CSE 322 Project Report

Muhtasim Noor Alif

1705108

1 Introduction

The term project for CSE322 Computer Networks Sessional course is divided into two Tasks. Task A is basic simulation and calculation of different metrics on different selected networks. Task A comprises of two parts. Since my student ID is 1705108, the selected networks for me was :

- Wireless high-rate (static)
- Wireless low-rate (mobile)

In Task B, any algorithmic level modification was required. For Task B, I selected a paper titled *"Vegas-W: An Enhanced TCP-Vegas for Wireless Ad Hoc Networks"*, which is an improvement on TCP-Vegas for ad hoc networks, to implement on the existing TCP-Vegas on NS3.

2 Task A

2.1 Wireless high-rate (static)

2.1.1 Topology

The topology was similar to Mesh. The nodes were placed in a grid maintaining a fixed distance from one another (x and y axis). The distance between the nodes were kept variable in the simulation. The number of sinks were also variable. The sources and sinks were made pairwise, meaning one source is attached to a single sink and vice versa. *nWifi* denotes the number of total nodes and *nSinks* denotes the number of sinks/sources. Actually, two source were created in this simulation for *nSinks=1*. This is because, when one source application was installed on one source node to send packets to one sink node, another source app was also installed on the corresponding sink node to send packets to the previous source node (These are not acknowledgement packets). Simply to say, two nodes in a source sink pair both sends data packets to one another in this simulation, they both act as a source and a sink. In this topology, the first and last nodes are made pairs, then the second and the second before last node, and so on.

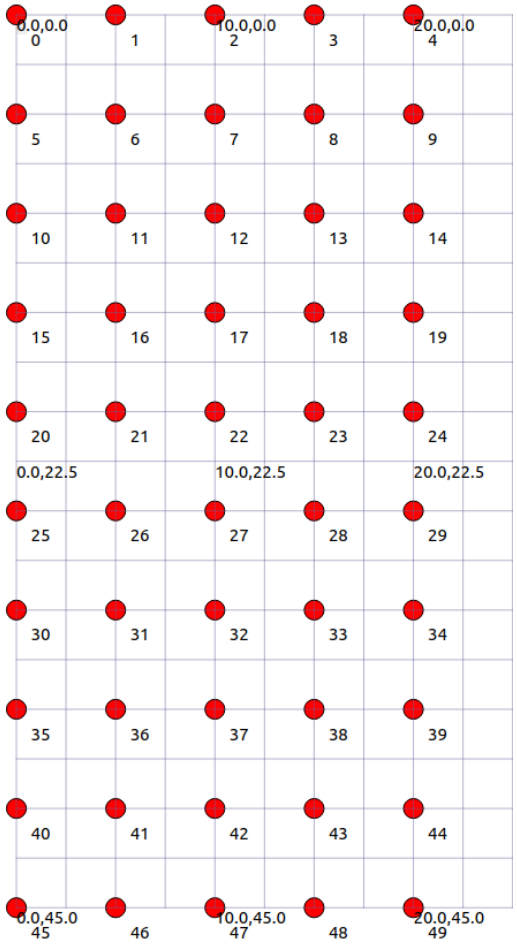


Figure 1: 50 Nodes in Grid

2.1.2 Variation of Parameters

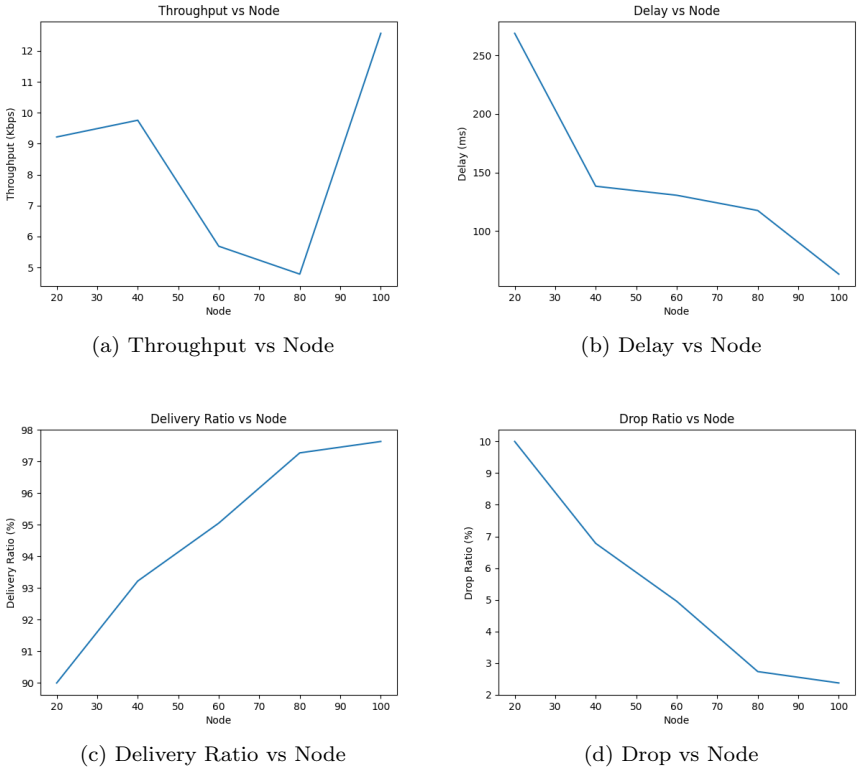
In the simulation, various parameters were tweaked to observe the network performance. Various metrics were recorded to understand the results of the parameters being varied. In the simulation, the parameters listed below were varied:

- Number of Nodes
- Number of Flows
- Number of Packets per Second
- Coverage Area

Besides these, parameters like packet size, error rate of physical channel, total simulation time, etc were kept variable for user convenience.

2.1.3 Metric Measurement and Graphs

Tweaking the parameters stated above, metrics like Network Throughput, End to End Delay, Packet Delivery Ratio and Packet Drop Ratio were observed. The average metric of all the flows were recorded here, as they provide an overall performance of the network.



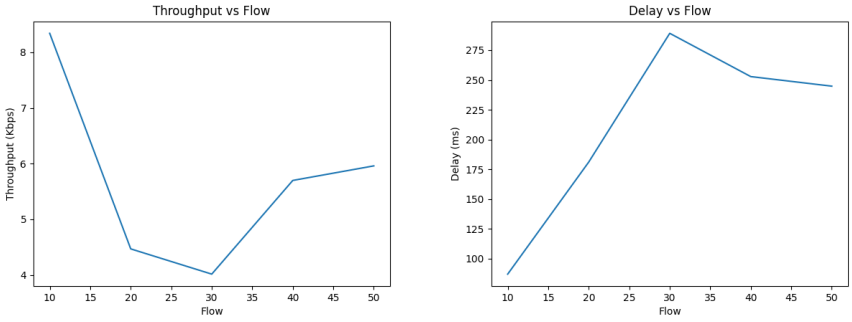
(a) Throughput vs Node

(b) Delay vs Node

(c) Delivery Ratio vs Node

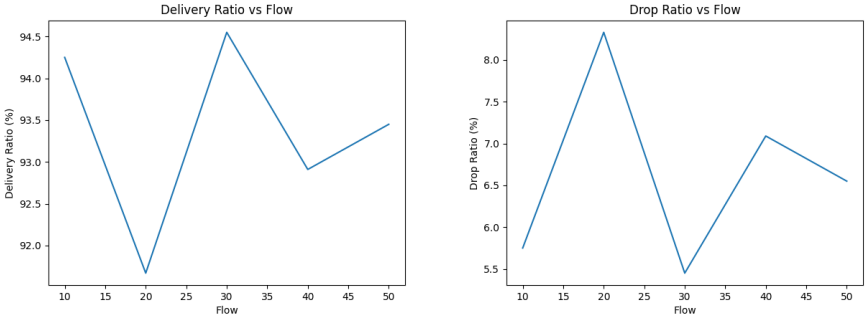
(d) Drop vs Node

Figure 2: Metrics vs Number of Nodes



(a) Throughput vs Flow

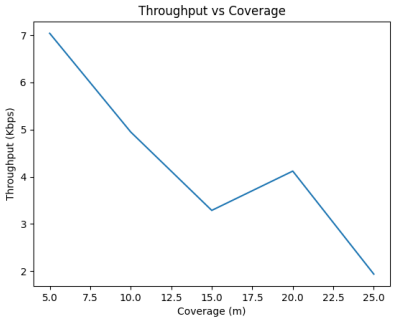
(b) Delay vs Flow



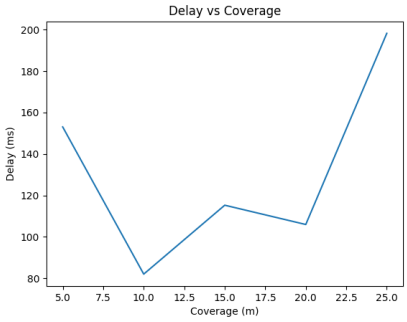
(c) Delivery Ratio vs Flow

(d) Drop vs Flow

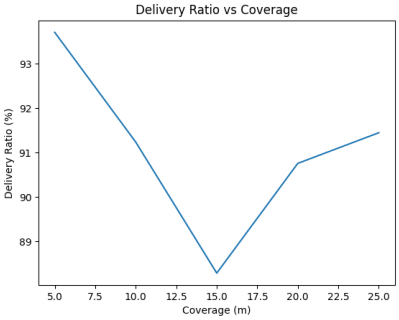
Figure 3: Metrics vs Number of Flows



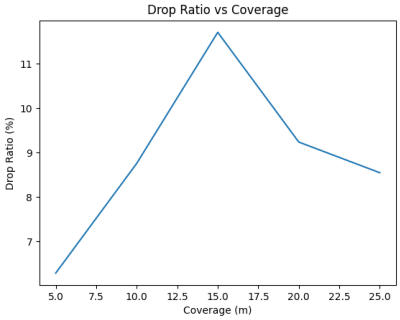
(a) Throughput vs Coverage



(b) Delay vs Coverage

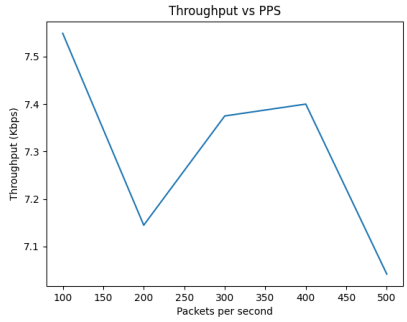


(c) Delivery Ratio vs Coverage

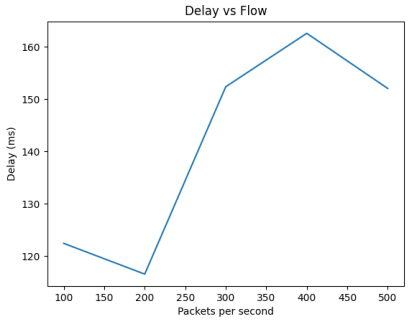


(d) Drop vs Coverage

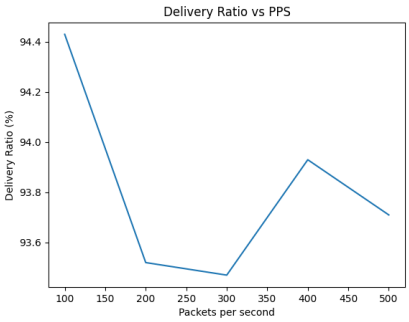
Figure 4: Metrics vs Coverage Area



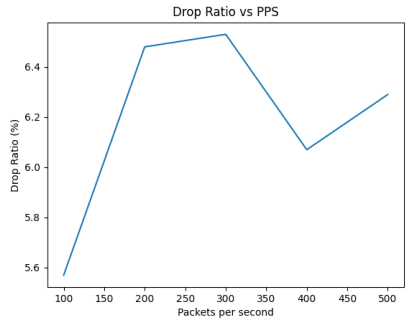
(a) Throughput vs Packets per Second



(b) Delay vs Packets per Second



(c) Delivery Ratio vs Packets per Second



(d) Drop vs Packets per Second

Figure 5: Metrics vs Packets per Second

2.1.4 Results and Explanation

From *Figure 2*, we can see the throughput of the network decreases as the number of nodes increase. In these results, the number of flows is kept constant and the number of nodes is increased. As the network becomes less dense for the same number of flows, the delay decreases, delivery ratio increases and drop ratio decreases.

From *Figure 3*, we can see the throughput more or less decreases as the number of flows increases in a network. Since there are higher chances of congestion with more flows, the throughput decreases and end to end delay increases.

From *Figure 4*, we can see as the coverage of nodes are increased, the throughput decreases and delay increases. The delivery ratio decreases and drop ratio increases. This might happen because of the error induced in long coverage area.

From *Figure 5*, we can see as the packets per second increases, the throughput of the network somewhat increases. The overall delay however increases. This might happen because all the source nodes are pushing packets in the network at a high speed. Hence the delivery ratio is decreased and drop ratio is increased.

2.2 Wireless low-rate (mobile)

2.2.1 Topology

In the topology, there are two wired nodes. The rest of the nodes are wireless. One node is common both in wired and wireless nodes. The wireless nodes send packets to the wired node at the end via the common node. The number of the nodes and senders are kept variable. The wireless nodes are initially placed in a grid where they move following the Random2DWalkModel in NS3.

2.2.2 Variation of Parameters

In the simulation, the following parameters were tweaked to observe the network performance:

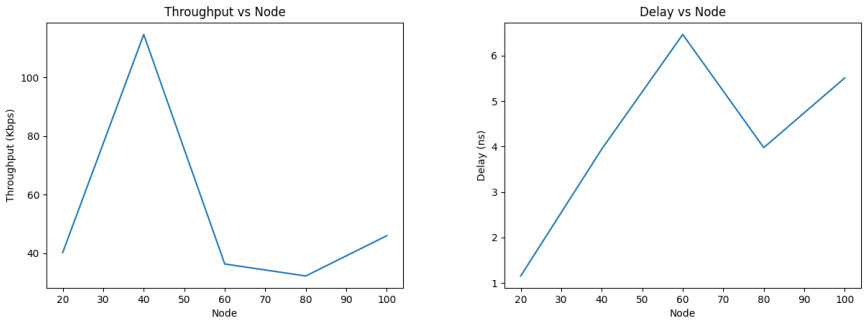
- Number of Nodes
- Number of Flows
- Number of Packets per Second
- Speed of Nodes

Different metrics were recorded under changing of these parameters.

2.2.3 Metric Measurement and Graphs

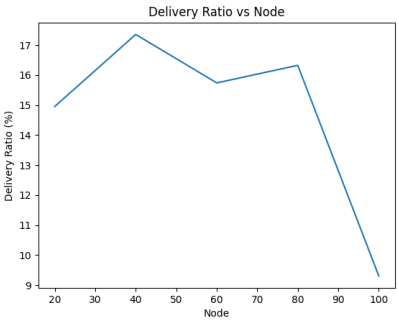
Tweaking the parameters stated above, metrics like Network Throughput, End to End Delay, Packet Delivery Ratio and Packet Drop Ratio were observed. The

average metric of all the flows were recorded here, as they provide an overall performance of the network.

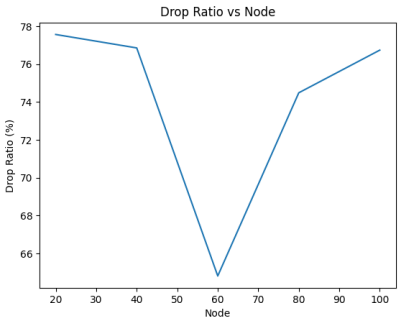


(a) Throughput vs Node

(b) Delay vs Node

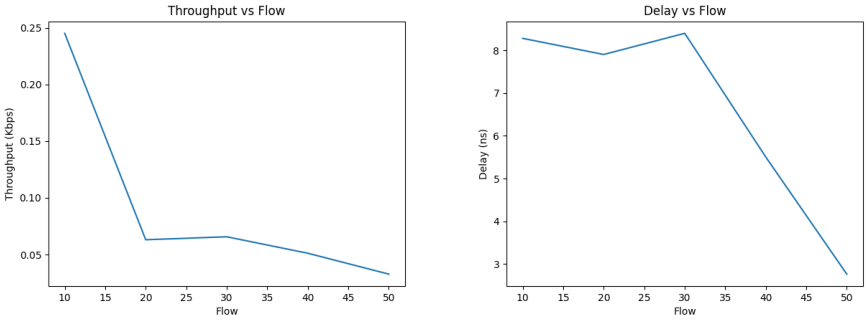


(c) Delivery Ratio vs Node



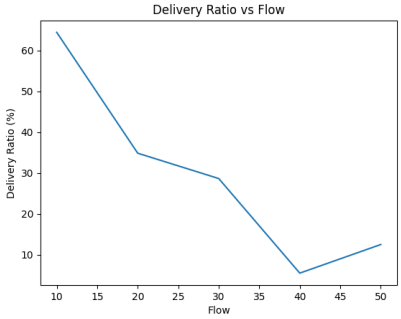
(d) Drop vs Node

Figure 6: Metrics vs Number of Nodes

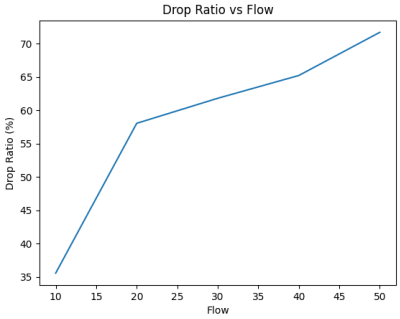


(a) Throughput vs Flow

(b) Delay vs Flow

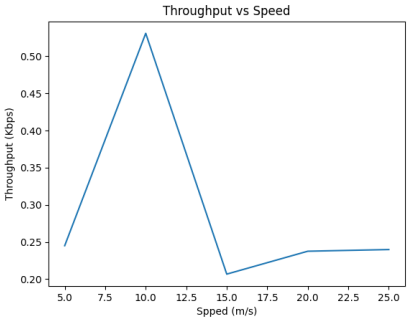


(c) Delivery Ratio vs Flow

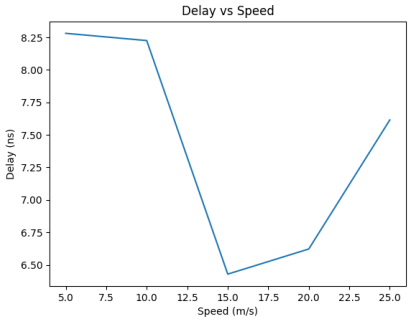


(d) Drop vs Flow

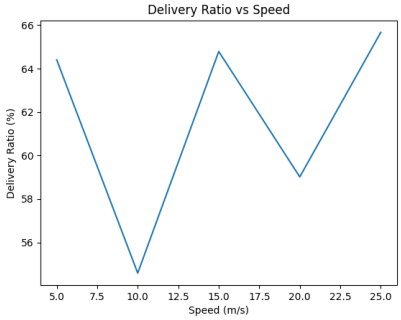
Figure 7: Metrics vs Number of Flows



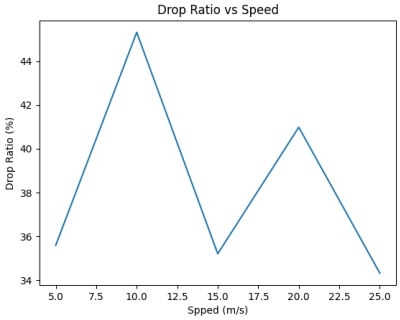
(a) Throughput vs Speed



(b) Delay vs Speed

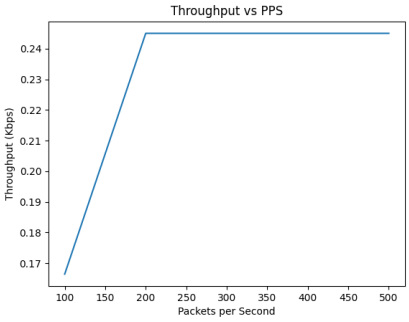


(c) Delivery Ratio vs Speed

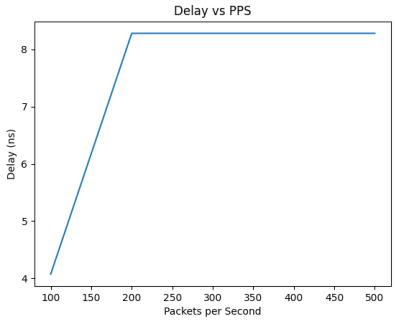


(d) Drop vs Speed

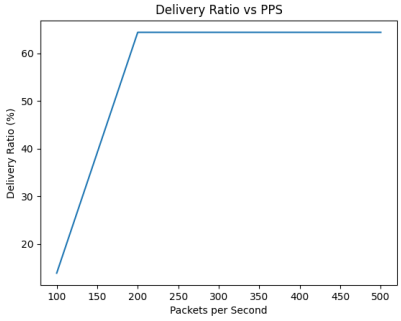
Figure 8: Metrics vs Speed



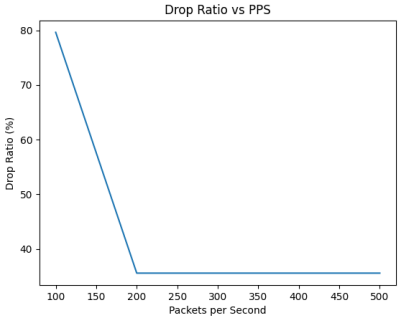
(a) Throughput vs Packets per Second



(b) Delay vs Packets per Second



(c) Delivery Ratio vs Packets per Second



(d) Drop vs Packets per Second

Figure 9: Metrics vs Packets per Second

2.2.4 Results and Explanation

In *Figure 6* we can see, as the count of nodes increases while the flows stay the same, the throughput of the network decreases. The end to end delay increases as more nodes are in the network. Delivery ratio faces a hard time to stay stable and drop ratio is high. In *Figure 7* we see that the throughput drastically drops as more flows are induced in the network. End to end delay is also high for a while. The delivery ratio also gets decreased as more nodes send flows into the network. Hence, the drop ratio increases. From *Figure 8* we can observe that the throughput of the network decreases as the nodes start to move more randomly. End to end delay decreases for an instance but it also increases as the speed increases. Packet delivery ratio is very unstable while nodes move in more speed, so is the drop ratio. In *Figure 9* we can see that the metrics remain same while packets sent per second increases.

3 Task B

3.1 Overview

TCP-Vegas predicts congestion by measuring the round trip times(RTT) of the packets. The congestion window is adjusted after each acknowledgement is received. In my selected paper, it is claimed that in TCP-Vegas, the throughput of a wireless ad hoc network decreases as more flows are induced in the network. This is because of the large minimum congestion window, large reset slow start threshold and aggressive window increase policy. Hence modifications are done to adjust the congestion window more slowly, probing the network with more than one RTT and keeping window unchanged until the number of received ACKs is larger than a threshold. This is done both in slow start and congestion avoidance phase. The slow start threshold is also adjusted the same trying to remain close to the optimum congestion window.

3.2 Modifications

3.2.1 Slower Start

In slower start, the window is changed after the number of received ACKs is larger than a threshold. In my paper, the following algorithm is proposed:

$W = W$, if $\Delta < \gamma$ and $n_s <= N_s$
$W = W + 1$, if $\Delta < \gamma$ and $n_s > N_s$
$W = W * (1 - p)$, if $\Delta > \gamma$

Here, n_s is the number of received ACKs satisfying $\Delta < \gamma$ in slow start phase. N_s is the window increase threshold. p is a percentage.

```
if (diff > m_gamma && (tcb->m_cWnd < tcb->m_ssThresh))
  {
    /*
     * We are going too fast. We need to slow down and change from
     * slow-start to linear increase/decrease mode by setting cwnd
     * to target cwnd. We add 1 because of the integer truncation.
     */
    NS_LOG_LOGIC ("We are going too fast. We need to slow down and "
                  "change to linear increase/decrease mode.");

    // segCwnd = std::min (segCwnd, targetCwnd + 1);
    segCwnd = segCwnd * (1-p);                                    // custom
    tcb->m_cWnd = segCwnd * tcb->m_segmentSize;
    tcb->m_ssThresh = tcb->m_cWnd.Get() - 1;

    NS_LOG_DEBUG ("Updated cwnd = " << tcb->m_cWnd <<" ssthresh=" << tcb->m_ssThresh);
  }
```

Figure 10: $\Delta > \gamma$

```
// ********** Slower Start ********** //
uint32_t
TcpVegasW::SlowStart (Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
  NS_LOG_FUNCTION (this << tcb << segmentsAcked);

  if (segmentsAcked >= 1)
    {
      if(ns > Ns)
      {
        tcb->m_cWnd += tcb->m_segmentSize;
        NS_LOG_INFO ("In SlowStart, updated to cwnd " << tcb->m_cWnd << " ssthresh " << tcb->m_ssThresh);
        ns = 0;
      }
      else
      {
        ns++;
      }
      return segmentsAcked - 1;
    }

  return 0;
}
```

Figure 11: $\Delta < \gamma$

3.2.2 Moderate Congestion Avoidance

In Moderate Congestion Avoidance, the window increase speed is slowed down in congestion avoidance phase.

$$
\begin{aligned}
W = W + 1/W, &\quad \text{if } \Delta <= \alpha \text{ and } n_C A > N_C A \\
W = W, &\quad \text{if } \alpha < \Delta < \beta \text{ or } (\Delta <= \alpha \text{ and } n_C A <= N_C A\ ) \\
W = W - 1/W, &\quad \text{if } \alpha >= \beta
\end{aligned}
$$

```
{     // Linear increase/decrease mode
  NS_LOG_LOGIC ("We are in linear increase/decrease mode");
  if (diff > m_beta)
    {
      // We are going too fast, so we slow down
      NS_LOG_LOGIC ("We are going too fast, so we slow down by decrementing cwnd");
      segCwnd--;
      tcb->m_cWnd = segCwnd * tcb->m_segmentSize;
      NS_LOG_DEBUG ("Updated cwnd = " << tcb->m_cWnd <<" ssthresh=" << tcb->m_ssThresh);

      nsca = 0;
    }
  else if (diff < m_alpha)
    {
      // We are going too slow (having too little data in the network),
      // so we speed up.
      NS_LOG_LOGIC ("We are going too slow, so we speed up by incrementing cwnd");

      if(nca <= Nca)
      {
        nca++;
      }
      else
      {
        segCwnd++;
        tcb->m_cWnd = segCwnd * tcb->m_segmentSize;
        nca = 0;
      }

      NS_LOG_DEBUG ("Updated cwnd = " << tcb->m_cWnd <<" ssthresh=" << tcb->m_ssThresh);
    }
```

Figure 12: Moderate Congestion Avoidance

3.2.3 W_{th} Update

Method:

1. Calculate $Expected$ and $Actual$
2. Calculate the integral number of packets congested:
   $\Delta = (int)((Expected - Actual) * baseRTT + 0.5)$
3. **IF** $w < W_{th}$
4. **IF** $\Delta > \gamma$
5. $W_{th} \leftarrow$ current window minus 1
6. **END**
7. **ELSE**
8. **IF** $\Delta < \alpha$
9. ns_{CA} keeps unchanged
10. **END**
11. **IF** $\Delta > \beta$
12. $ns_{CA} \leftarrow 0$
13. **END**
14. **IF** $\alpha \leq \Delta \leq \beta$
15. **IF** $Currentwindow > W_s$
16. $ns_{CA} \leftarrow 0$
17. $W_s \leftarrow$ current window
18. **END**
19. $ns_{CA} \leftarrow ns_{CA} + 1$
20. **IF** $ns_{CA} > NS_{CA}$
21. $W_{th} \leftarrow w$
22. $ns_{CA} \leftarrow 0$
23. **END**
24. **END**
25. **END**

Figure 13: Proposed Algorithm

```
if (diff > m_gamma && (tcb->m_cWnd < tcb->m_ssThresh))
  {
    /*
     * We are going too fast. We need to slow down and change from
     * slow-start to linear increase/decrease mode by setting cwnd
     * to target cwnd. We add 1 because of the integer truncation.
     */
    NS_LOG_LOGIC ("We are going too fast. We need to slow down and "
                  "change to linear increase/decrease mode.");

    // segCwnd = std::min (segCwnd, targetCwnd + 1);
    segCwnd = segCwnd * (1-p);                                    // custom
    tcb->m_cWnd = segCwnd * tcb->m_segmentSize;
    tcb->m_ssThresh = tcb->m_cWnd.Get() - 1;

    NS_LOG_DEBUG ("Updated cwnd = " << tcb->m_cWnd <<" ssthresh=" << tcb->m_ssThresh);
  }
```

Figure 14: $w < W_{th}$

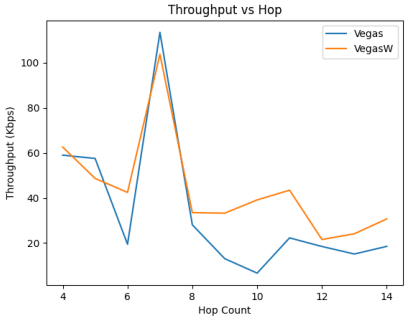
```
{     // Linear increase/decrease mode
  NS_LOG_LOGIC ("We are in linear increase/decrease mode");
  if (diff > m_beta)
    {
      // We are going too fast, so we slow down
      NS_LOG_LOGIC ("We are going too fast, so we slow down by decrementing cwnd");
      segCwnd--;
      tcb->m_cWnd = segCwnd * tcb->m_segmentSize;
      NS_LOG_DEBUG ("Updated cwnd = " << tcb->m_cWnd <<" ssthresh=" << tcb->m_ssThresh);

      nsca = 0;
    }
```

Figure 15: $\alpha >= \beta$

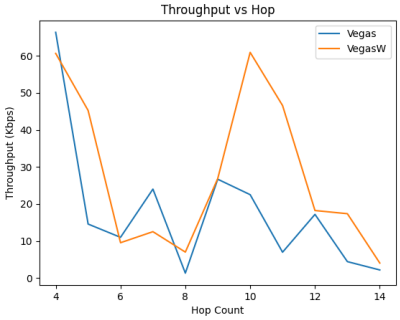
```
    else
      { // alpha < diff < beta
        // We are going at the right speed
        NS_LOG_LOGIC ("We are sending at the right speed");

        if(tcb->m_cWnd.Get() > Ws)
        {
          nsca = 0;
          Ws = tcb->m_cWnd.Get();
        }
        nsca++;
        if(nsca > Nsca)
        {
          tcb->m_ssThresh = tcb->m_cWnd;
          nsca = 0;
        }
      }
```

Figure 16: $\alpha < \Delta < \beta$

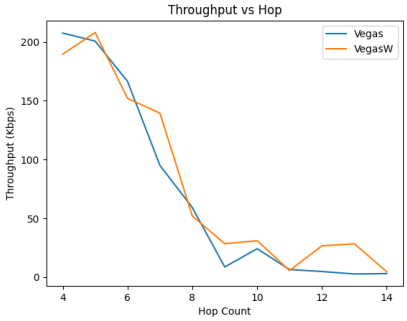
3.3 Metric Measurement and Graphs



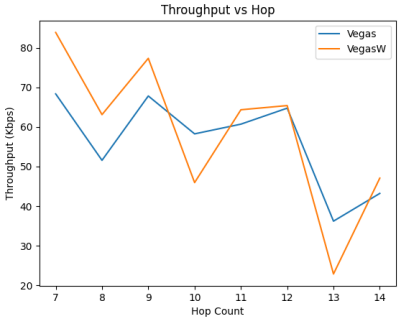
(a) Throughput vs One Flow



(b) Throughput vs Two Flow



(c) Throughput vs Four Flow



(d) Throughput vs Eight Flow

Figure 17: Throughput comparison

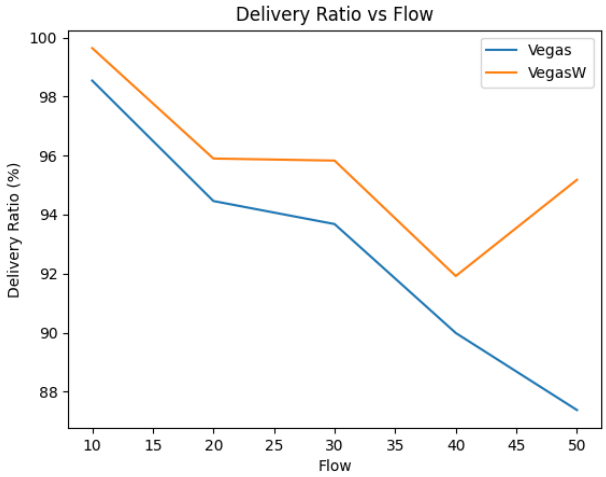


Figure 18: Packet Delivery Ratio vs Flow

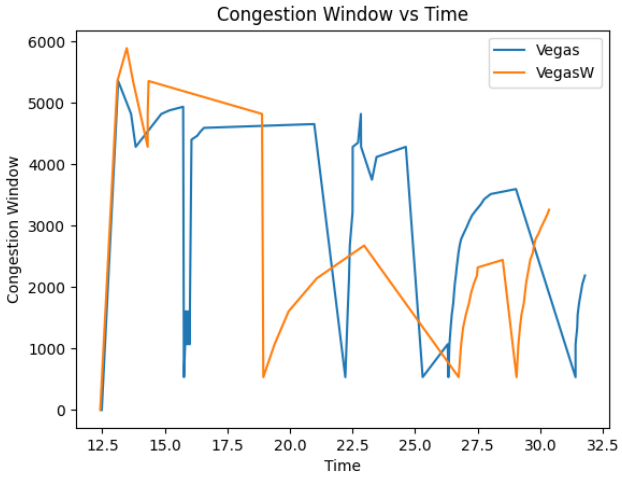


Figure 19: Congestion Window vs Time

3.4 Results and Explanation

In *Figure 17* throughput vs hop count for both TCP-Vegas and proposed TCP-VegasW has been plotted for different flows in the network. The comparison has been shown in the original paper and so the similar graph is plotted. The output might not be completely similar to the paper, but the overall throughput of TCP-Vegas-W is observed to be better than that of TCP-Vegas. For more flows and more hop counts, TCP-Vegas-W performs somewhat better than TCP-Vegas.

In *Figure 18* TCP-Vegas-W achieves a higher percentage of packet delivery ratio than TCP-Vegas for higher flows.

In *Figure 19* it can be observed that through time, the congestion window of TCP-Vegas-W faces somewhat less up and downs than that of TCP-Vegas. The congestion window of TCP-Vegas-W stays more stable than TCP-Vegas.

4 Conclusion

The results are not always perfect. The inconsistencies might happen because of the different values of other variables, orientation of the nodes and the overall configuration of the network.

Supervised by -
Md. Tareq Mahmood
Department of CSE, BUET