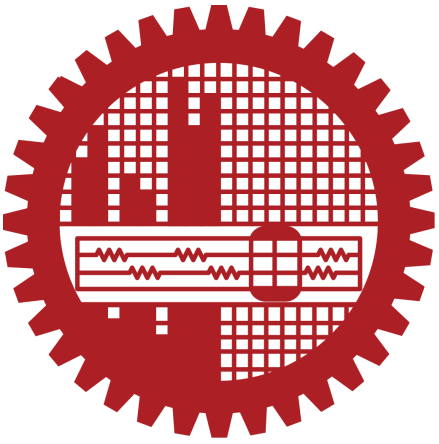
Bangladesh University of Engineering and Technology
Department of Computer Science and Engineering



CSE 322

Networking Sessional

NS3 Project Presentation

Name	Sumaiya Azad
Student ID	1705048
Adviser	Syed Md. Mukit Rashid
Date	23 February, 2022

Implemented Literature

TCP-Fusion: A Hybrid Congestion Control Algorithm for High-speed Networks

Kazumi KANEKO, Tomoki FUJIKAWA, Zhou SU and Jiro KATTO

Graduate School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjyuku-ku, Tokyo, 169-8555 Japan
E-mail: {kaneko, katto}@katto.comm.waseda.ac.jp

Journal: Proceedings of Protocols for fast long-distance network

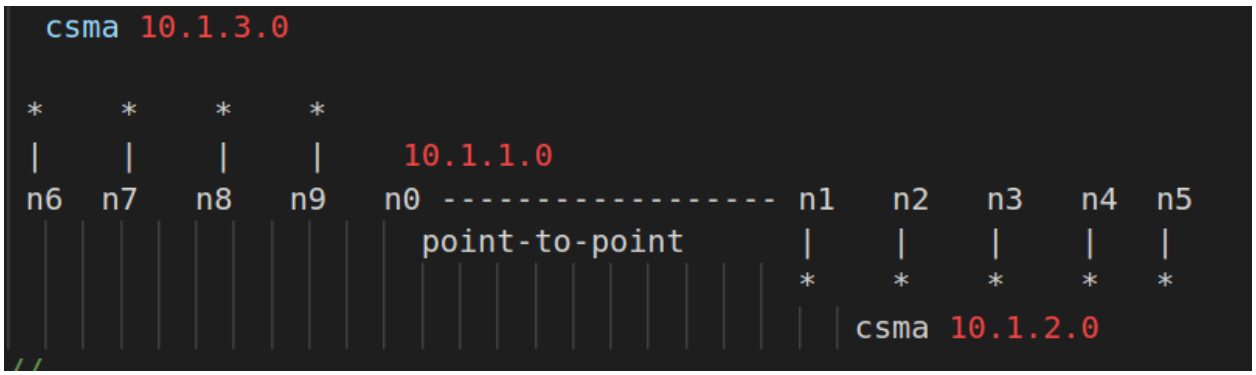
Published: 2007

Tokyo, Japan

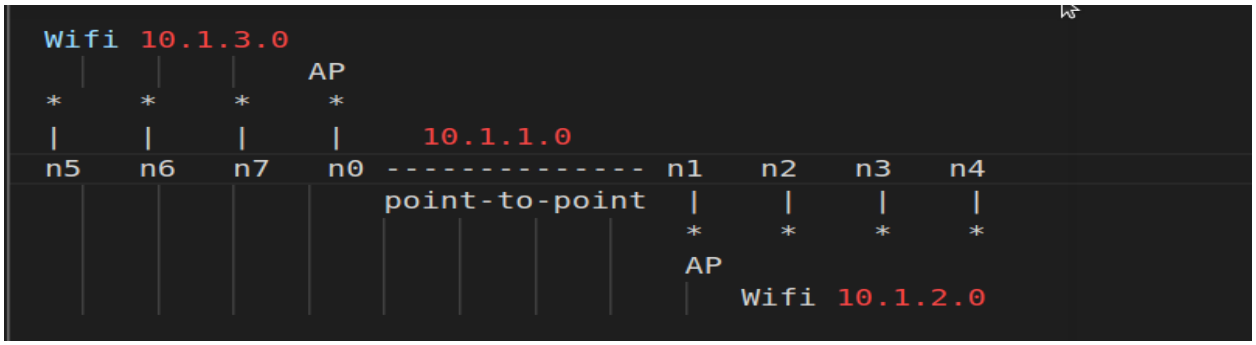
Link: https://www.researchgate.net/publication/242414090_TCP-Fusion_A_Hybrid_Congestion_Control_Algorithm_for_High-speed_Networks

Network topologies under simulation

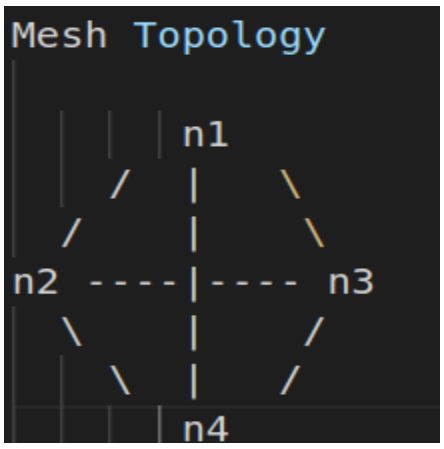
Wired Topology



High Rate Wireless Topology



Low Rate Wireless Topology



Parameters under variation

Task A

Roll No: 1705048
1705048 % 6 = 4

Parameters to be varied

1. For Wireless High Rate (Static)
    a. Number of Nodes (20, 40, 60, 80, 100)
    b. Number of Flows (10, 20, 30, 40, 50)
    c. Number of Packets Per Second (100, 200, 300, 400, 500)
    d. Range of Coverage Area (Tx_range, 2 x Tx_range, 3 x Tx_range, Tx_range, and 5 x Tx_range)
2. For Wireless Low Rate (Mobile)
    a. Number of Nodes (20, 40, 60, 80, 100)
    b. Number of Flows (10, 20, 30, 40, 50)
    c. Number of Packets Per Second (100, 200, 300, 400, 500)
    d. Speed of nodes (5 m/s, 10 m/s, 15 m/s, 20 m/s, and 25 m/s)

Task B

- In wired network, varied error rate to plot the change in congestion window
- In wired network, varied simulation start time of multiple flow (For TCP NewReno, TCP Vegas, TCP Westwood, TCP Fusion)
- In wired network, varied simulation start time and error rate of multiple flow (For TCP Fusion)
- In wired network, varied drop ratio to plot throughput of congestion control algorithm (For TCP NewReno, TCP Cubic, TCP Fusion, TCP Vegas, TCP Westwood)
- In wired network, varied packets per second to plot throughput
- In wireless network, varied simulation start time of multiple flow (For TCP Fusion)
- Congestion window in wireless network

Overview of the proposed algorithm

TCP-Fusion maintains two congestion window sizes. The one has two properties of TCP-Vegas and TCPW, which provide efficiency in large leaky pipe, i.e., the link with large bandwidth-delay product and non-negligible random losses. The other one is updated like TCP-Reno, i.e., the value is increased by 1MSS/RTT and halved upon packet losses, and then TCP-Fusion adopts either big one as its new congestion window size. Therefore, TCP-Fusion ensures at least TCP-Reno performance.

Congestion Window Reduction Phase

TCP-Fusion adopts optimization of the decrease parameter based on TCPW-RE (Rate Estimation) to improve efficiency particularly in the leaky pipe. In TCPW-RE, the decrease parameter after a loss can be expressed as RTT min /RTT, where RTT min and RTT are the minimum RTT and the RTT right before the packet loss, respectively. This equation indicates that TCPW-RE reduces its congestion window size to clear the buffer and, as a result, it is friendly to TCP-Reno only if the buffer capacity is equal to the BDP [10, 17] where RTT grows up to 2*RTT min . If the buffer capacity is larger than the BDP, the decrease value is less than 1/2 upon a congestion loss and TCPW-RE cannot obtain a share of the bandwidth. To address this issue, we then set the thresholds to 1/2 as follows,

$$cwnd_{new} = \max(\frac{RTT_{\min}}{RTT} cwnd_{last}, \frac{cwnd_{last}}{2})$$

Here,
RTT_{min} = the minimum RTT
RTT = the RTT right before the packet loss
$Cwnd_{last}$ = congestion window sizes right before the packet loss
$Cwnd_{new}$ = congestion window sizes right after the packet loss

Congestion Window Increase

Similar to TCP-Vegas, TCP-Fusion has three phases- increase phase, decrease phase, and steady phase, which are switched by a number of packets in the bottleneck queue (diff). The diff can be estimated as;

$$diff = cwnd \frac{(RTT - RTT_{\min})}{RTT}$$

If the diff is less than the lower bound threshold, the link is determined as underutilized, and its congestion window size is increased rapidly to fill the pipe size. If the diff is larger than the upper bound threshold, the link is determined as utilized and early congestion, and its congestion window size is decreased to the value that has at least the lower bound threshold in the bottleneck queue. Otherwise, the link is determined in a good balance and non-congestion condition, and its congestion window size is fixed.

$$cwnd_{new} = \begin{cases} cwnd_{last} + W_{inc} / cwnd_{last}, & \text{if } diff < \alpha \\ cwnd_{last} + (-diff + \alpha) / cwnd_{last}, & \text{if } diff > 3 * \alpha \\ cwnd_{last}, & \text{otherwise} \end{cases}$$

$$\text{cwnd}_{\text{new}} = reno\_cwnd, \quad \text{if } cwnd_{new} < reno\_cwnd$$

Here,
$Cwnd_{last}$ = congestion window sizes right before the packet loss
$Cwnd_{new}$ = congestion window sizes right after the packet loss
reno_cwnd = congestion window size equivalent to TCP-Reno
α = the lower bound threshold to switch three phases
W_{inc} = the increment parameter to increase congestion window size

Modifications made in the simulator

Task A

High Rate

1. Modified third.cc, sixth.cc and seventh.cc files
2. Added TCP congestion control algorithm and relevant parameters
3. Added wireless network
4. Added mobility model to the wifi static and access point devices
5. Added application class
6. Modified the file to add multiple flows between different sources and destinations
7. Varied node counts for graph plotting
8. Added payload size, packets per second and data rate calculation
9. Added RangePropagationLossModel::MaxRange for varying coverage
10. Added flowmonitor for network simulation

Low Rate

1. Modified example-ping-lr-wpan-mesh-under.cc file
2. Removed ping6 attributes
3. Removed csma device related segment
4. Modified the file to add multiple flows between different sources and destinations
5. Added IP header and TCP header
6. Added payload size and data rate calculation
7. Varied node counts for simulation
8. Added speed in the mobility model
9. Created mesh topology between nodes
10. Added flowmonitor for network simulation

Task B

- Modified wscript file in src/internet folder.
- Added tcp-fusion.h and tcp-fusion.cc in src/internet/model folder.
- Implemented the algorithm in tcp-fusion.h and tcp-fusion.cc files.

Wired Topology

1. Modified sixth.cc and seventh.cc files
2. Added TCP congestion control algorithm and relevant parameters
3. Added csmaDevices for wired network
4. Added error model to the devices
5. Added application class
6. Modified the file to add multiple flows between different sources and destinations
7. Varied node counts for graph plotting
8. Added payload size, packets per second and data rate calculation
9. Added necessary functions for congestion window, throughput, drop ratio
10. Added flowmonitor for network simulation

Wireless Topology

Same as Task A

Modification in tcp-fusion.cc

```
void
TcpFusion::EstimateBW (const Time &rtt, Ptr<TcpSocketState> tcb)
{
  NS_LOG_FUNCTION (this);

  NS_ASSERT (!rtt.IsZero ());

  m_currentBW = m_ackedSegments * tcb->m_segmentSize / rtt.GetSeconds ();


  Time currentAck = Simulator::Now ();
  m_currentBW = m_ackedSegments * tcb->m_segmentSize / (currentAck - m_lastAck).GetSeconds ();
  m_lastAck = currentAck;

  m_ackedSegments = 0;
  NS_LOG_LOGIC ("Estimated BW: " << m_currentBW);

  // Filter the BW sample
  NS_LOG_LOGIC ("Estimated BW after filtering: " << m_currentBW);
}
```

Figure: Estimate bandwidth function in tcp-fusion.cc

```
void
TcpFusion::PktsAcked (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked,
                      const Time& rtt)
{
  NS_LOG_FUNCTION (this << tcb << segmentsAcked << rtt);

  if (rtt.IsZero ())
    {
      return;
    }
  m_ackedSegments += segmentsAcked;
  EstimateBW (rtt, tcb);

  m_currentRtt = rtt;
  // std::cout << " seconds : " << m_currentRtt.GetSeconds () << "\n";

  m_minRtt = std::min (m_minRtt, rtt);
  NS_LOG_DEBUG ("Updated m_minRtt = " << m_minRtt);

  m_baseRtt = std::min (m_baseRtt, rtt);
  NS_LOG_DEBUG ("Updated m_baseRtt = " << m_baseRtt);

  // Update RTT counter
  m_cntRtt++;
  NS_LOG_DEBUG ("Updated m_cntRtt = " << m_cntRtt);
}
```

Figure: PktsAcked function in tcp-fusion.cc

```
void
TcpFusion::IncreaseWindow (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
  // std::cout << " < reno window\n";
    if (tcb->m_cWnd < tcb->m_ssThresh)
    {
      // std::cout << "tcb->m_cWnd < tcb->m_ssThresh\n";
      // std::cout <<Simulator::Now () <<" prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
      segmentsAcked = TcpNewReno::SlowStart (tcb, segmentsAcked);
      // std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
    }
    if (tcb->m_cWnd >= tcb->m_ssThresh)
    { uint32_t  reno_cWnd = tcb->m_cWnd;
      std::cout << "tcb->m_cWnd >= tcb->m_ssThresh\n";
      if (m_cntRtt < 3)
      {
        std::cout << "m_cntRtt < 3\n";
        std::cout <<Simulator::Now () <<"  prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
        TcpNewReno::IncreaseWindow (tcb, segmentsAcked);
        std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
      }
      else if(m_currentRtt.GetSeconds ()>2*m_minRtt.GetSeconds ()){
        std::cout << "m_currentRtt.GetSeconds ()>2*m_minRtt.GetSeconds ()\n";
        std::cout <<Simulator::Now () <<"  prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
        tcb->m_cWnd = std::max ((m_minRtt.GetSeconds ()/m_currentRtt.GetSeconds ()), 0.5) * tcb->m_cWnd.Get ();
        std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
      }
```

Figure: Increase window function in tcp-fusion.cc(Part 1)

```
      else{
        std::cout << "else\n";
        m_wInc=(m_currentBW)/(8*1500);
        m_alpha=(tcb->m_cWnd.Get ()*(.004))/m_currentRtt.GetSeconds ();
        m_diff=(tcb->m_cWnd.Get ()*(m_currentRtt.GetSeconds ()-m_minRtt.GetSeconds ()))/m_currentRtt.GetSeconds ();
        if(m_diff<m_alpha){
          std::cout << "diff < alpha \n";
          std::cout <<Simulator::Now () <<"  prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
          tcb->m_cWnd = tcb->m_cWnd.Get () + (m_wInc)*tcb->m_segmentSize;
          std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
        }else if(m_diff>3*m_alpha){
          std::cout << "diff > alpha \n";
          std::cout <<Simulator::Now () <<"  prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
          tcb->m_cWnd = tcb->m_cWnd.Get () + ((-m_diff+m_alpha)/tcb->m_cWnd.Get ())*tcb->m_segmentSize;
          std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
        }else{
          std::cout << "nothing to do \n";
          std::cout << Simulator::Now () <<"  prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
          std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
        }
        if (segmentsAcked > 0)
        {
          double adder = static_cast<double> (tcb->m_segmentSize * tcb->m_segmentSize) / tcb->m_cWnd.Get ();
          adder = std::max (1.0, adder);
          reno_cWnd += static_cast<uint32_t> (adder);
        }
        if(tcb->m_cWnd<reno_cWnd){
          std::cout << "tcb->m_cWnd<reno_cWnd\n";
          std::cout <<Simulator::Now ()<< " prev congestion window : "<<tcb->m_cWnd.Get ()<<"   ";
          tcb->m_cWnd = reno_cWnd;
          std::cout << "congestion window : "<<tcb->m_cWnd.Get ()<<"\n";
        }
      }
    }
}
```

Figure: Increase window function in tcp-fusion.cc(Part 2)

Results with graphs

Task A High Rate

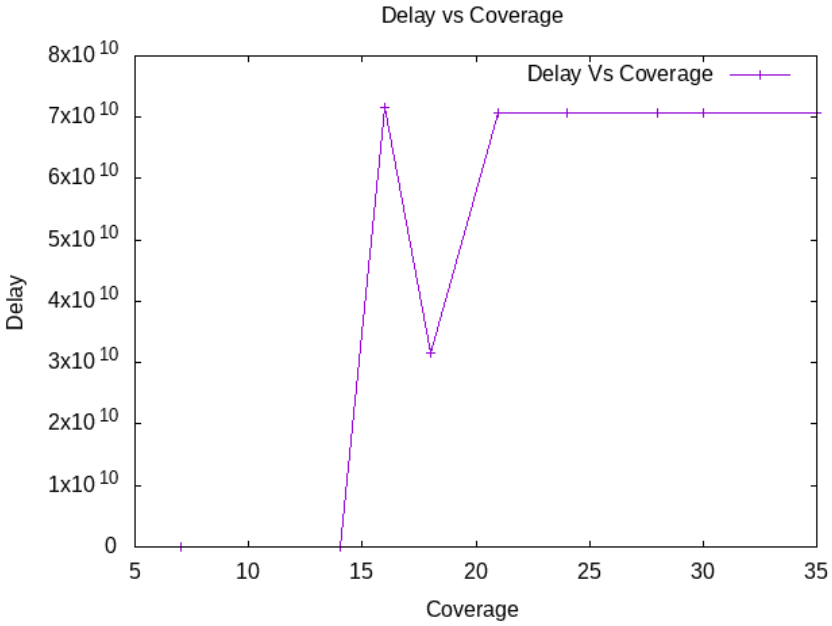


Figure: Wireless high rate delay vs coverage graph

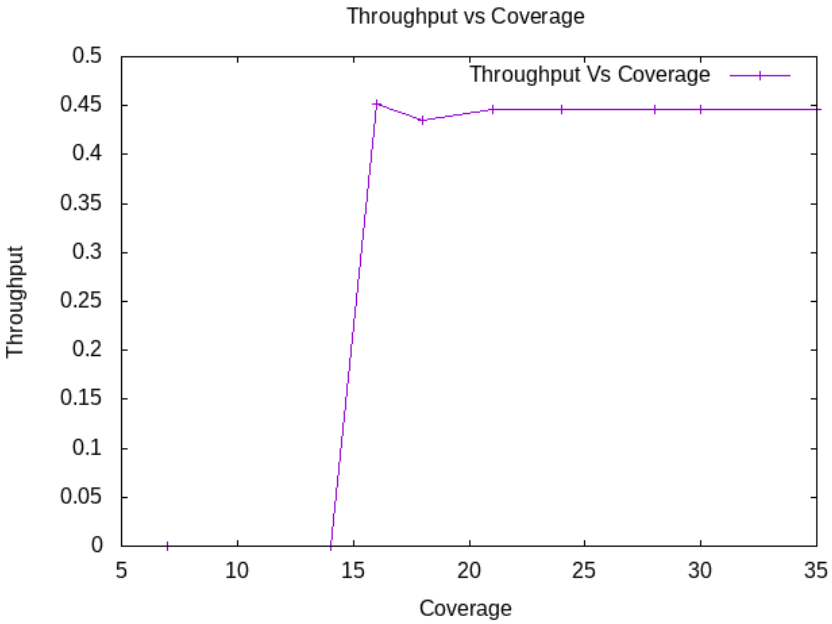


Figure: Wireless high rate throughput vs coverage graph

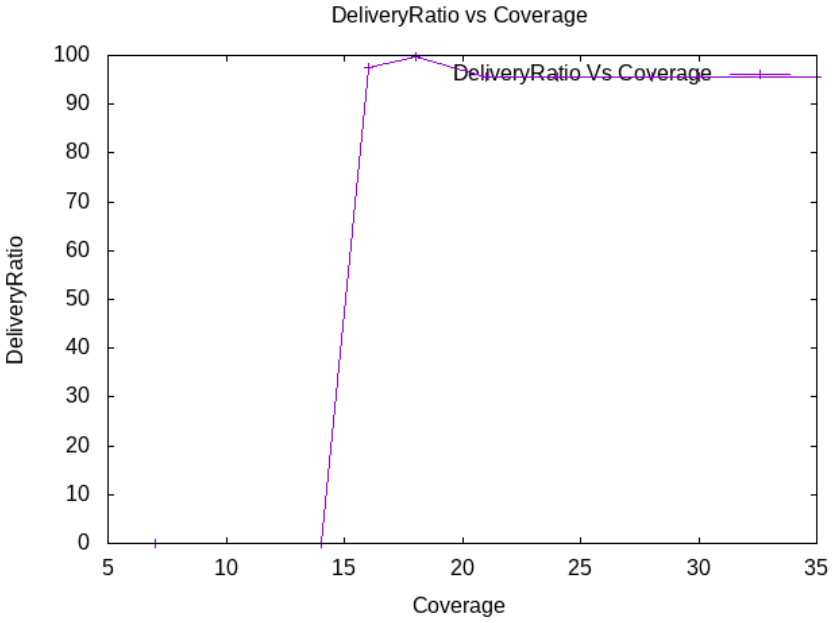


Figure: Wireless high rate delivery ratio vs coverage

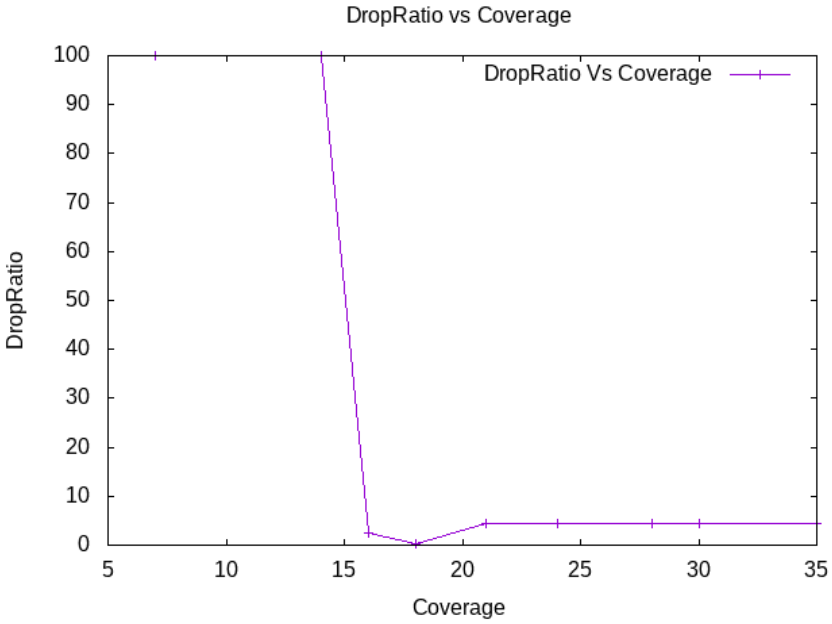


Figure: Wireless high rate drop ratio vs coverage graph

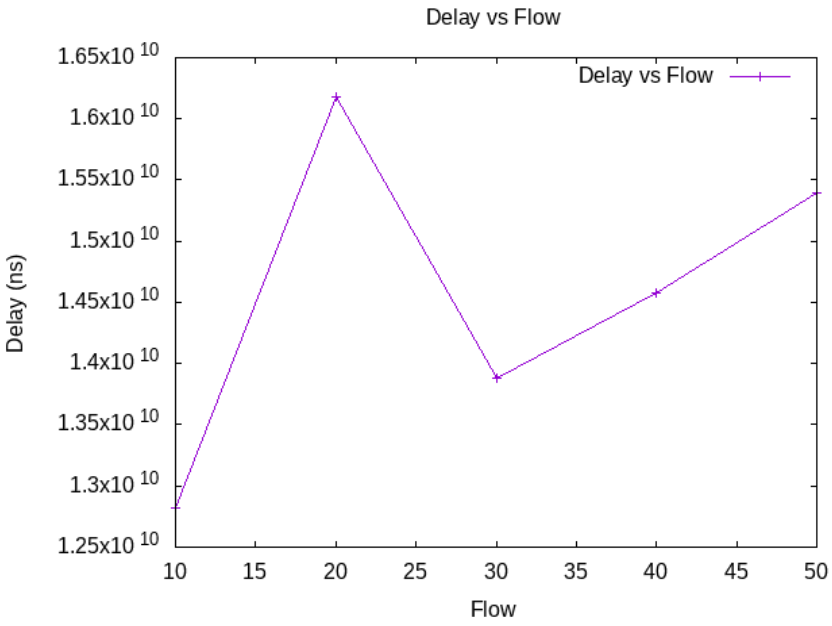


Figure: Wireless high rate delay vs flow graph

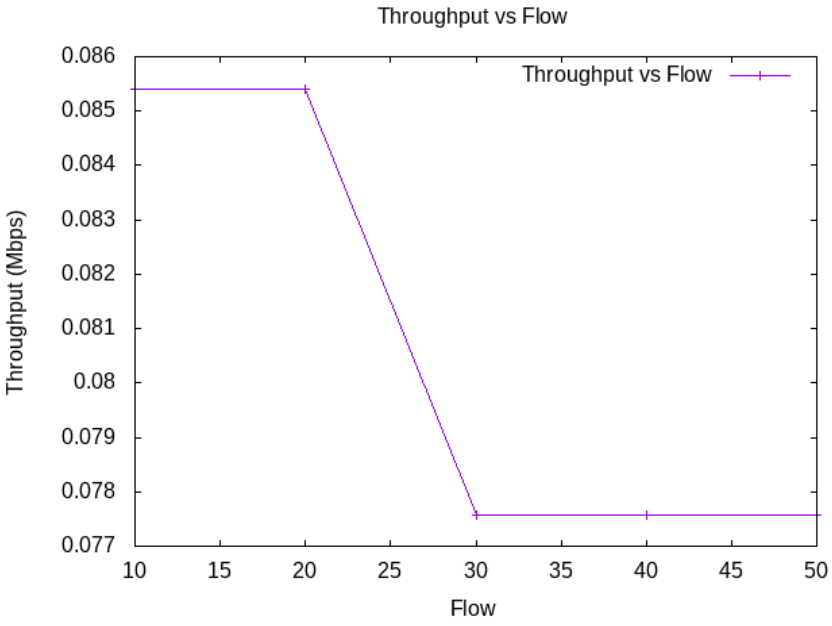


Figure: Wireless high rate throughput vs flow graph

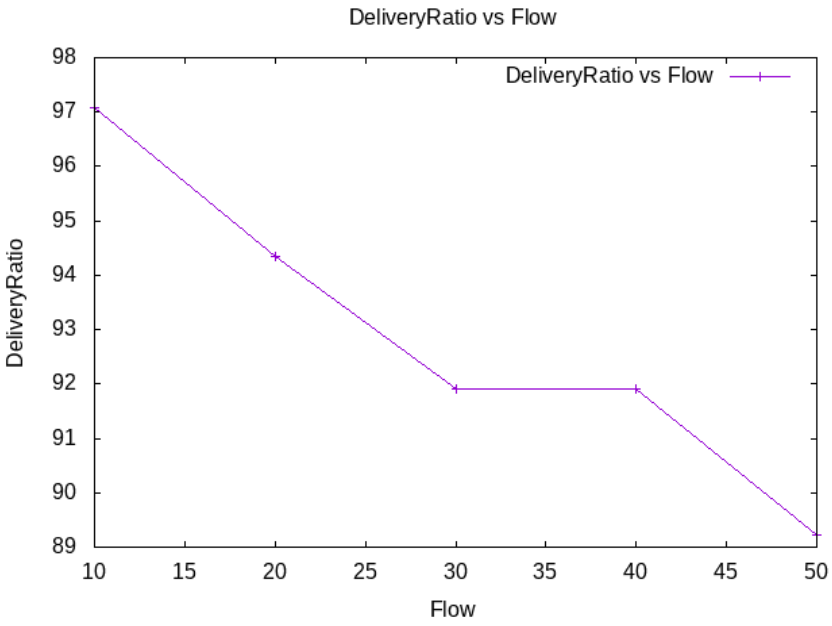


Figure: Wireless high rate delivery ratio vs flow graph

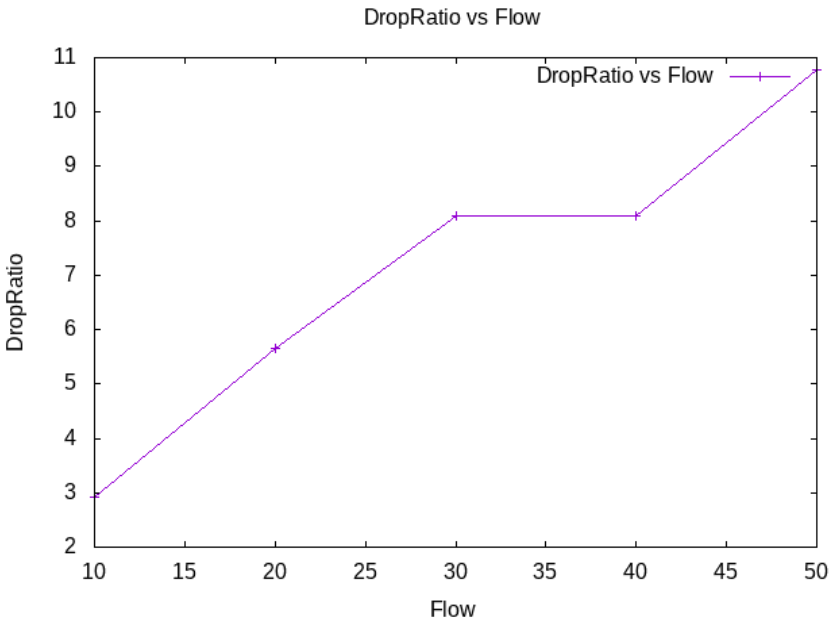


Figure: Wireless high rate drop ratio vs flow graph

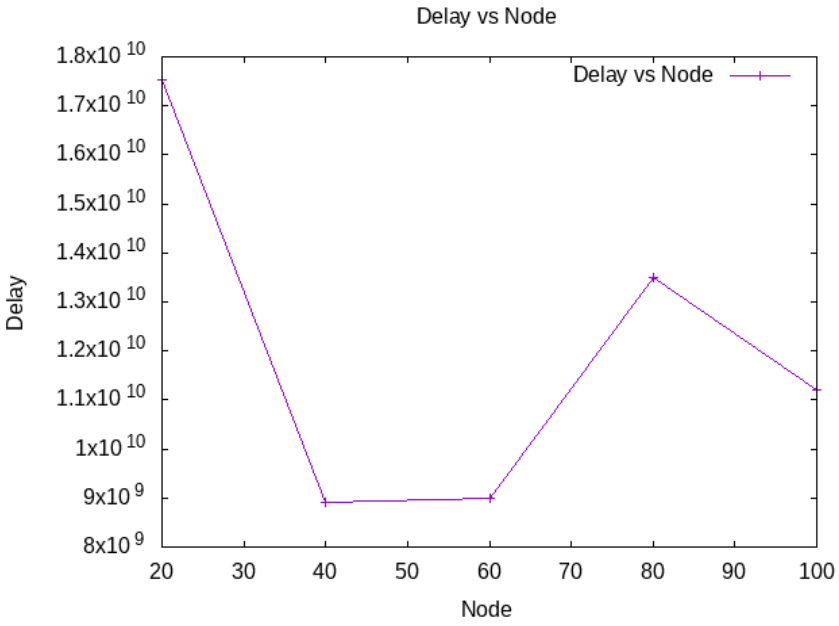


Figure: Wireless high rate delay vs node graph

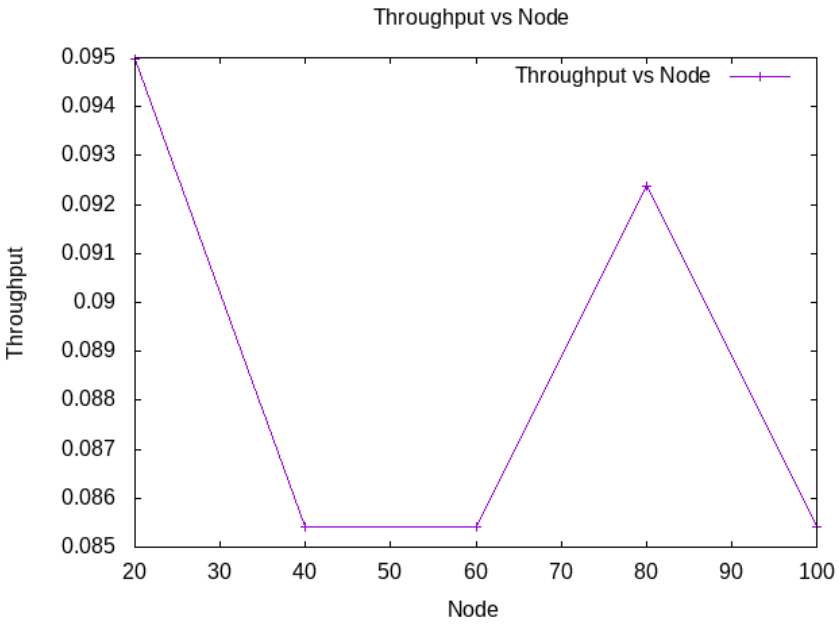


Figure: Wireless high rate throughput vs node graph

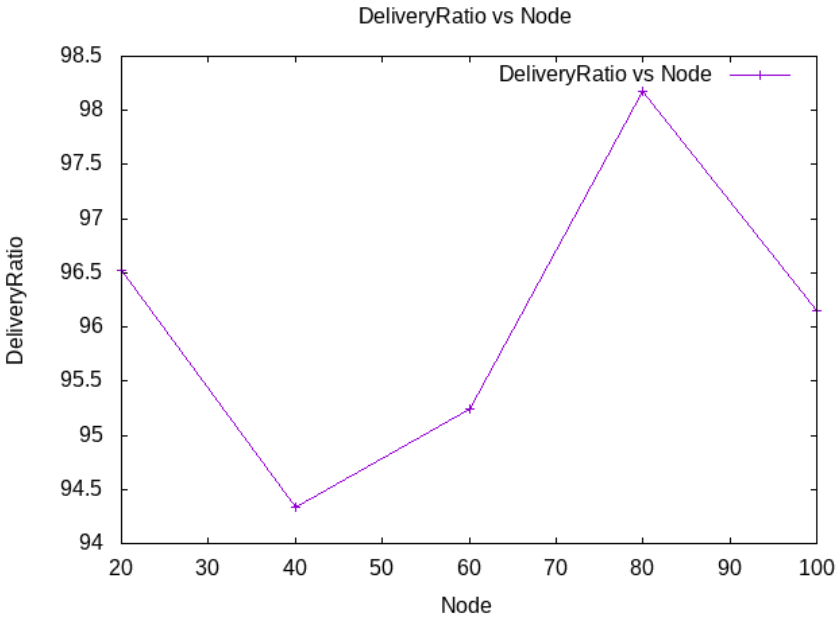


Figure: Wireless high rate delivery ratio vs node graph

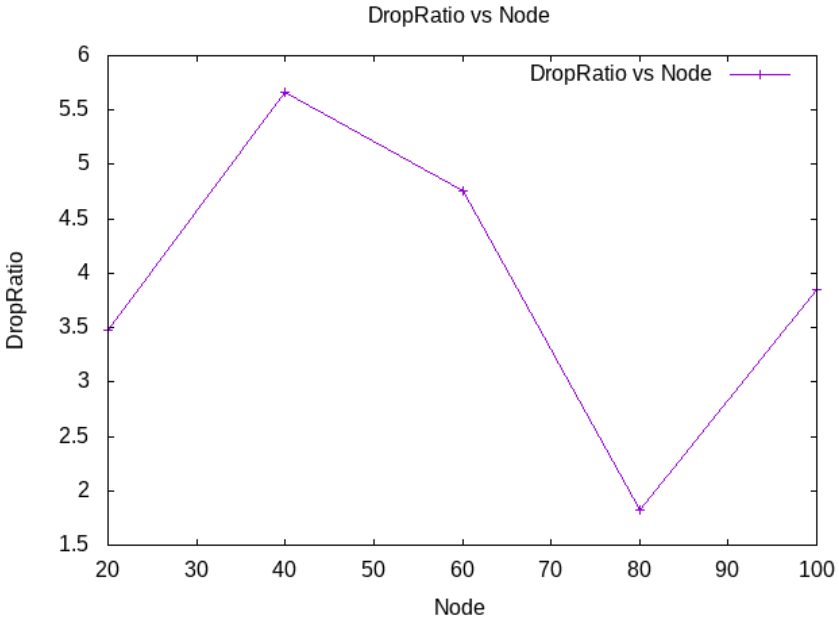


Figure: Wireless high rate drop ratio vs node graph

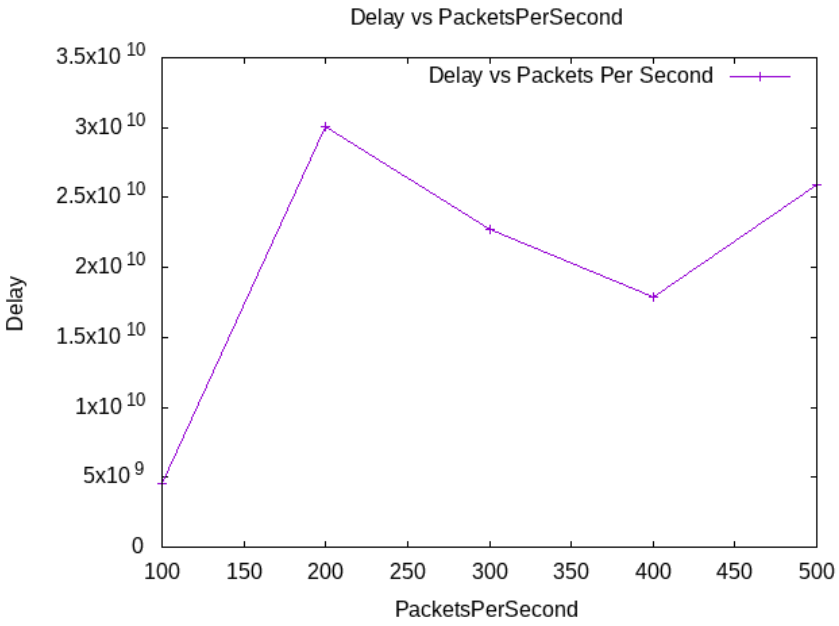


Figure: Wireless high rate delay vs packets per second graph

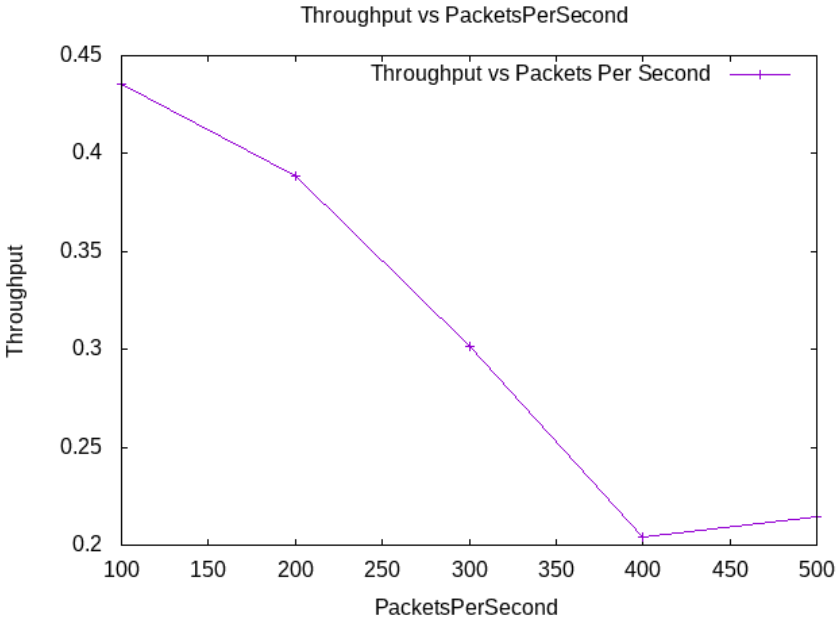


Figure: Wireless high rate throughput vs packets per second graph

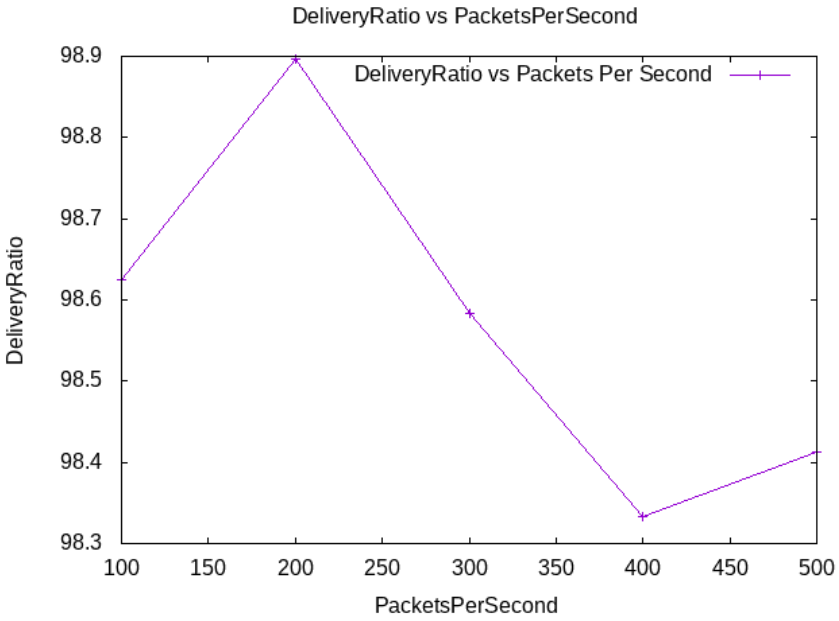


Figure: Wireless high rate delivery ratio vs packets per second graph

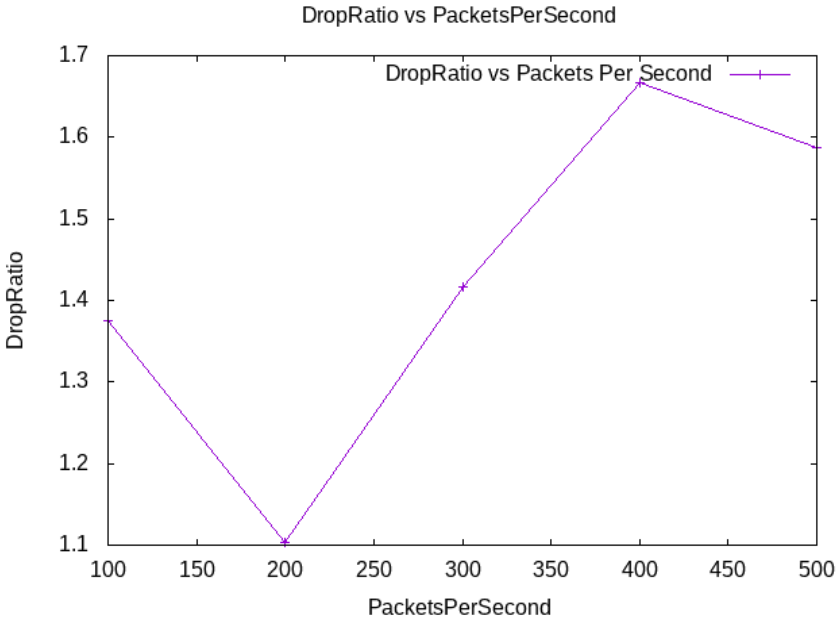


Figure: Wireless high rate drop ratio vs packets per second graph

Task A Low Rate

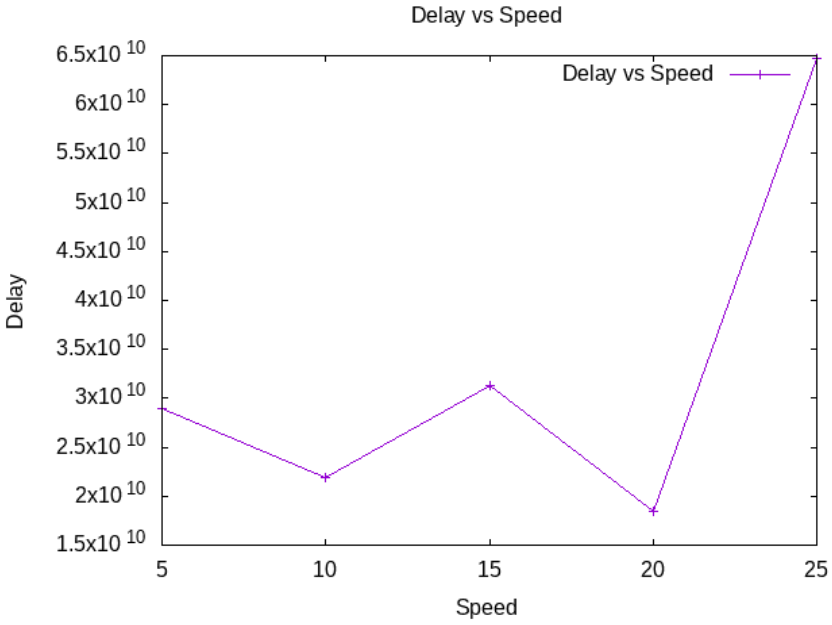


Figure: Wireless low rate delay vs speed graph

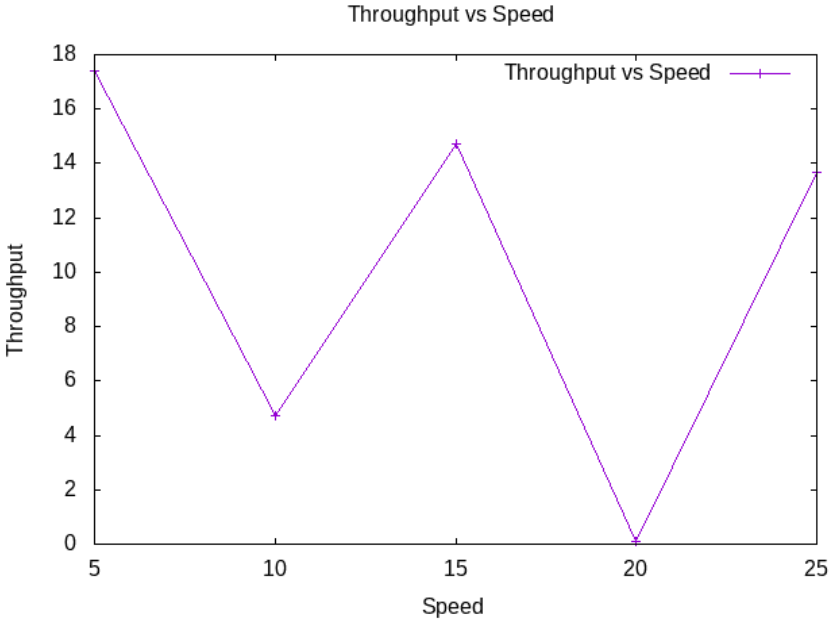


Figure: Wireless low rate throughput vs speed graph

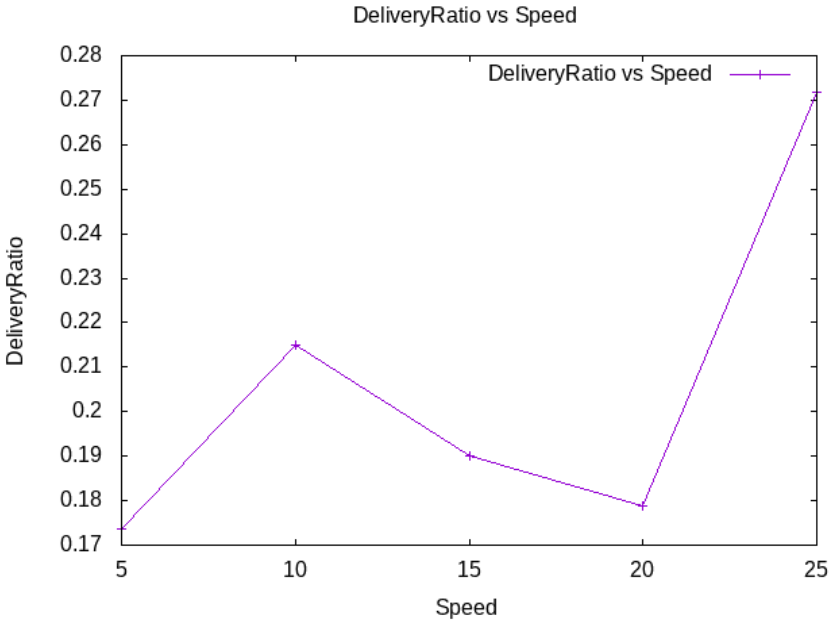


Figure: Wireless low rate delivery ratio vs speed graph

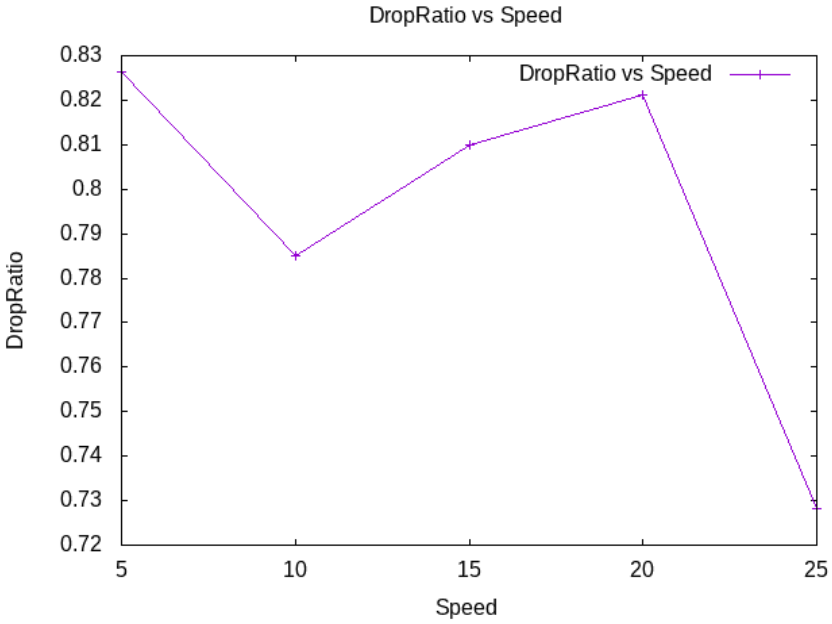


Figure: Wireless low rate drop ratio vs speed graph

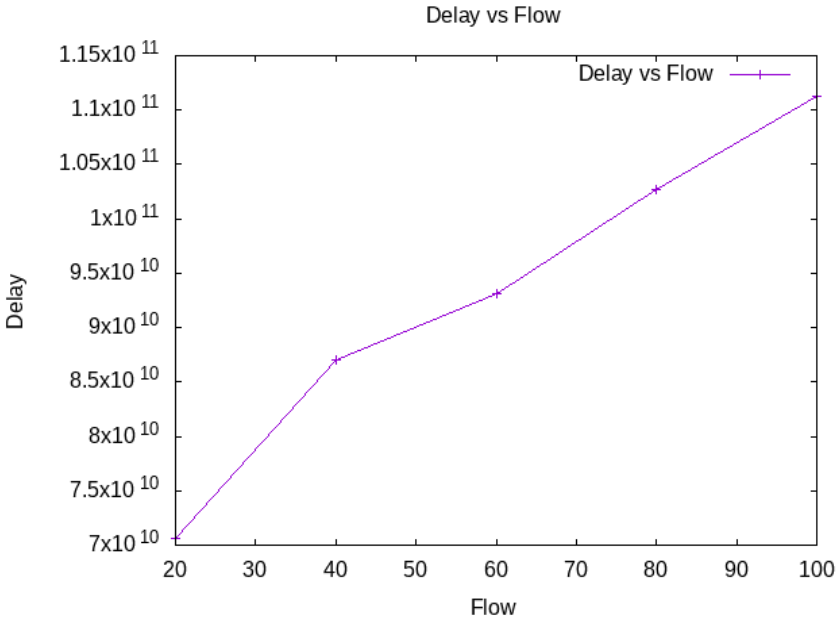


Figure: Wireless low rate delay vs flow graph

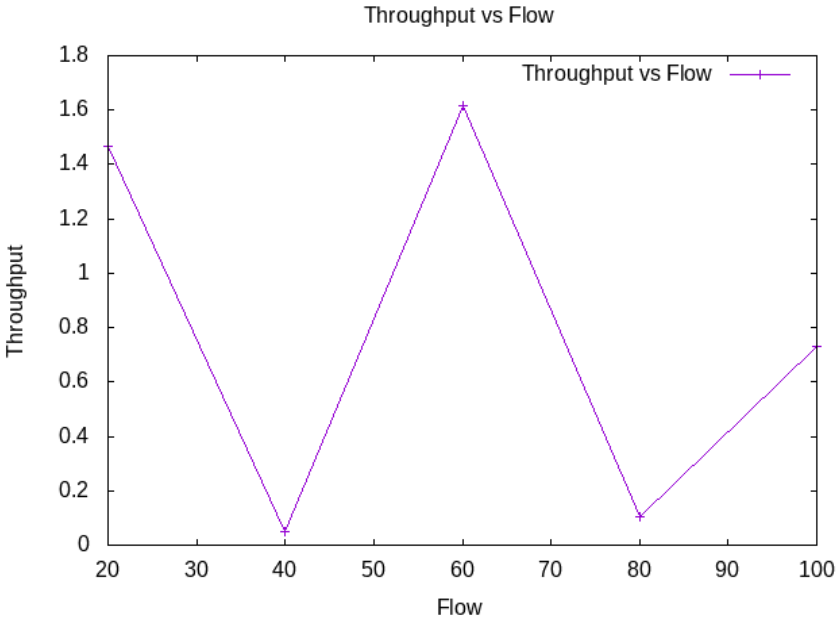


Figure: Wireless low rate throughput vs flow graph

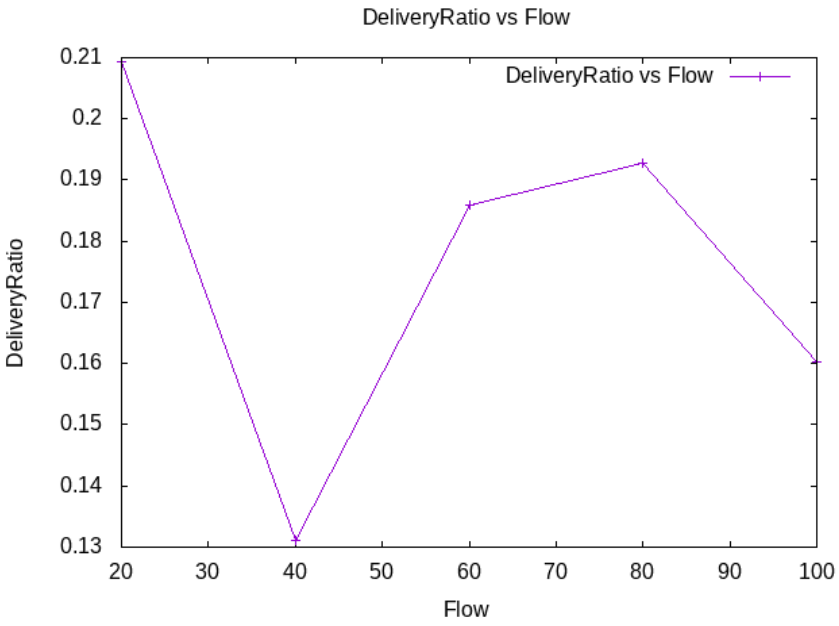


Figure: Wireless low rate delivery ratio vs flow graph

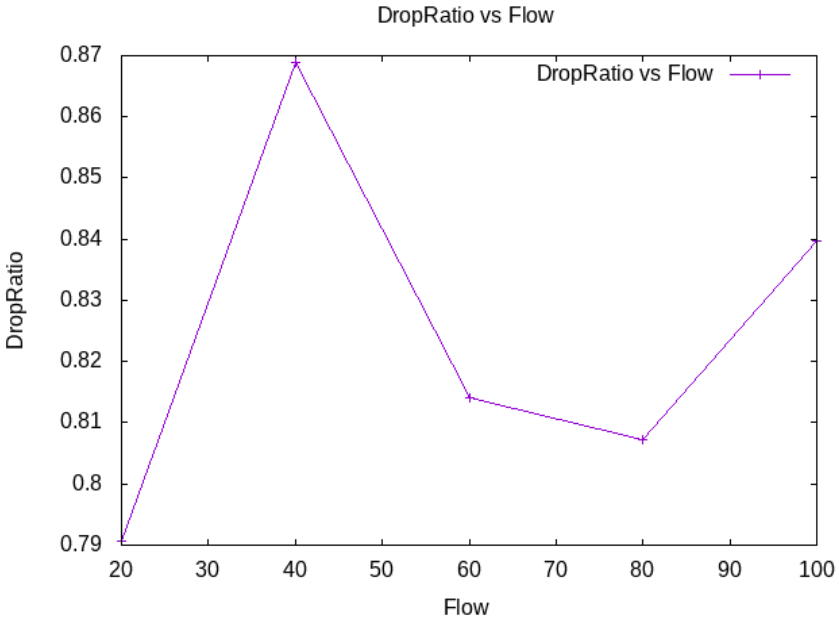


Figure: Wireless low rate drop ratio vs flow graph

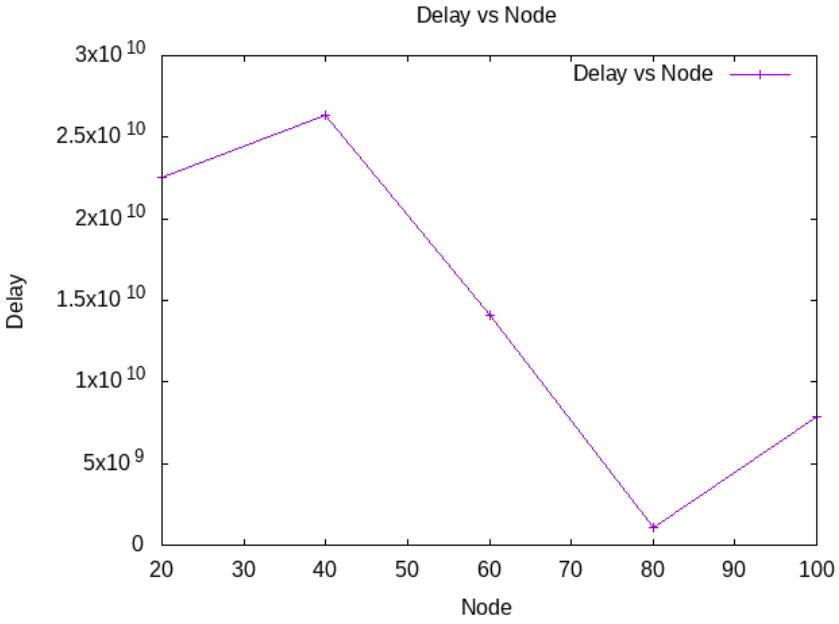


Figure: Wireless low rate delay vs node graph

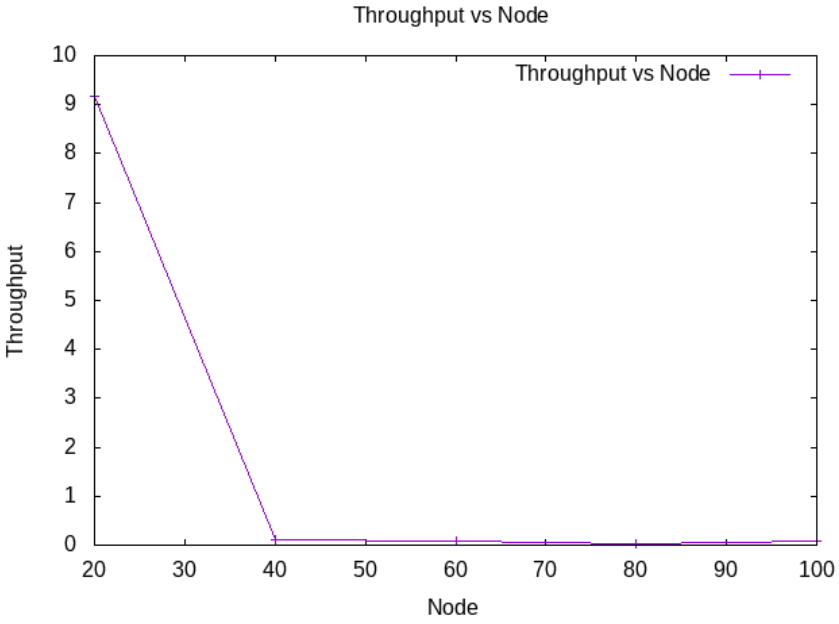


Figure: Wireless low rate throughput vs node graph

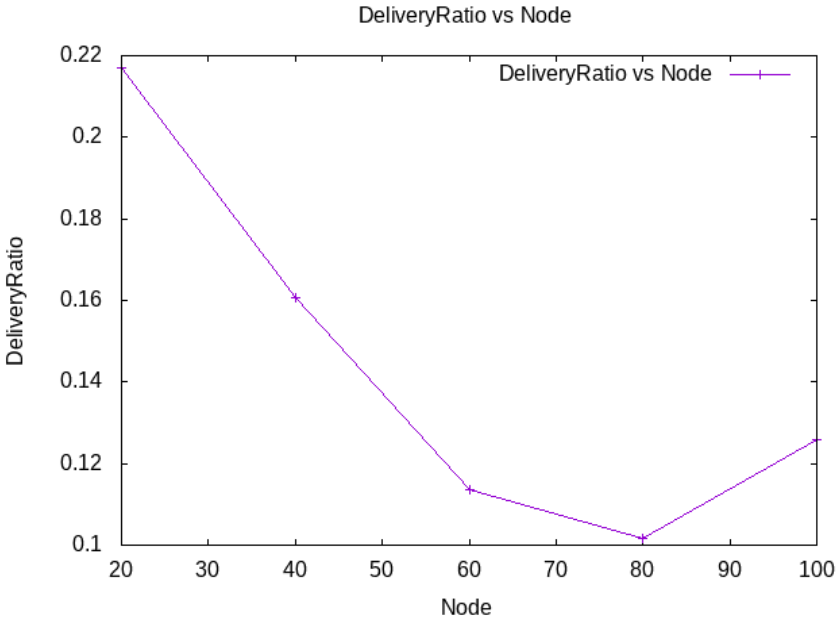


Figure: Wireless low rate delivery ratio vs node graph

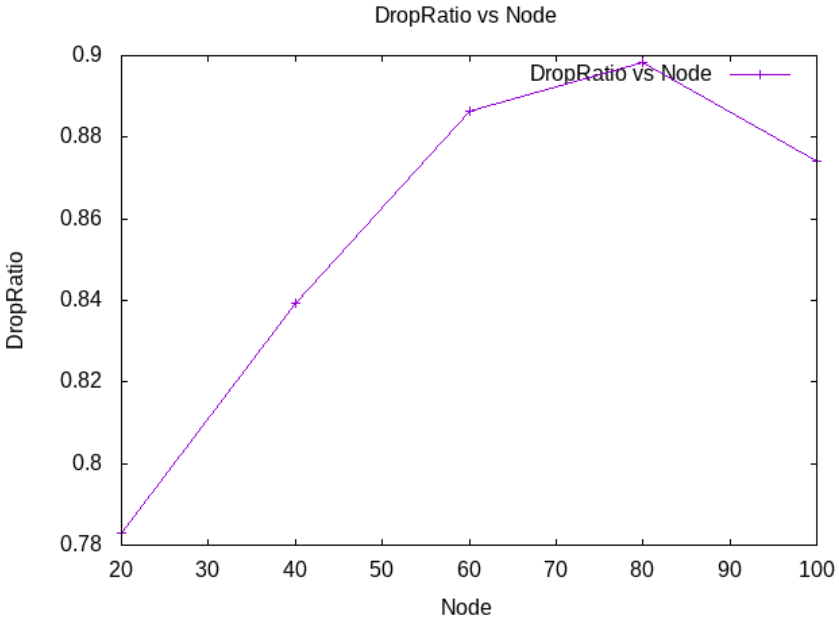


Figure: Wireless low rate drop ratio vs node graph

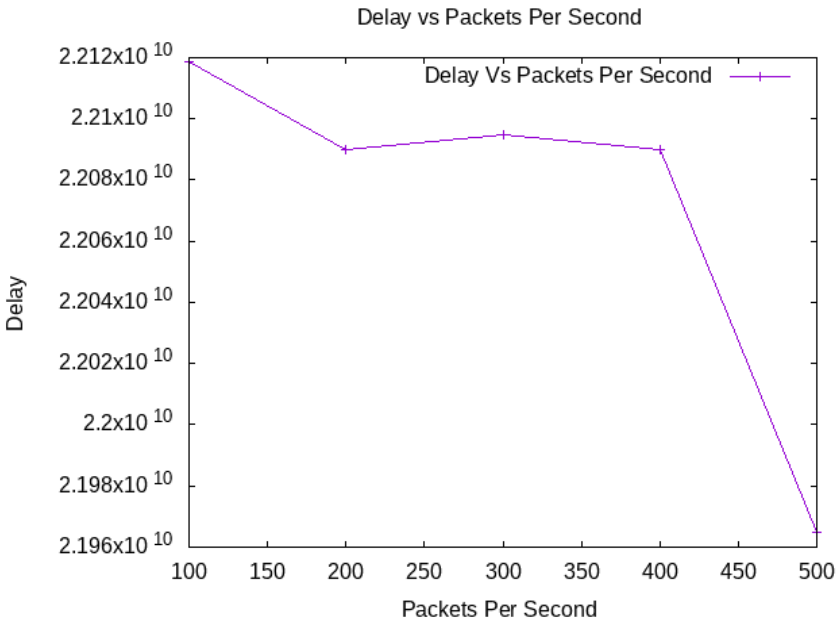


Figure: Wireless low rate delay vs packets per second graph

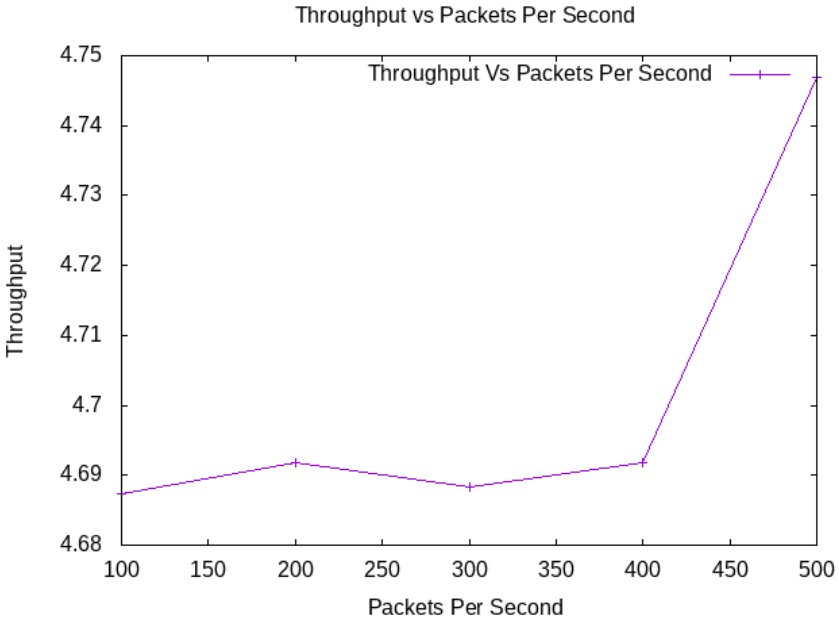


Figure: Wireless low rate throughput vs packets per second graph

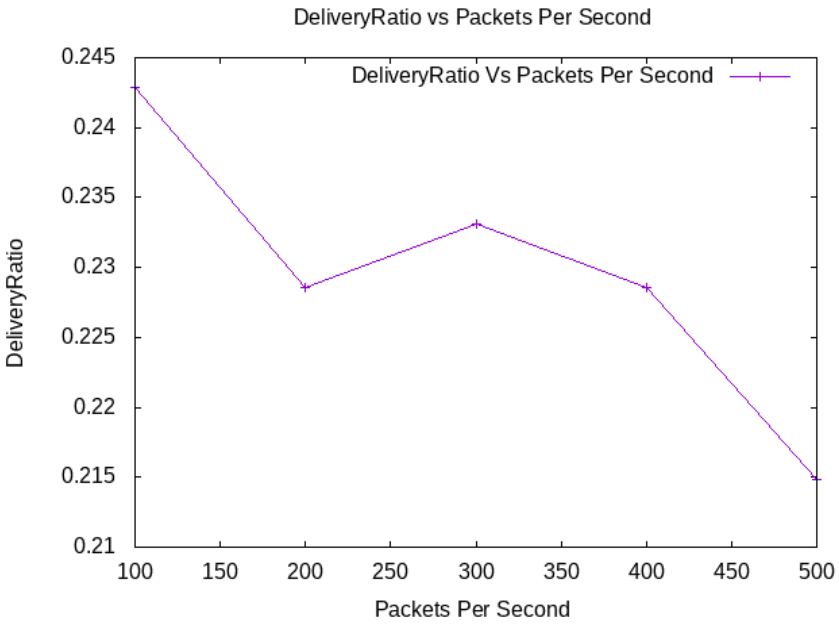


Figure: Wireless low rate delivery ratio vs packets per second graph

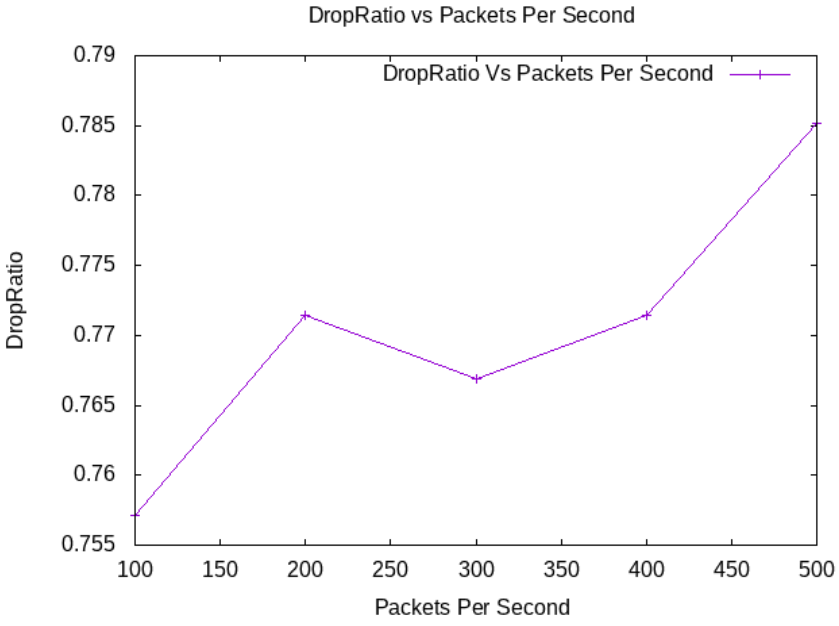


Figure: Wireless low rate drop raio vs packets per second graph

Task B

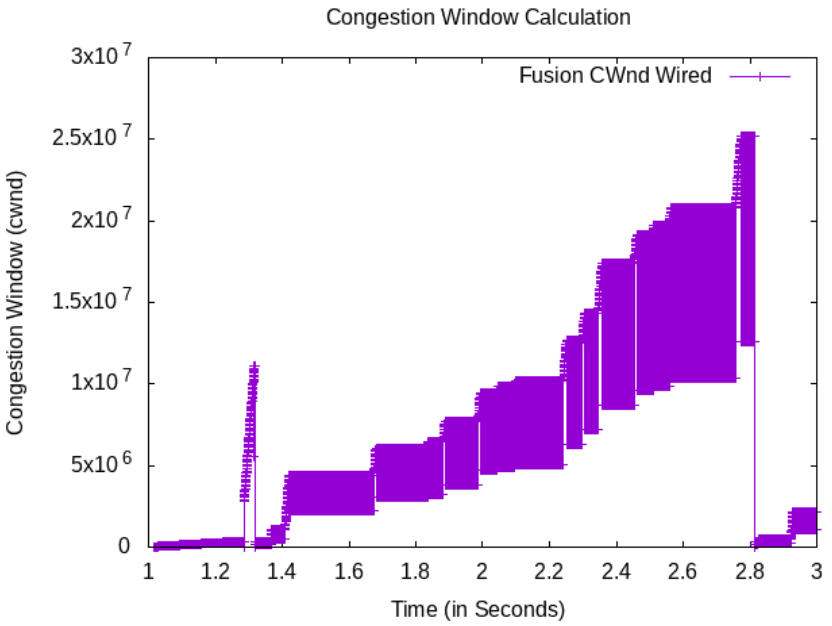


Figure: TCP-Fusion congestion window with error rate 10^{-6}

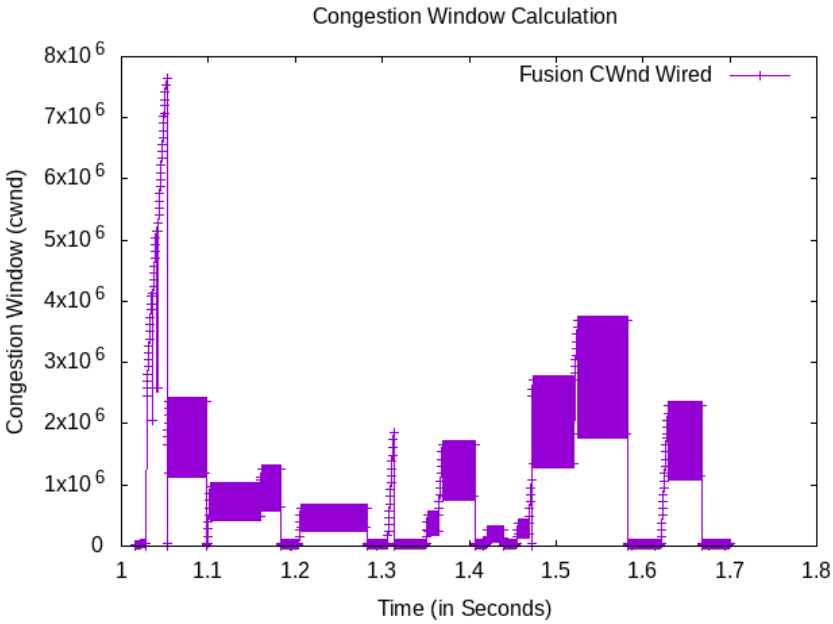


Figure: TCP-Fusion congestion window with error rate 10^{-5}

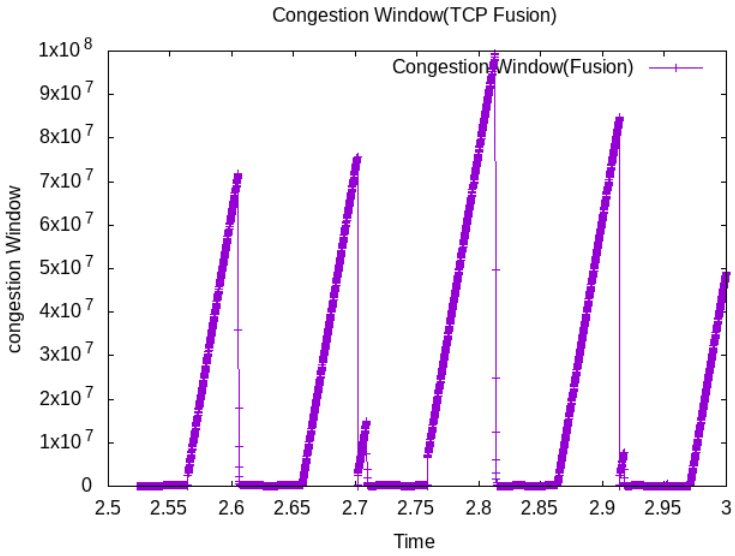


Figure: Congestion window of TCP-Fusion

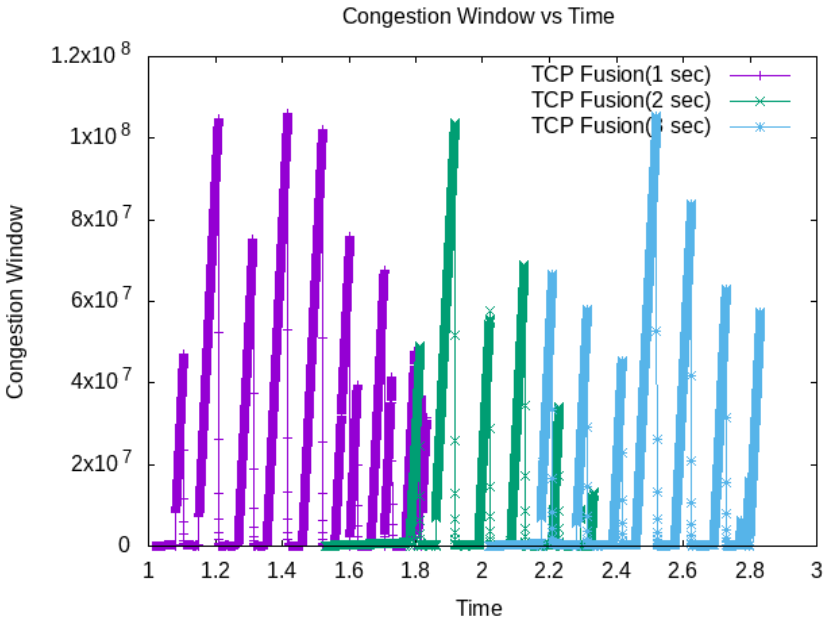


Figure: Multiple flow of TCP-Fusion

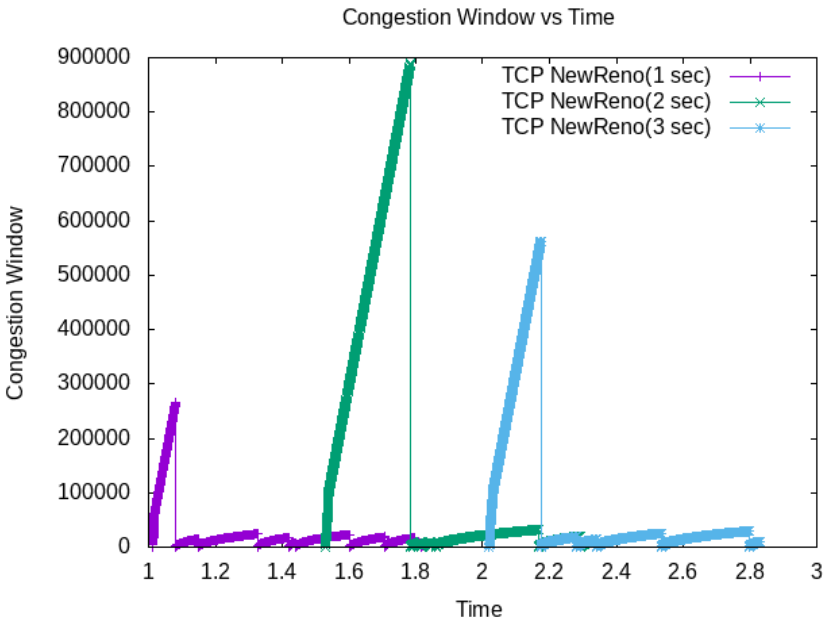


Figure: Multiple flow of TCP-NewReno

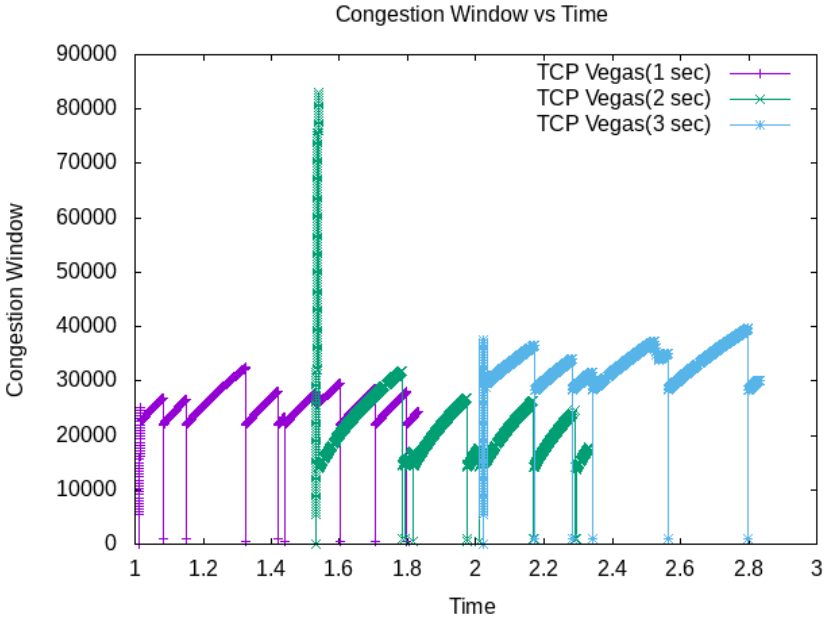


Figure: Multiple flow of TCP-Vegas

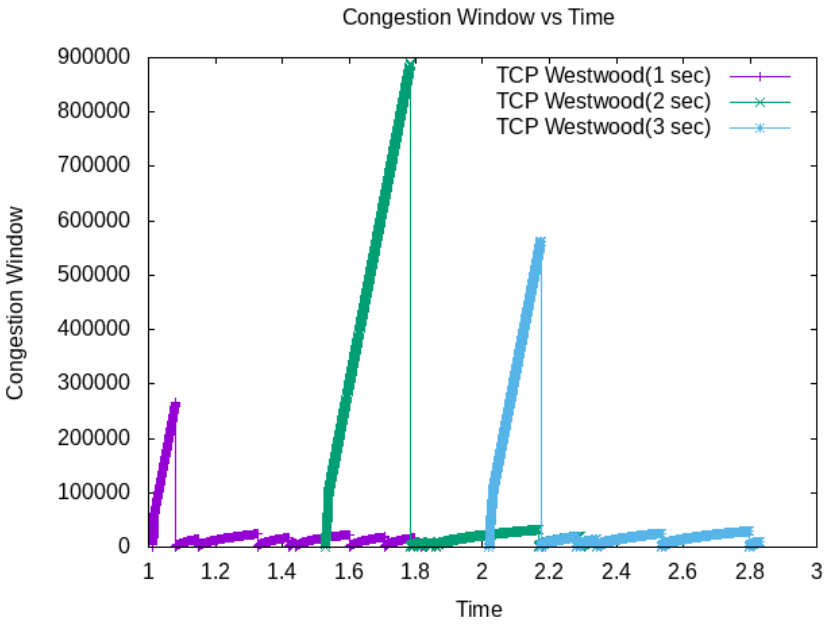


Figure: Multiple flow of TCP-Westwood

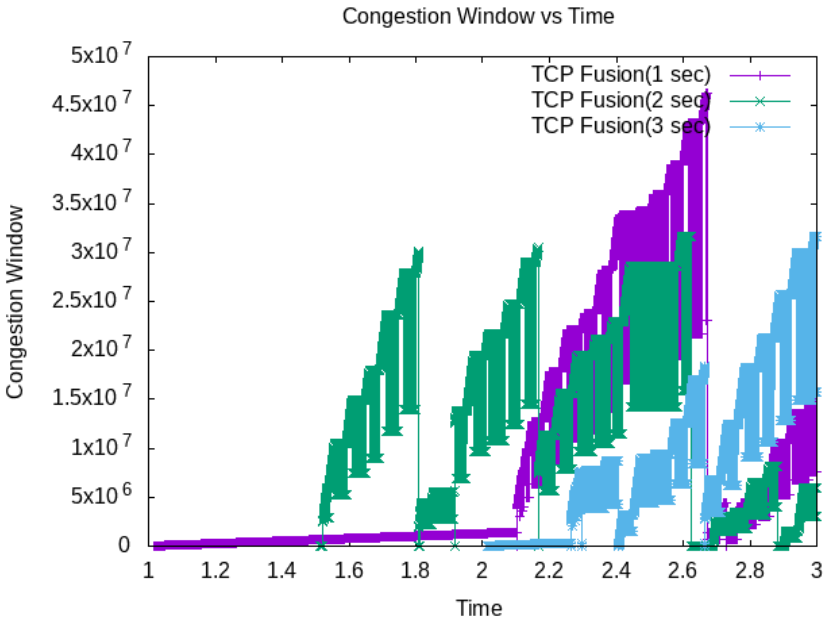


Figure: Multiple flow of TCP-Fusion (modified parameters)

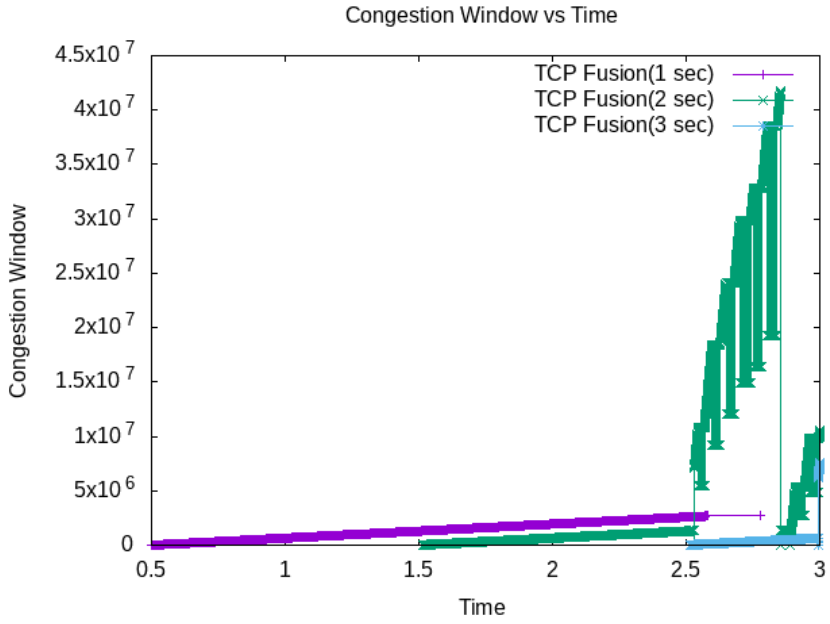


Figure: Multiple flow of TCP-Fusion (modified parameters with high error rate)

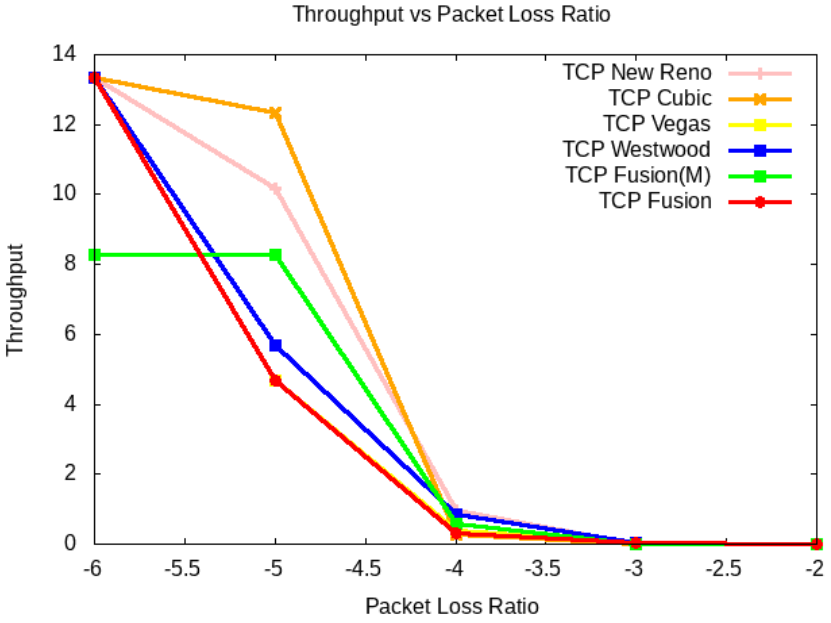


Figure: Throughput vs loss ratio for different TCP congestion control algorithms

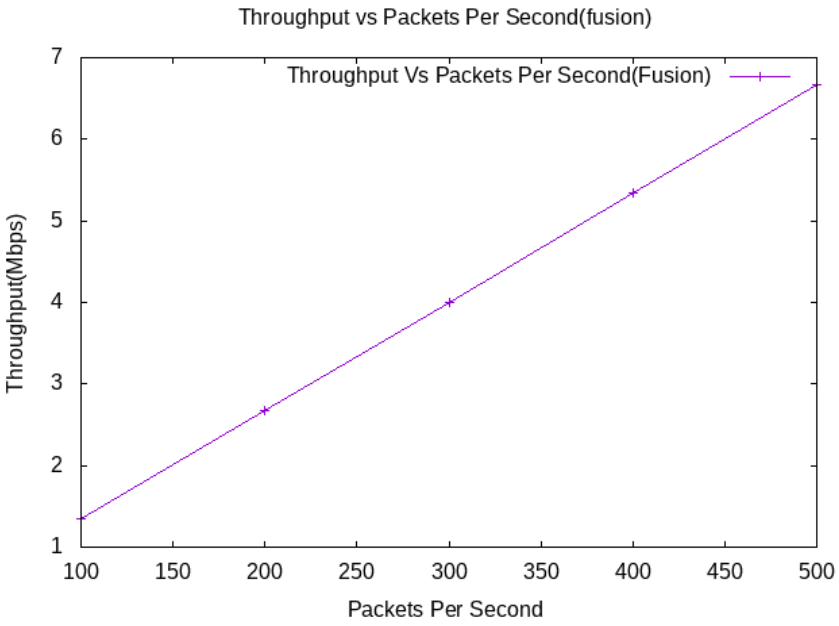


Figure: Throughput vs packets per second for TCP-Fusion

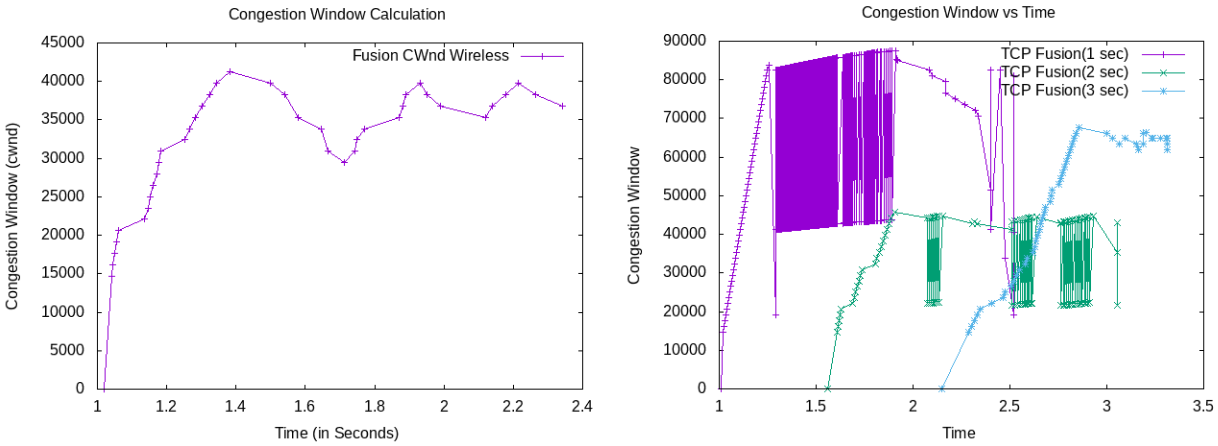


Figure: (Left) Congestion window of TCP-Fusion in wireless network,
(Right) Congestion window of multiple flow in wireless

Summary

Task A

For Task A High rate, coverage vs varying parameter graphs became constant after a certain range. It means that the parameters were the same for nodes in particular range. In flow vs varying parameter graphs, delay and drop ratio were on the increase with the increment of number of flows while delivery ratio and throughput decreased. Congestion of the network due to a higher number of flows may cause this increased number of drop ratio and delays. On the other hand, node vs varying parameter graphs did not follow any particular trend. Packets per second vs varying parameter graphs were similar to those of flow vs varying parameter graphs.

For Task A low rate, speed vs varying parameter graphs delivery ratio and throughput increased with the increment of speed while delay and drop ratio decreased. In flow vs varying parameter graphs, delay increased with the increment of number of flows while delivery ratio, drop ratio and throughput varied a lot. Simulation start time and different parameter setup may cause this. In node vs varying parameter graphs delay, delivery ratio and throughput decreased while drop ratio increased. Because of passing through a higher number of nodes, delay increased. Packets per second vs varying parameter graphs throughput increased while other parameters decreased.

There are many reasons behind these results. Such as, increasing number of nodes causes more traffic and reduces throughput while increasing end-to-end delay. When the number of flow increases, first throughput increases because channel bandwidth is more. After increasing enough flows, the channel becomes saturated and causes more traffic. So, throughput decreases afterwards. Same reason is applicable for increasing the number of packets per second as the number of flows. When coverage area is increased, more nodes can be within the coverage, and connectivity will be stronger. Therefore, increasing coverage area,increases delivery ratio and decreases drop ratio.

Task B

For Task B, varying the algorithm parameters drastically changed the congestion window graph of the TCP Fusion algorithm.The paper I followed assumed that no routers have smaller than G packets that correspond to the queuing delay D_min in the bottleneck queue. The paper has no indication of the bottleneck bandwidth range, It also did not mention about the change in slow start threshold. Here, they did not mention any straightforward process of calculating the parameters and their units. They stated their algorithm vaguely.

In this paper, they presented a new hybrid congestion control algorithm, called TCP-Fusion. They stated that this protocol integrates three characteristics; TCP-Reno, TCP-Vegas and TCP-Westwood, which provides a good balance between efficiency and friendliness. The implementation and simulation results of the paper showed that TCP-Fusion can achieve the highest throughput in existing protocols. Moreover, they stated that a TCP-Fusion flow competes with a TCPReno flow, it can obtain more than fair share when there is unused residual capacity(but no mention of the process of creating channel or topology with unused residual capacity); otherwise, it shares the same bandwidth to coexisting flows(they presented some graph, but no detailed description was mentioned . They also emphasized that the fairness among TCP-Fusion flows is almost the same as that of TCP-Reno, of which I did not find any practical proof.

So, I took help from other congestion control algorithms and varied the parameters and equations a bit. They stated that the algorithm performs well in lossy links, but no description of that link was mentioned. They also stated that the algorithm works well in leaky pipe with large bandwidth delay products and non negligible random losses, but no topology or network was suggested. After several modifications, I got a reasonable graph of the congestion window, multiple flow of congestion windows and throughput vs drop ratio. Though the result graphs did not have 100% similarity with those mentioned in the paper, they were good enough.