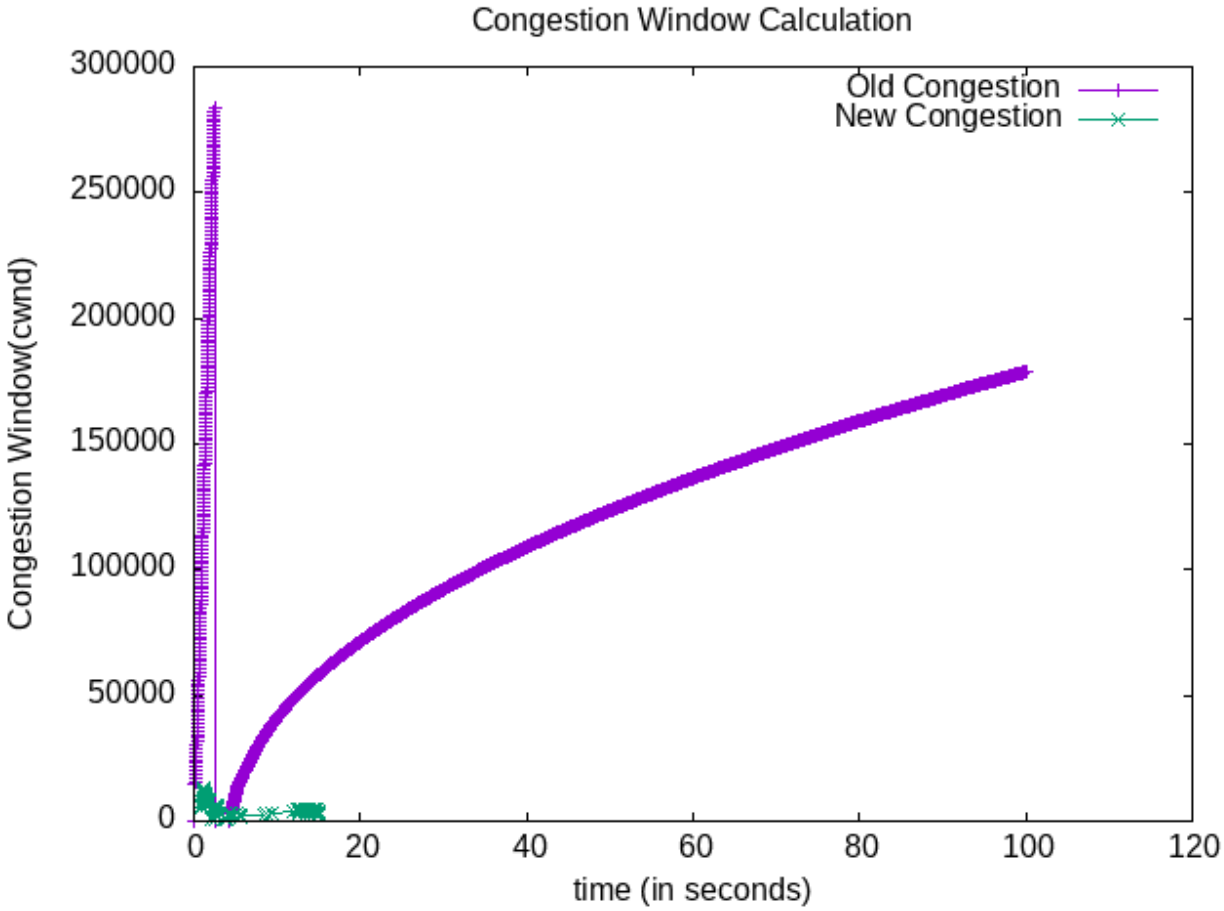
Results with Graphs (TASK B):

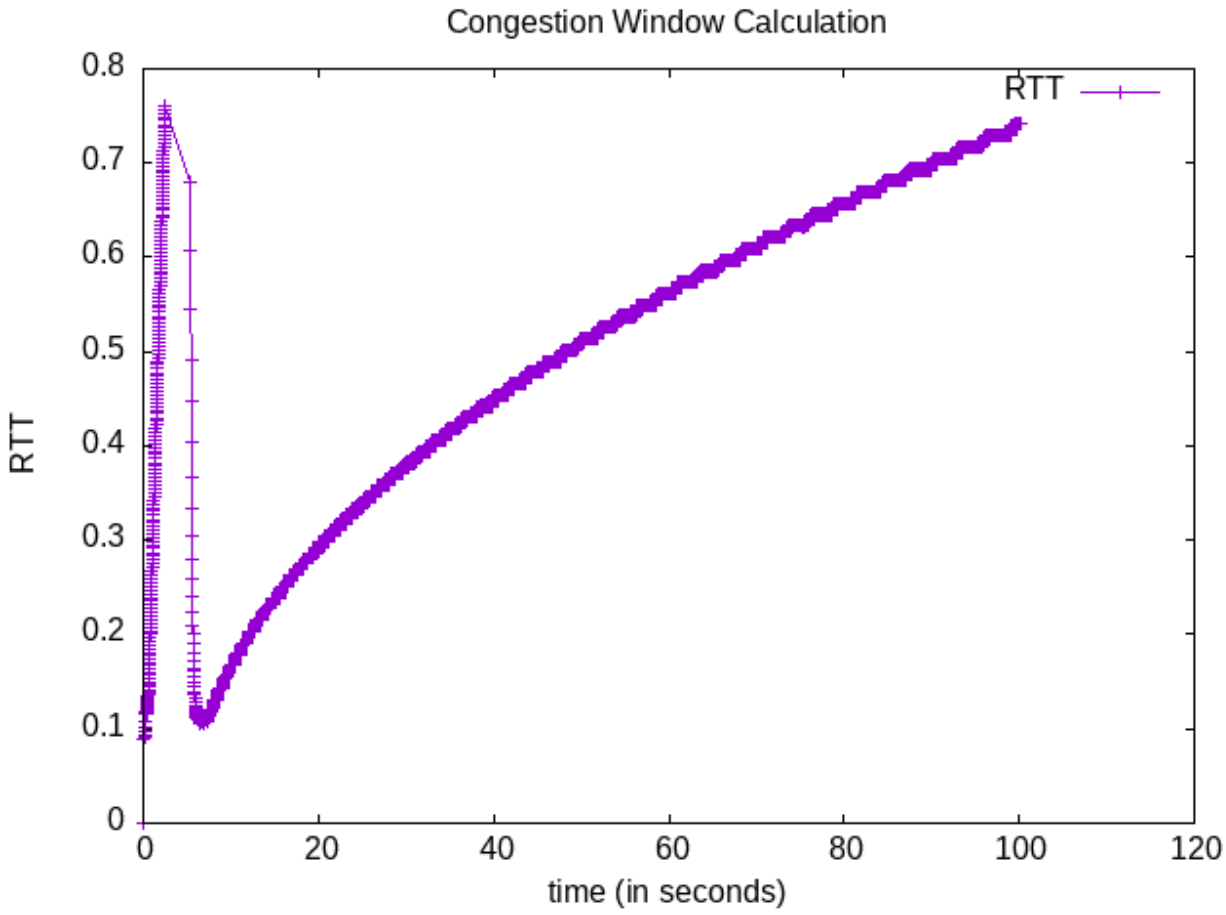

Congestion Window Calculation
RTT
RTT
time (in seconds)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
20
40
60
80
100
120


Congestion Window Calculation
Old Congestion
New Congestion
Congestion Window(cwnd)
time (in seconds)
300000
250000
200000
150000
100000
50000
0
0
20
40
60
80
100
120



Our modified algo's graph:


Congestion Window Calculation
Old Congestion
New Congestion
Congestion Window(cwnd)
time (in seconds)
300000
250000
200000
150000
100000
50000
0
0
20
40
60
80
100
120



Summary Findings:

TASK A:

From task A, we can see from the graphs generated in the wired topology and Wireless low-rate (e.g., 802.15.4) (static). Also, if we look at the metric of average network throughput, we can see that with the increase of flows in the inter-network, the throughput increases but it will reach a saturation point which will be equal to bandwidth of the bottleneck link. On the contrary, we can see a downward trend in the metric of packet drop ratio. Therefore, we can see an inverse relationship between throughput and packet drop ration here. Again, in

the wireless topology, we can see the packet drop ratio is significantly more in wireless than in wired network. The reason for it is I think down to the “RangePropagationLossModel” installed in the wireless channel which basically means the farther a node is, the more the loss. This channel behaviour may result in the randomness noticed in the generated plots. Also analyzing the flows generated by the flow monitor, it can be seen some nodes in the wifi network receive significantly less packets than other wifi nodes and that can be due to the distance of these nodes from the router node.

TASK B:

We proposed a new congestion control scheme to improve the performance of bandwidth reservation networks.The following results should be gained-

- The proposed scheme stabilizes the congestion window size.
- The proposed scheme has greater throughput than TCP New Reno.

As there can be some problems while simulating the networks and implementing the new algorithm.There maybe some bug left in the new algorithm of TcpBR.That’s why we didn’t get the desired output result.