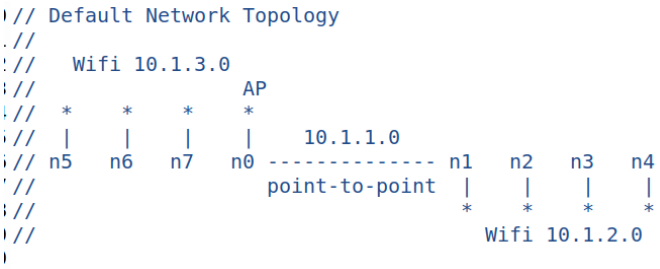
CSE316 : Computer Networks Sessional

Project Name: TCP Westwood With Agile Probing

Network Topology



This topology has been used to simulate Task A for mobility model and Task B. In this topology, there are 3 different networks. There are two different networks in both side and there is a point to point network between two access point node of both wireless networks.

Overview of the Project:

The key idea of TCPW-A is to use Eligible Rate Estimation (ERE) to set the congestion window and slow start threshold (ssthresh) after a packet loss. Instead of multiplicative decrease of ssthresh when a packet loss occurs, TCPW-A set the ssthresh and congestion window according to ERE. TCPW-A achieves its goal by incorporationg the following two mechanisms :

1. In slow start stage, TCPW-A continuously resets ssthresh and cwnd based on the ERE.
2. In congestion avoidance stage, upon detecting non congestion it resets ssthresh and cwnd according to the ERE instead of linear increase.

ERE sample is calculated based on the amount of data in bytes that were successfully delivered in a certain time interval (T_k). Agile Probing uses ERE to adaptively and repeatedly reset ssthresh. During Agile Probing, when the current ssthresh is lower than ERE, the sender resets ssthresh higher accordingly, and

increases cwnd exponentially. Otherwise, cwnd increases linearly to avoid overflow.

In congestion avoidance stage, instead of increasing congestion window linearly TCPW-A uses a scheme called Persistent Non Congestion Detection (PNCD) which detects non congestion in the network. Upon detecting Non Congestion it restarts the agile probing again to reset the ssthresh and congestion window accordingly.

Modifications made in the simulator

1. **ERE calculation :** In TCPW-A, ssthresh and cwnd are reset according to Eligible Rate Estimate. To calculate this ERE, a function CalculateERE() has been included and scheduled in every Tk interval.

```
void
TcpWestwoodWithAgileProbing::CalculateERE()
{
  uint32_t ackedSegs;
  if (ackedSegments - segments > 0)
    ackedSegs = ackedSegments - segments;
  ERE = ackedSegs * segment_size / Tk;
  Time now = Simulator::Now();
  segments = ackedSegments;
  Simulator::Schedule(Seconds(Tk),
                      &TcpWestwoodWithAgileProbing::CalculateERE, this);
}
```

2. **AgileProbing**

In slow start stage and congestion avoidance stage, Agile probing increases ssthresh and cwnd according to ERE. A function AgileProbing has been included in both slow start and congestion avoidance funnction.

```
void
TcpWestwoodWithAgileProbing::Agileprobing(Ptr<TcpSocketState> tcb)
{
  TracedValue<uint32_t> thresh = ERE * m_minRtt.GetSeconds() / tcb->m_segmentSize;

  tcb->m_ssThresh = std::max(tcb->m_ssThresh, thresh);

  if (tcb->m_cWnd >= tcb->m_ssThresh)
  {
    std::cout << " agile congestion window >= ssthresh" << std::endl;
    tcb->m_cWnd += 1;
  }
  else
  {
    std::cout << " agile congestion window < ssthresh" << std::endl;
    tcb->m_cWnd += tcb->m_segmentSize;
  }
  restartAP = 0;
}
```

3. Slow start

In slow start function Agile probing has been added

```
uint32_t
TcpWestwoodWithAgileProbing::SlowStart(Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
  if (segmentsAcked >= 1)
  {
    Agileprobing(tcb);
    return segmentsAcked - 1;
  }
  return 0;
}
```

4. Congestion Avoidance

In congestion avoidance function, Persistent Non Congestion Detection function has been included.

```
void
TcpWestwoodWithAgileProbing::CongestionAvoidance(Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
  NS_LOG_FUNCTION(this << tcb << segmentsAcked);

  if (segmentsAcked > 0)
  {
    PersistentNonCongestionDetection(tcb);
    if (restartAP == 1)
    {
      double adder = static_cast<double>(tcb->m_segmentSize * tcb->m_segmentSize) / tcb->m_cWnd.Get();
      adder = std::max(1.0, adder);
      tcb->m_cWnd += tcb->m_segmentSize;
      NS_LOG_INFO("In CongAvoid, updated to cwnd " << tcb->m_cWnd << " ssthresh " << tcb->m_ssThresh);
    }
  }
```

5. **Persistent Non Congestion Detection**

This function maintains a no_congestion_counter to detect non congestion in the network. Upon a no congestion condition it invokes agile probing.

6. **Tk (Time Interval) Calculation**

CalculateERE function is invoked in every Tk time interval which has been calculated when packets has been acknowledged in PktAcked function.

Parameters under Variations

- Number of nodes (20,40,60,80,100)
- Number of Flow (10,20,30,40,50)
- Packets per second (100,200,300,400,500)
- Velocity for mobility (5,10,15,20,25m/s)
- Coverage area for static nodes (10,20,30,40,50)

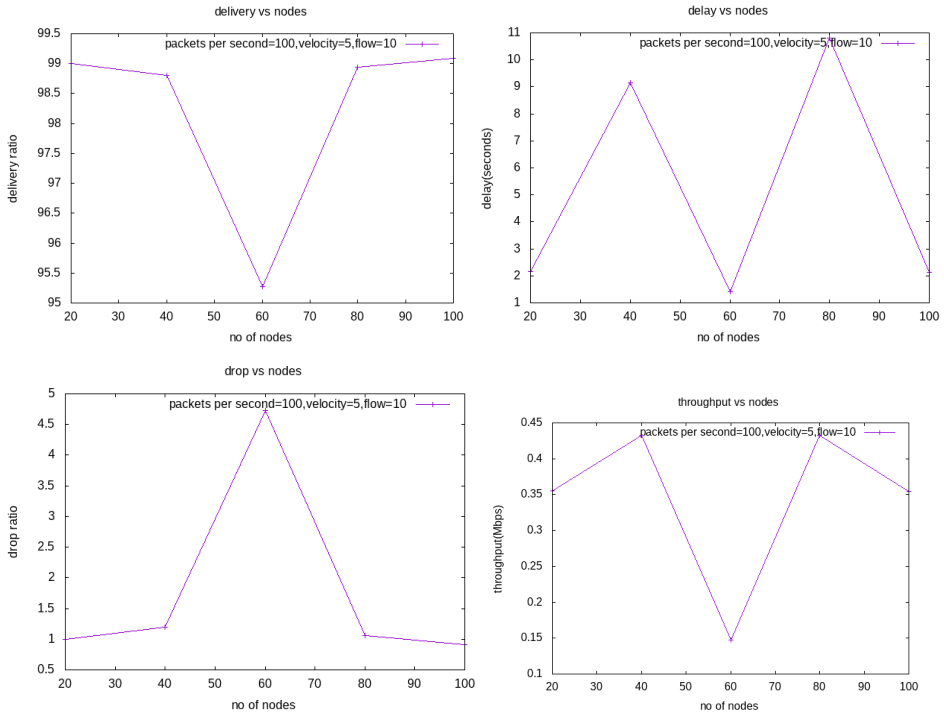
Metrics

- Throughput
- End to End Delat
- Packet Drop ratio
- Packet Delivery Ratio

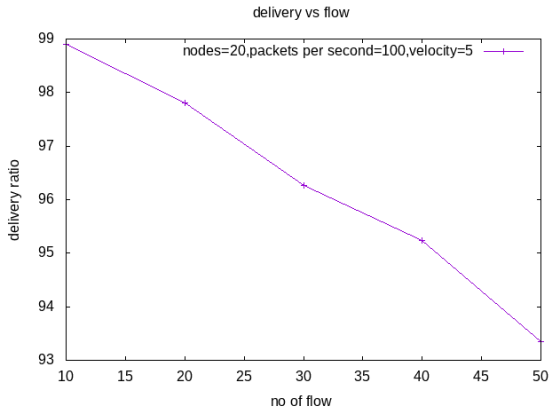
Results with Graph for Task A

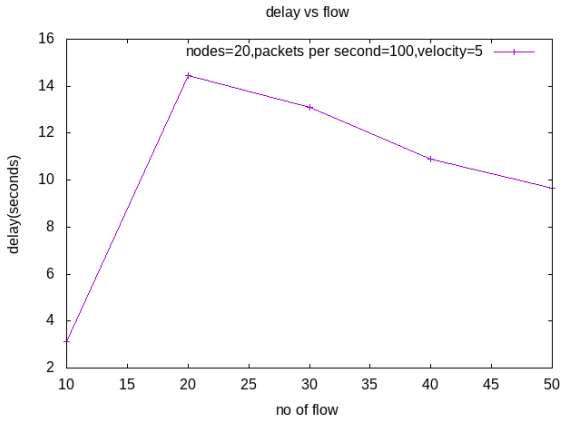
1. Wireless High-rate (Mobile)

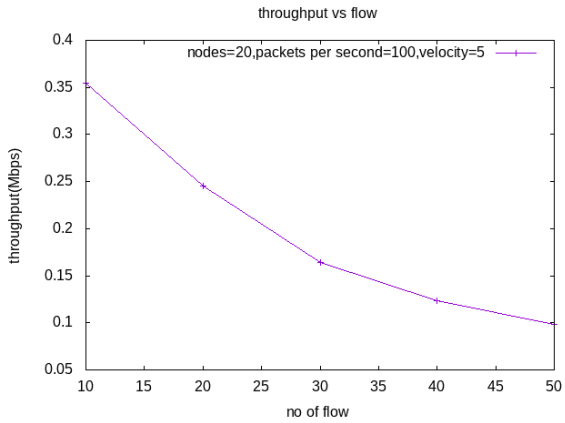
- **Measuring metrics varying No of Nodes:** While simulating varying nodes, there are found inconsistent changes in each metric.

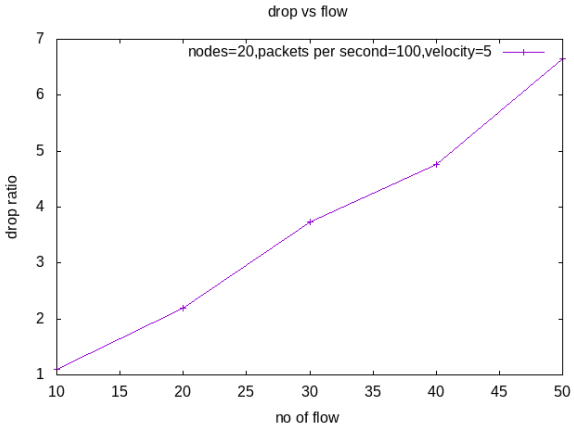


- **Measuring metrics varying No of Flow:** With the increase in flow throughput and delivery ratio decreases. Because as flow increases congestion in network increases. Hence, successful packet delivery per second decreases. For this reason, packet drop increases as congestion in network increases. But in case of delay no consistent change.

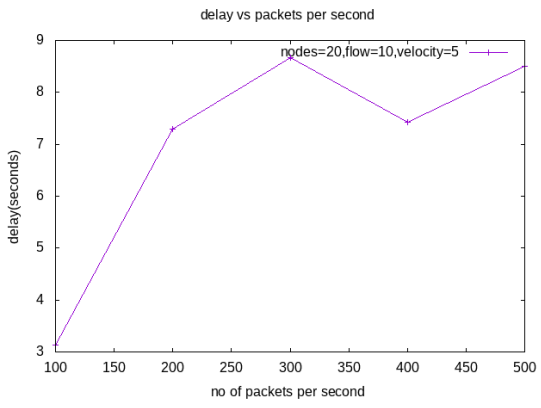


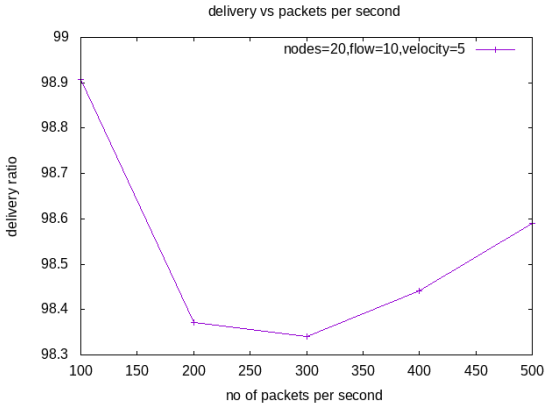


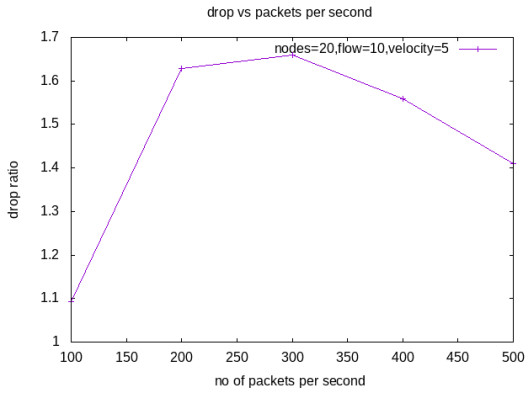


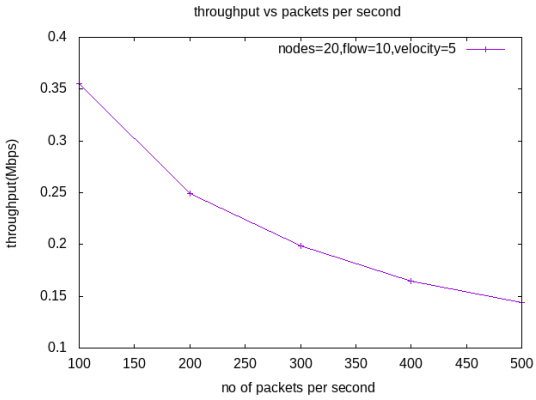


- **Measuring metrics varying packets per second:**

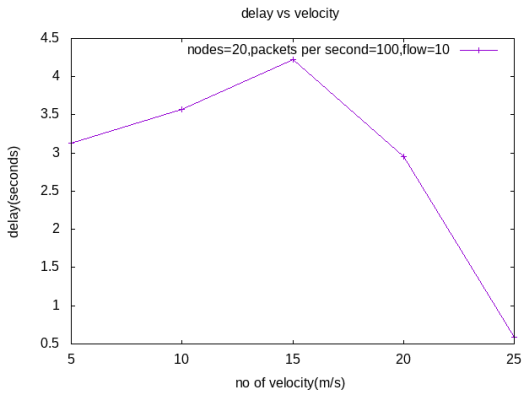


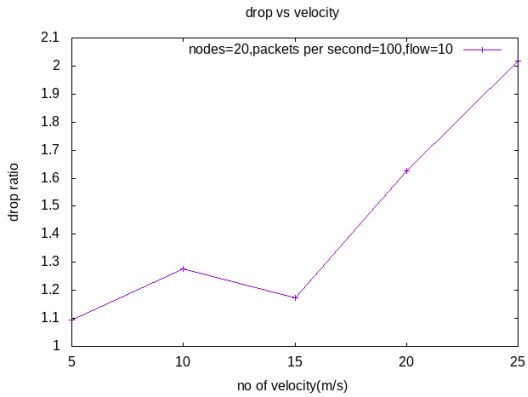


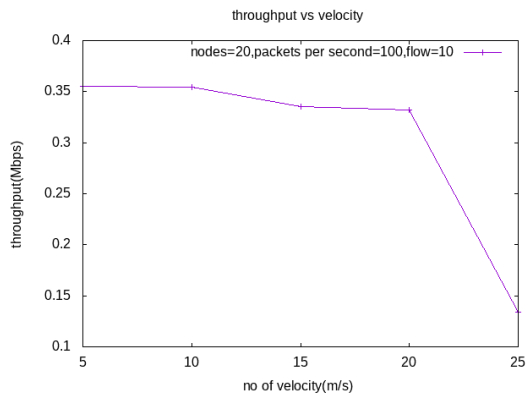




- **Measuring metrics varying velocity of nodes :**



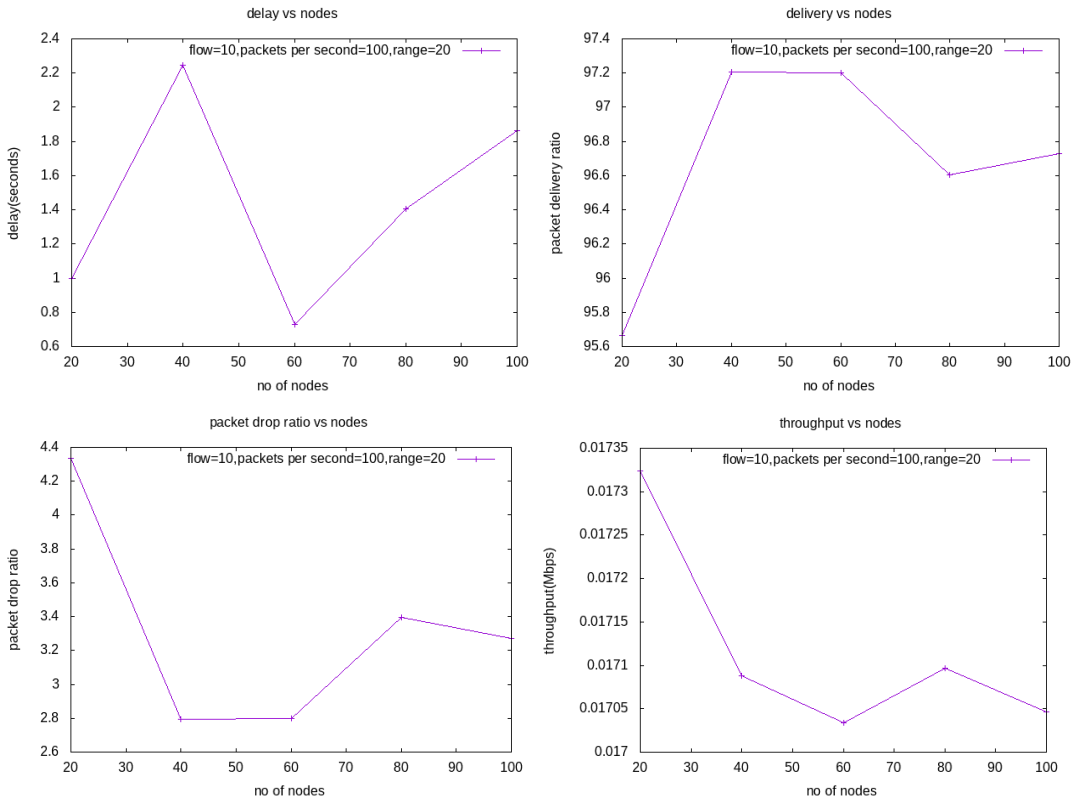




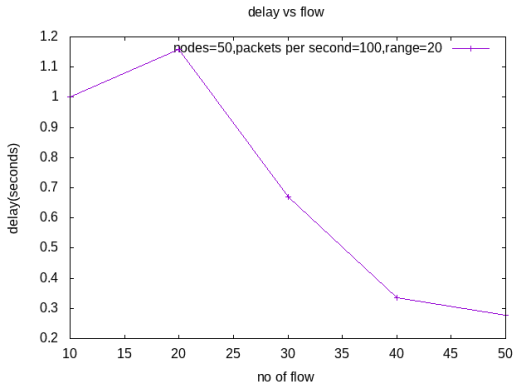
2. Wireless Low-rate (Static)

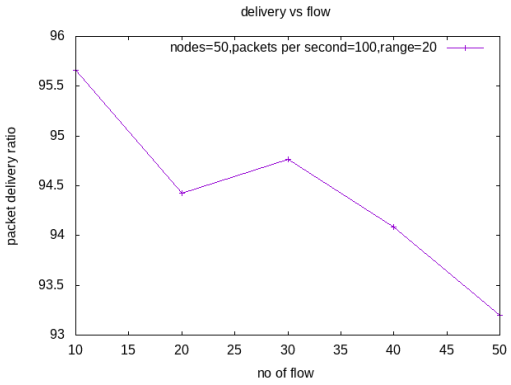
Wireless low-rate has been implemented using Lr-Wpan.

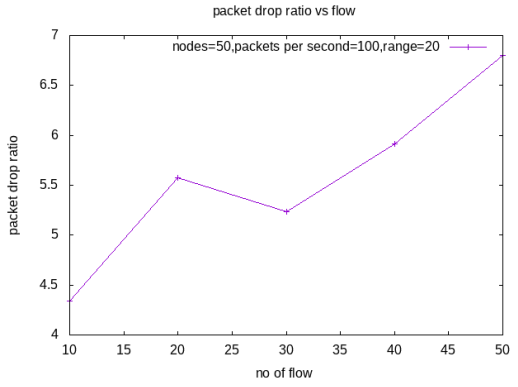
- **Measuring metrics varying no of nodes**

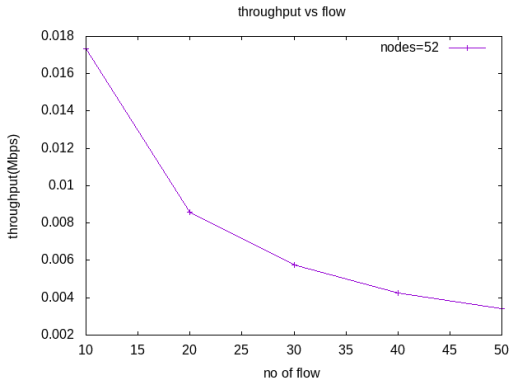


- **Measuring metrics varying no of flow**

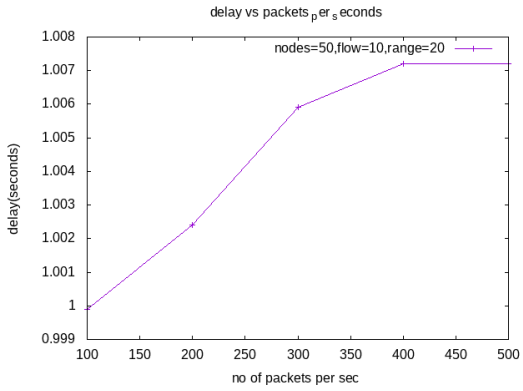


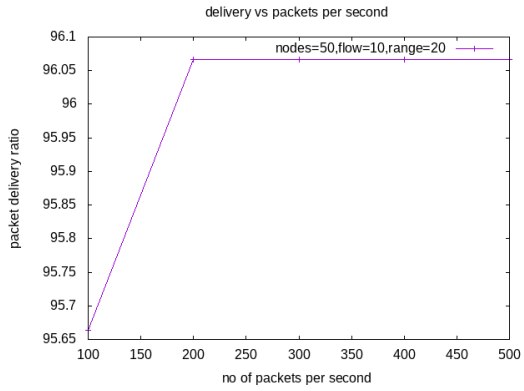


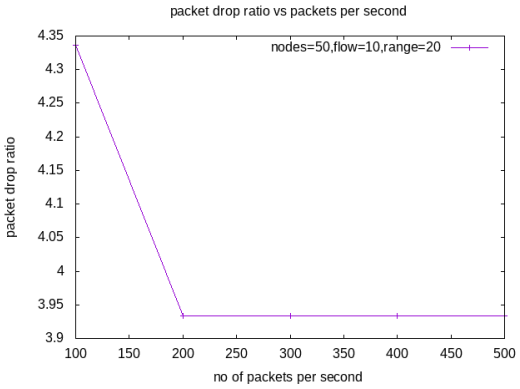


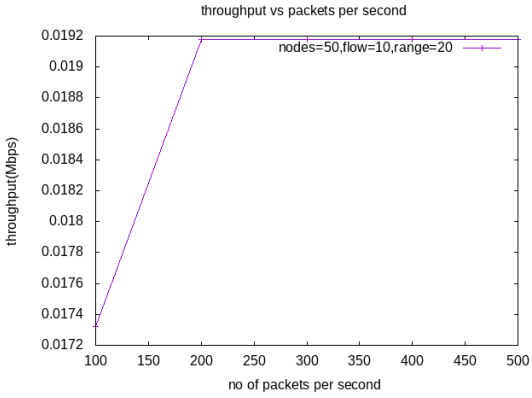


- **Measuring metrics varying packets per second**

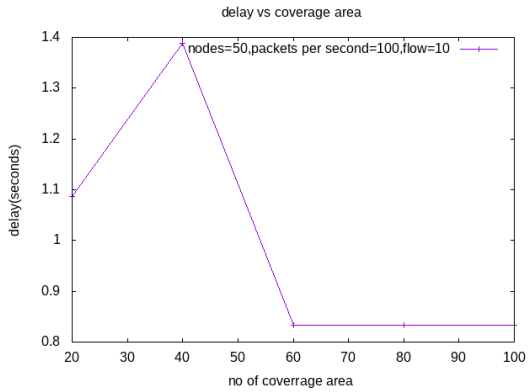


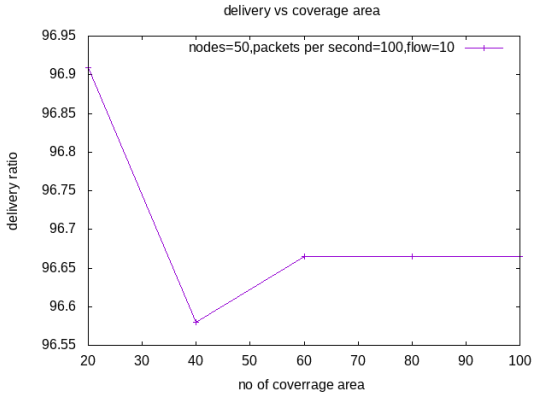


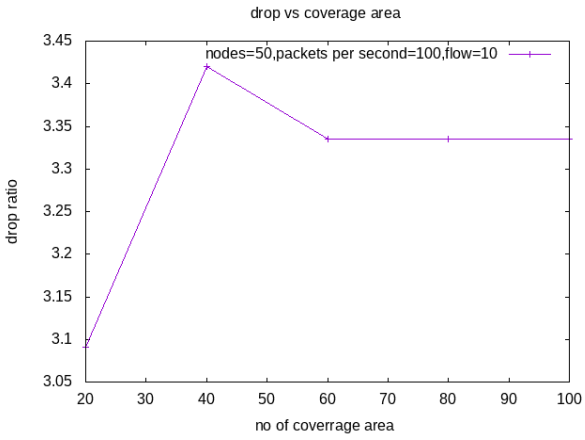


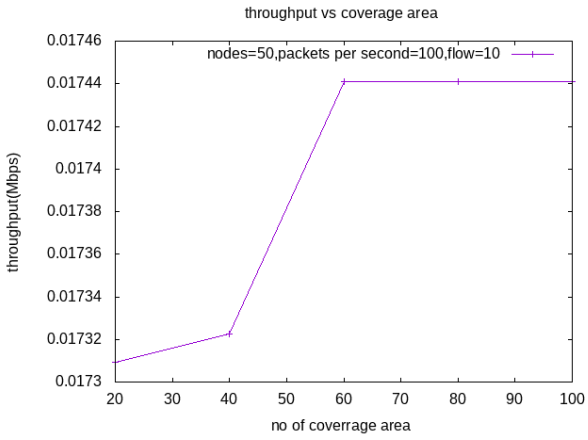


- **Measuring metrics varying Coverage Area**









Explanation of Task A results

In Task A different graph has been plotted for varying different parameters while keeping other parameters fixed.

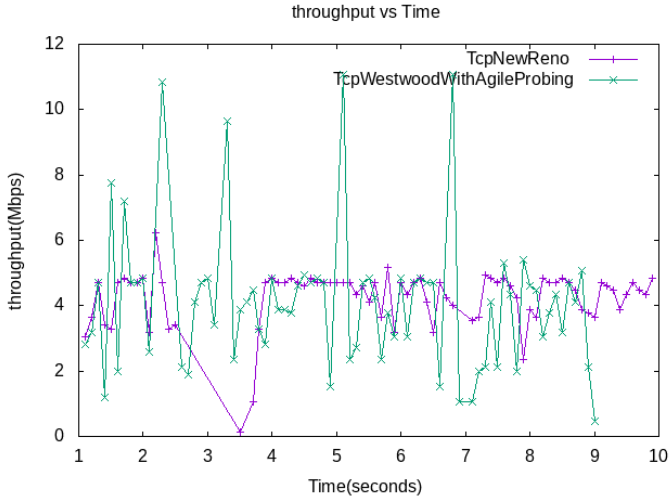
- As the number of nodes in simulations are changed throughput moderately decreases. the throughput of network traffic decreases with an increase in the number of nodes between the source and the destination because the packet error probability increases with an increase in the number of nodes, causing network throughput to decrease.
- As the number of flows increases throughput, packet delivery ratio decreases resulting in more packets drop ratio.
- Initially as velocity increases, throughput decreases. The increase in the speed of the nodes decreases both end-to-end delay and packet delivery ratio. Because node velocity results in route failures and decreases packet delivery results in more packet drops.
- As packets per second increase throughput and packet delivery ratio decreases. Because the larger the packets per second, the more probability of packet loss resulting in more packets drop and decreases throughput.
- As the coverage area increases node density decreases which increases throughput.

Simulation using proposed Algorithms

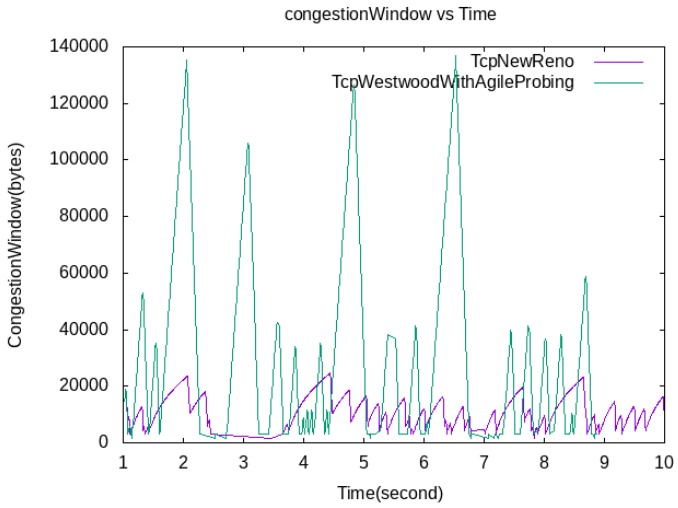
In this project TcpNewReno has been considered as baseline alogorithm.

Comparison of TcpNewReno and TcpWestwood with Agile Probing

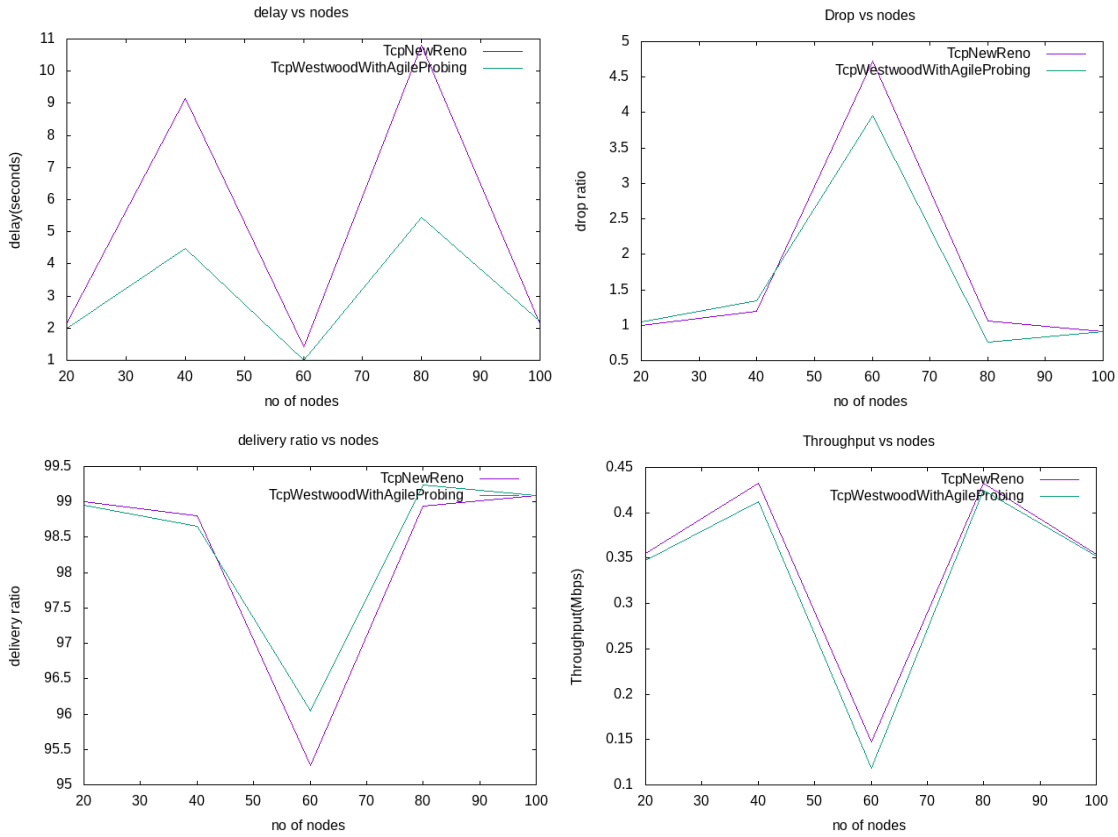
- **Throughput Comparison Graph**



- **Congestion Window Comparison Graph**

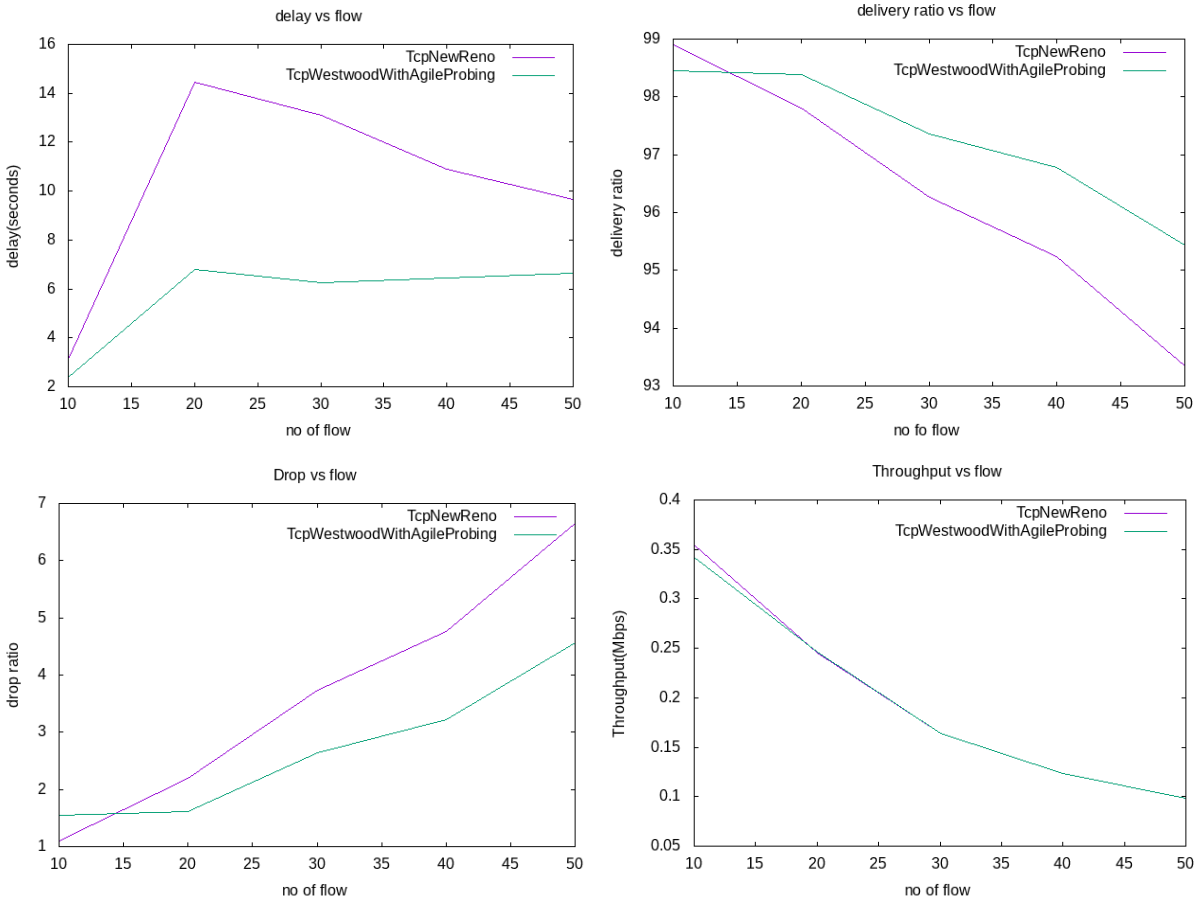


- **Comparison Graph varying numbers of nodes**



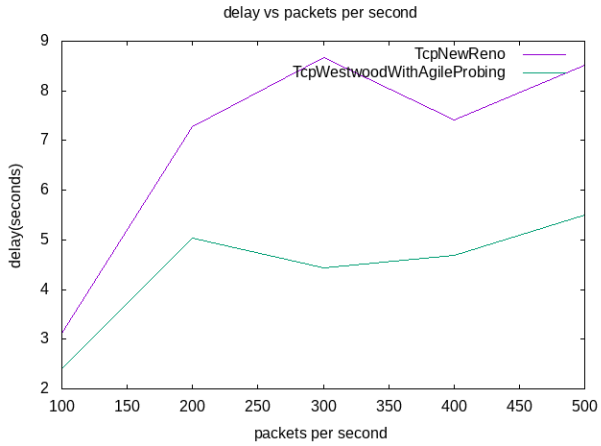
Since Congestion window increases using agile probing which resets ssthresh and cwnd according to eligible estimation rate cwnd is larger than in TcpNewReno resulting in decreasing end to end delay.

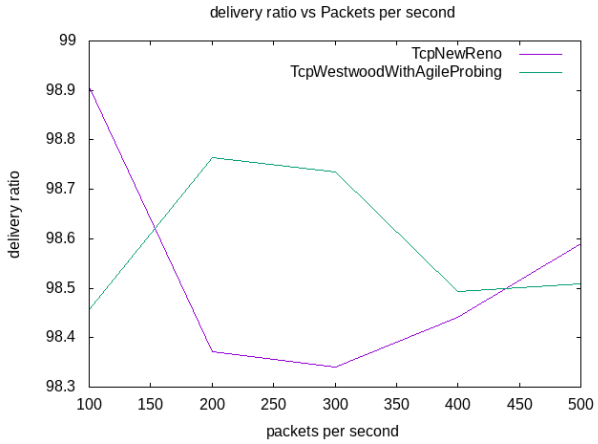
- **Comparison Graph varying numbers of flows**

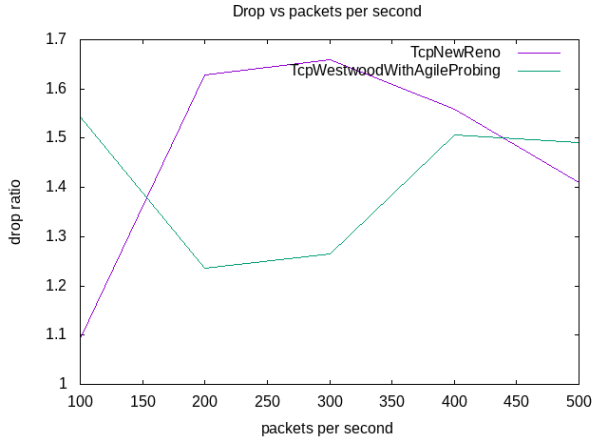


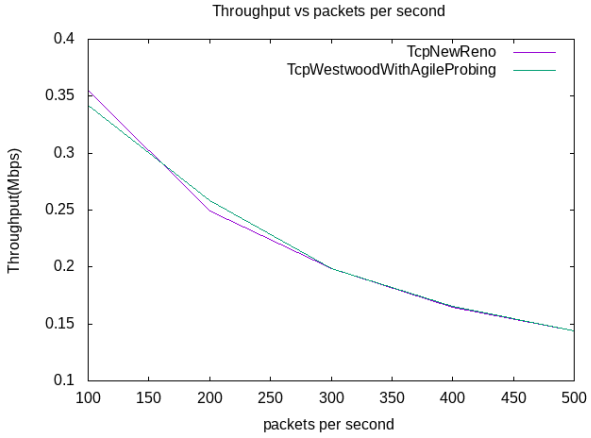
As flows increases algorithm shows better result in less packet drop ratio and in decreasing delay.

- **Comparison Graph varying packets per second**









Conclusion

TcpWestwood with agile probing has claimed better congestion window in both slow start and congestion avoidance stages by resetting the ssthresh and cwnd. This algorithm proposed a moderate decrease in congestion window upon a packet loss. In TcpWestwood, upon a packet loss cwnd is reset according to the value of ERE but TcpWestwood with agile probing approaches for better utilization of the ERE by repeatedly updating ssthresh accordingly.

According to the proposed algorithm, it was supposed to increase the network throughput better than TcpNewReno. Since congestion window is larger end to end delay also decreases.

While implementation there was inaccuracy in calculating ERE and T_k time interval which deteriorated the overall performance in simulation. But it still ensures decrease in end to end delay and larger congestion window in congestion avoidance stage.