CSE 322: NS3 Project Report

TCP Adaptive Reno

Najibul Haque Sarker
1705044

Supervisor:
Syed Md. Mukit Rashid
Lecturer, Dept. of CSE
Bangladesh University of Engineering and Technology

February 22, 2022

Contents

1 Introduction

For Task A, I need to implement networks from row 0. ($ID \bmod 6 = 1705044 \bmod 6 = 0$). For Task B, my proposed algorithm is a TCP congestion control algorithm named TCP-AReno or Adaptive Reno[1].

- Task A1 : Wireless high-rate (e.g. 802.11) (static)
- Task A2 : Wireless low-rate (e.g. 802.15.4) (static)
- Task B : TCP-AReno Implementation

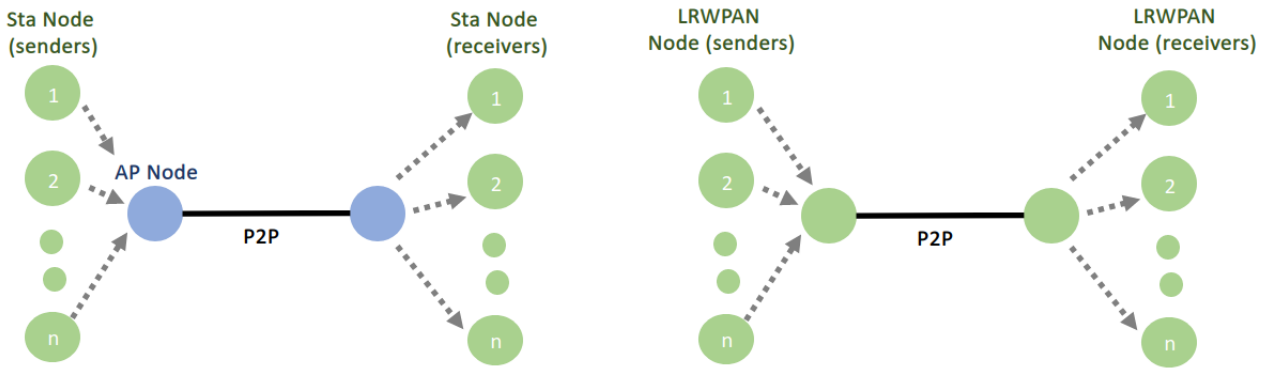
2 Network Topologies under simulation

Task A1

Topology used is similar to a n-node dumbbell topology. A point to point wired connection is established between two wifi AP nodes. One AP node is connected to a series of sender wifi station nodes, and the other AP node is connected to a series of receiver nodes. All these nodes are positioned in a grid.

Task A2

Topology used is similar to a n-node dumbbell topology. $2N + 2$ low rate wireless pan nodes are created from which 2 nodes are connected in a point to point wired connection. One of these nodes is connected to N sender nodes and the other is connected with N receiver nodes. All these nodes are positioned in a grid.



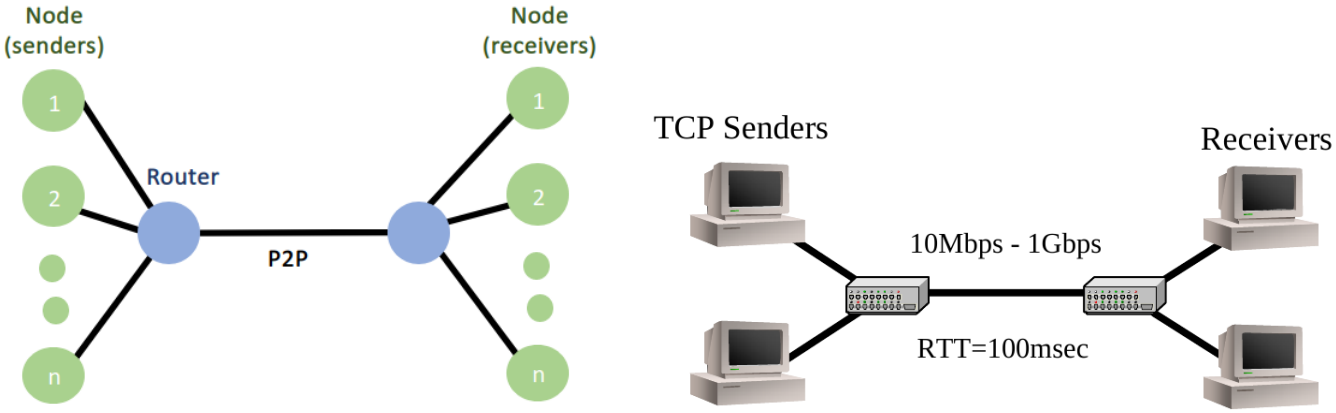
(a) Wireless high-rate network topology

(b) Wireless low-rate network topology

Figure 1: **Task A Topology**

Task B

Topology used is a n-node dumbbell topology. *Point to point dumbbell helper* is used to create a point to point connection between 2 routers, where one router will have point to point connections with N sender nodes and the other router will have point to point connections with N receiver nodes.



(a) Task B Dumbbell Topology (b) Paper Topology

Figure 2: **Task B Topology vs Paper Topology**

The paper topology uses 2 nodes as leaf nodes on each side. The experiments from the paper are conducted on 2 topologies, one is considering no of leaf nodes = 2, another is considering no of leaf nodes = 10. Two TCP variants will be installed, one on each odd nodes and the other on each even leaf nodes.

3 Parameters under variation

Task A

From both networks under TaskA, the following metrics were calculated:

1. Network Throughput
2. End-to-end Delay
3. Packet delivery ratio
4. Packet drop ratio

All of these were calculated while varying the following parameters:

1. Number of nodes along with number of flows
2. Number of flows
3. Number of packets per second
4. Coverage area

Task B

For TaskB, the following metrics were calculated using Fig. 2 topology. The number of leaf nodes was varied to 2 and 10 to make two different dumbbell topologies for the following experiments:

1. Network Throughput
2. Jain's Fairness Index

The following parameters were varied:

1. Bottleneck Link Capacity
2. Random Packet Loss Rate

Furthermore, the congestion window change over time for used congestion control algorithms were also calculated and plotted.

4 Overview of the proposed algorithm

The widely used congestion control algorithm TCP-Reno has some disadvantages. Its' throughput decreases in networks with large bandwidth delay product and with non negligible packet loss. Other congestion control algorithms like TCP-Westwood / TCP-HighSpeed attempts to solve this problem but their potential unfriendliness to TCP-Reno is one of the reasons hampering their deployment.

In order to tackle this problem, a new congestion algorithm is proposed named TCP-AReno or Adaptive Reno. This algorithm is based on TCP-Westwood-BBE. It has the following attributes:

- Estimates congestion level via RTT to determine whether a packet loss is due to congestion or not.
- Introduces a fast window expansion mechanism to quickly increase congestion window whenever it finds network underutilization. The congestion window increase will have 2 parts: The **base part** increases linearly in congestion avoidance phase just like TCP-Reno whereas the **probe part** increase exponentially to utilize the network fully.
- Adjusts congestion window reduction based on the congestion measurement. Congestion window is halved when the network is congested and a packet loss is likely to happen, while the reduction is mitigated when the network is underutilized.

Finally the paper claims the TCP-AReno modifications results in:

- Higher throughput than TCP-Reno.
- Friendlier to TCP-Reno over a wide range of link capacities and random packet loss rates.

5 Modifications made in the simulator

TCP-AReno is implemented in NS3 simulator via inheriting the class **TcpWestwood** and creating a new class named **TcpAdaptiveReno**. The header file of the class is in **tcp-adaptive-reno.h** and the class is in **tcp-adaptive-reno.cc**. More details as follows:

tcp-adaptive-reno.h

Contains class declarations, function prototypes of **TcpAdaptiveReno**. The following new class variables were added:

- **m_minRtt** : the minimum Rtt over the whole simulation
- **m_currentRtt** : the current Rtt value for a packet ACK event
- **m_jPacketLRtt** : the Rtt for j-th packet loss event
- **m_conjRtt** : the congestion Rtt for j-th packet loss event
- **m_prevConjRtt** : the congestion Rtt for (j-1)-th packet loss event
- **m_incWnd** : the increase of window for the window probe part
- **m_baseWnd** : base part of the calculated congestion window
- **m_probeWnd** : probe part of the calculated congestion window

The following inherited functions were changed:

- **PktsAcked** : The function is called every time an ACK is received
- **GetSsThresh** : This function returns the slow start threshold after a loss event
- **CongestionAvoidance** : This function increases congestion window in congestion avoidance phase

The following functions were added unique to **TcpAdaptiveReno**:

- **EstimateCongestionLevel** : This function estimates the current congestion level
- **EstimateIncWnd** : This function estimates the increase of the probe portion of congestion window.

tcp-adaptive-reno.cc

Contains the function implementations. The exact modifications in each functions are described below:

- **PktsAcked** : Calculates the values for m_minRtt and m_currentRtt.

- **EstimateCongestionLevel** : Estimates the current congestion level using m_minRtt, m_currentRtt, m_prevConjRtt, m_jPacketLRtt. The j-th packet loss congestion RTT is calculated as follows $RTT^j_{cong} = aRTT^{j-1}_{cong} + (1-a)RTT^j$ where a=0.85. Then the congestion level c is is:

$$c = \min(\frac{RTT - RTT_{min}}{RTT_{cong} - RTT_{min}}, 1)$$

- **EstimateIncWnd** : Calculate W^{max}_{inc} using **TcpWestWood**'s **EstimateBW** function which estimates the current bandwidth (done in **PktsAcked** function) and update the value of m_incWnd using the following equations:

$$\begin{aligned} W^{max}_{inc} &= B/M * MSS \\ \alpha &= 10 \\ \beta &= 2W^{max}_{inc}(1/\alpha - (1/\alpha + 1)/e^{\alpha}) \\ \gamma &= 1 - 2W^{max}_{inc}(1/\alpha - (1/\alpha + 1/2)/e^{\alpha}) \\ W_{inc}(c) &= W^{max}_{inc}/e^{c\alpha} + c\beta + \gamma \end{aligned}$$

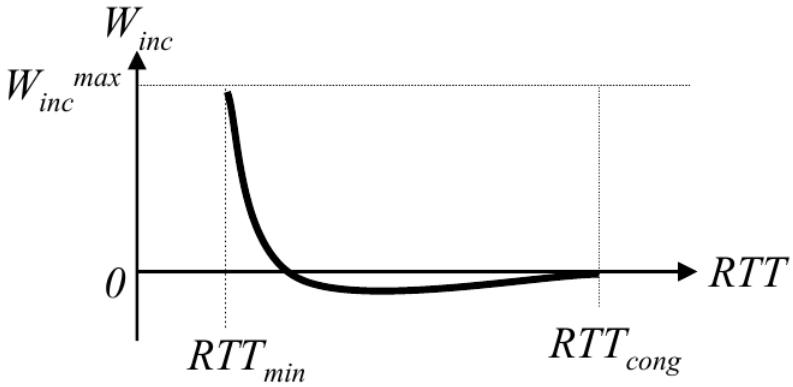


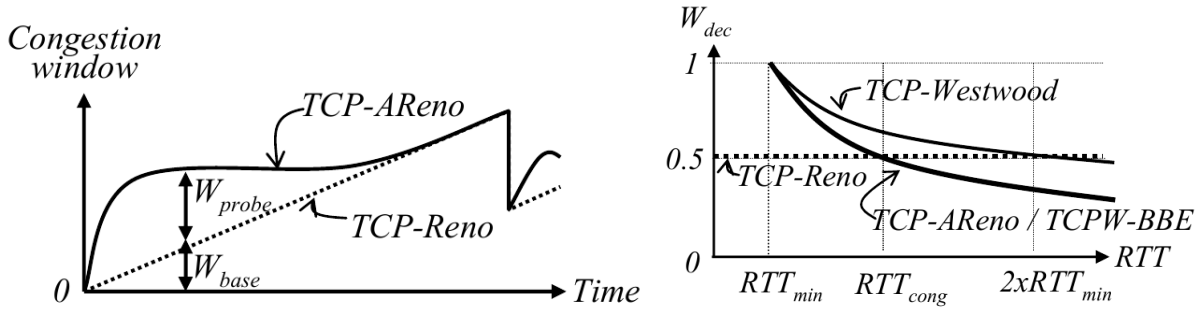
Figure 3: Window calculation in EstimateIncWnd

- **CongestionAvoidance** : Updates m_baseWnd and m_probeWnd. Calculates the new congestion window using the following:

$$\begin{aligned} W_{base} &= W_{base} + 1MSS/W \\ W_{probe} &= \max(W_{probe} + W_{inc}/W, 0) \\ W &= W_{base} + W_{probe} \end{aligned}$$

- **GetSsThresh** : A loss event has occured. So update the values of m_prevConjRtt, m_jPacketLRtt. Estimate the congestion and estimate the new reduced window. Reset values of m_baseWnd and m_probeWnd.

$$W_{base} = W * W_{dec} = \frac{W}{1+c}, \quad W_{probe} = 0$$



(a) Window calculation in Congestion-Avoidance

(b) Window calculation in GetSsThresh

Figure 4: **Window size Calculations**

wscript

The new classes needed to be declared here.

point-to-point-dumbbell.h

The paper uses a dumbbell topology where random packet loss rate needed to be set in the routers of the bottleneck layer. But the **PointToPointDumbbellHelper** class which was used to build the topology in the simulator had its router attributes as private. But access was needed in order to set the loss rate in the device. Thus the member variable m_routerDevices was made public so that the loss rate can be set directly.

6 Results with graphs and Explanations

6.1 Task A1: High-rate Wireless Static

6.1.1 Varying Nodes (and flows)

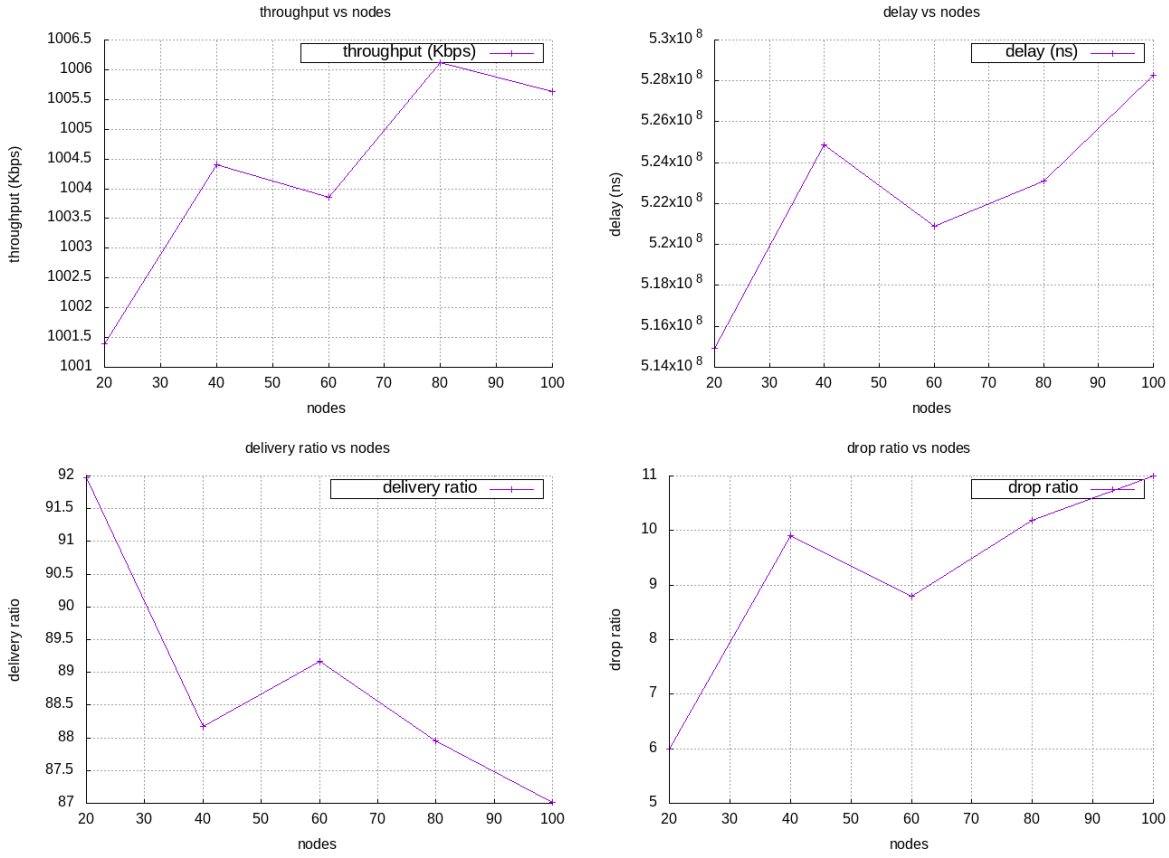


Figure 5: Varying Nodes VS Metrics

Explanation

As number of nodes increase, throughput increases slightly. The more the number of nodes increase, the number of flow increases and the bottleneck link gets utilized. It should be noted that the link capacity is set at 1Mbps, so even from the beginning (node=20) it was used to its full capacity. That is why there is only a slight increase as the nodes grow.

End to end delay increases as more nodes are added because of the congestion control algorithm. This makes packets wait until congestion is below a suitable level to send the packets. Thus the more nodes/flow there are, the more the packets have to wait and the delay increases.

Delivery ratio decreases as more nodes means more congestion and the ratio of received and sent packets will continue to decrease.

For the same reason, that means the drop ratio will increase as more packets are being dropped due to increased congestion.
P.S. There seems to be an outlier at node 60.

6.1.2 Varying Flow

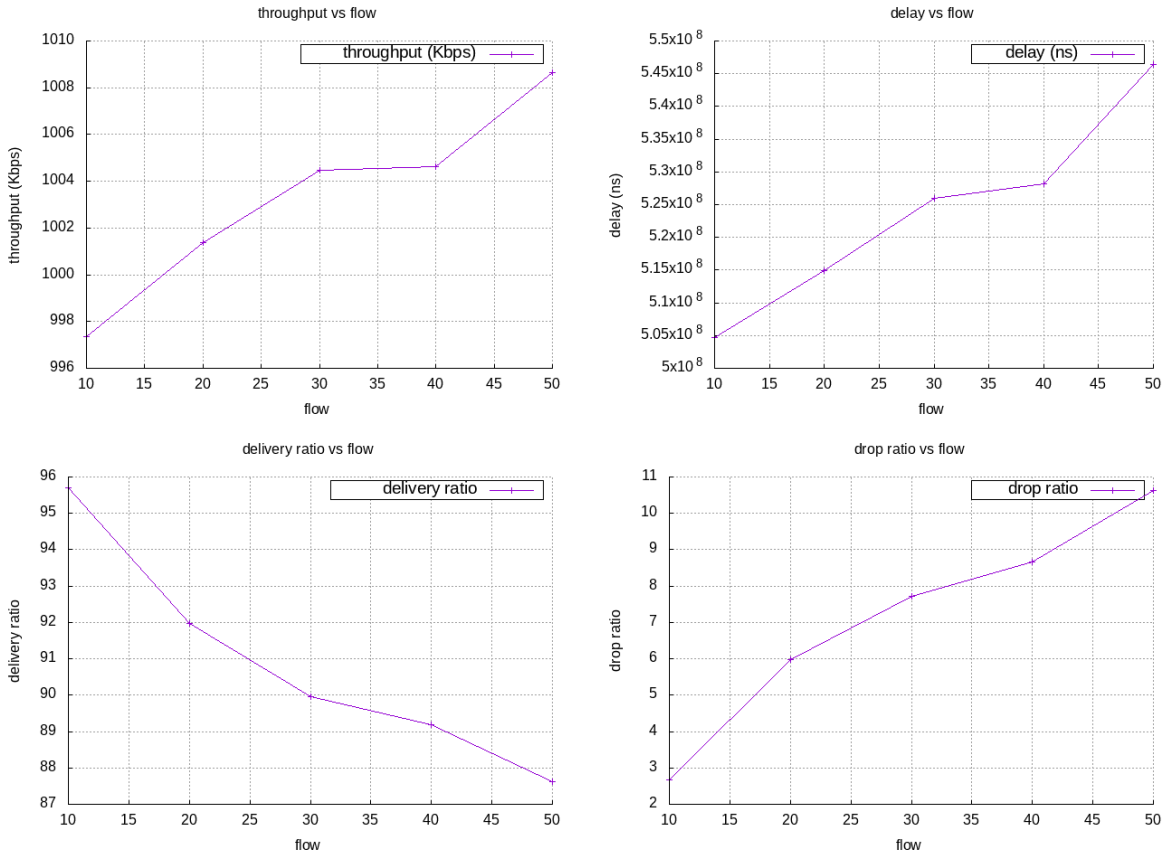


Figure 6: Varying Flow VS Metrics

Explanation

Here nodes are kept constant (at 20) and flows are being varied. This shows the effect of flow increase more clearly. Throughput increases slightly as more flows utilize the bottleneck layer but again it was already running upto capacity from the beginning.

End to end delay increases as an increase of flow creates congestion.

This means that delivery ratio will continue to decline.

And the drop ratio will continue to increase.

6.1.3 Varying Packets Per Second

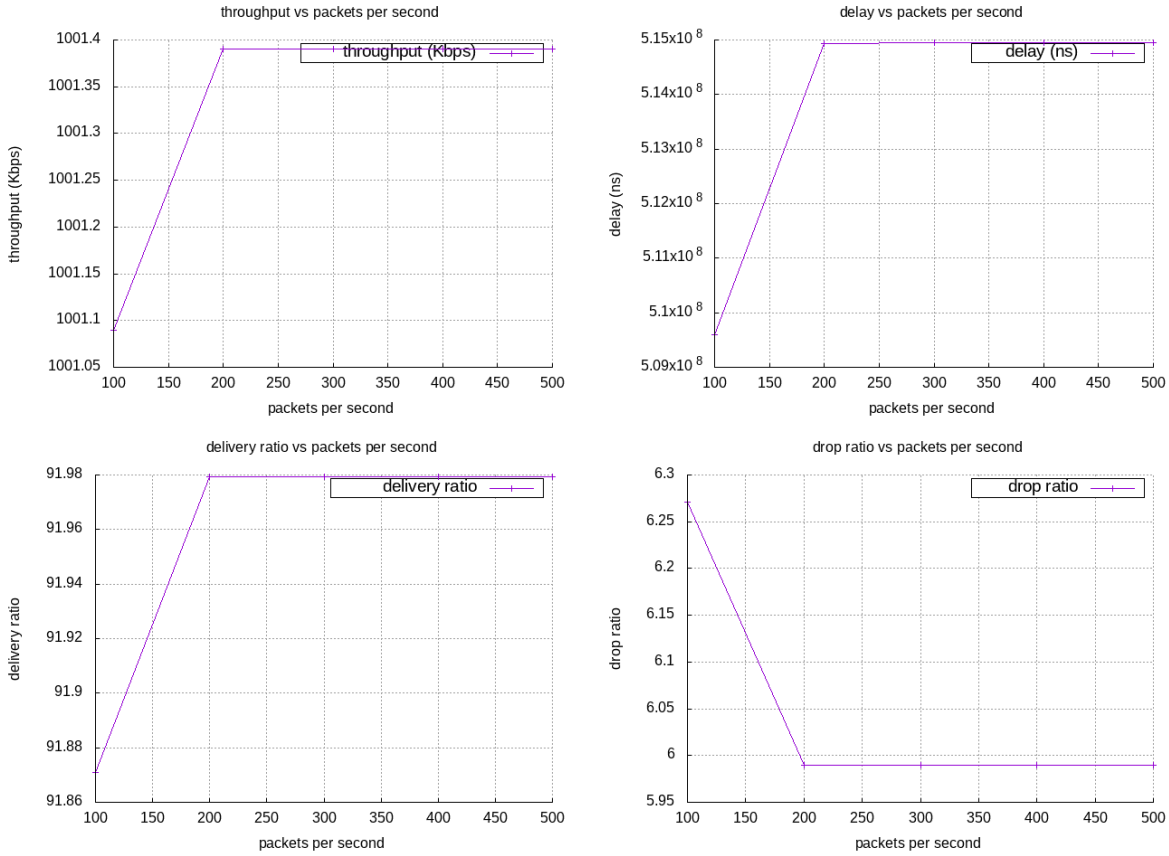


Figure 7: Varying Packets Per Second VS Metrics

Explanation

Here only transmission number of packets per second is being varied. We can see that there is a very slight increase of throughput, end to end delay, delivery ratio and slight decrease of drop ratio. These increase and decrease are too small to be substantial. This points to the fact that, as packets per second is being increased, the congestion control algorithm properly does its job and synchronises the transmission of packets with congestion by delaying some packet transmission until a suitable congestion level. That is why all the metrics remain kind of constant across the increase of packets per second transmission.

6.1.4 Varying Coverage Area

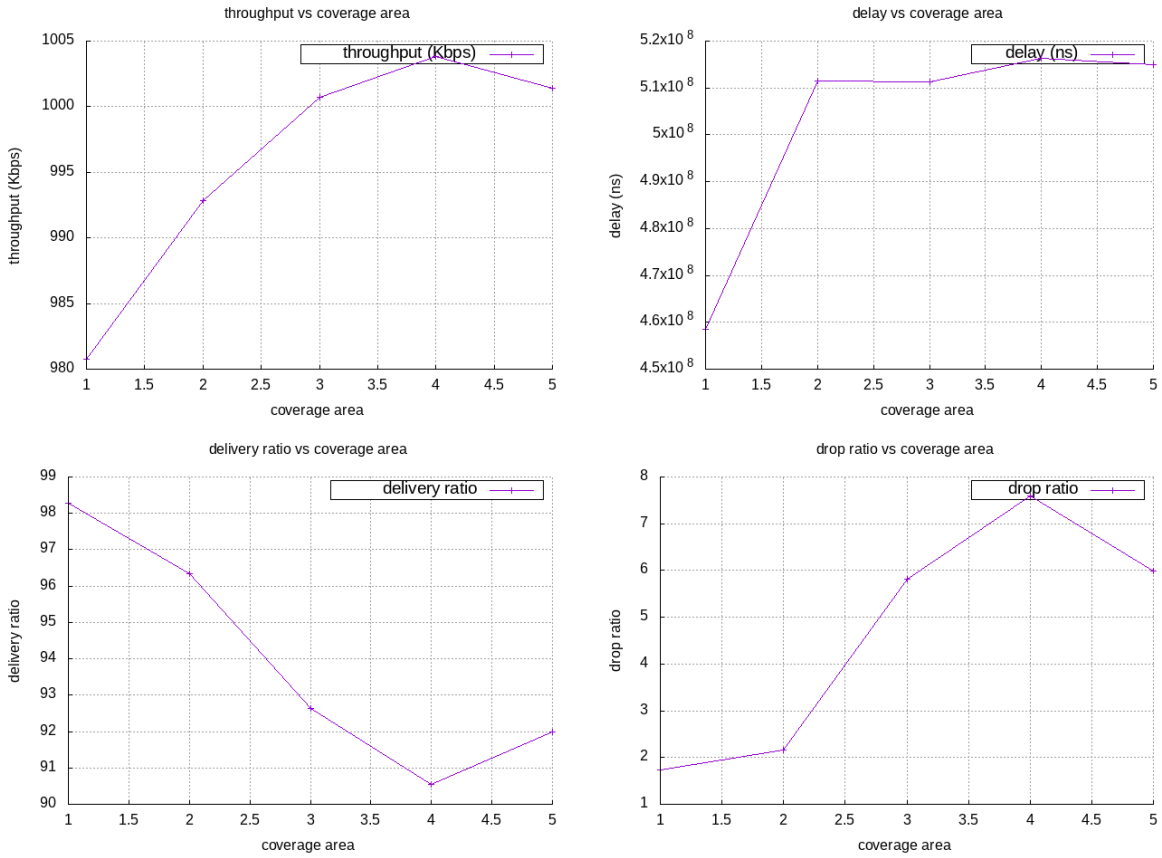


Figure 8: Varying Coverage Area VS Metrics

Explanation

As the nodes were positioned using a **Grid position allocator**, the number of nodes being able to communicate with each other depended on their individual coverage area. For the case where there is supposed to be 20 flows, the following amount of flows were detected at each coverage area level:

- 1*Tx_range : 11 flows
- 2*Tx_range : 13 flows
- 3*Tx_range : 17 flows
- 4*Tx_range : 19 flows
- 5*Tx_range : 19 flows

Thus, as coverage area increased, more nodes were able to communicate and thus flows increased. That is why we see a slight increase of throughput, end to end delay and drop ratio, whereas the delivery ratio decreased. (Similar to section 6.1.2).
P.S. Coverage area 5*Tx_range seems to be an outlier.

6.2 Task A2: Low-rate Wireless Static

6.2.1 Varying Nodes

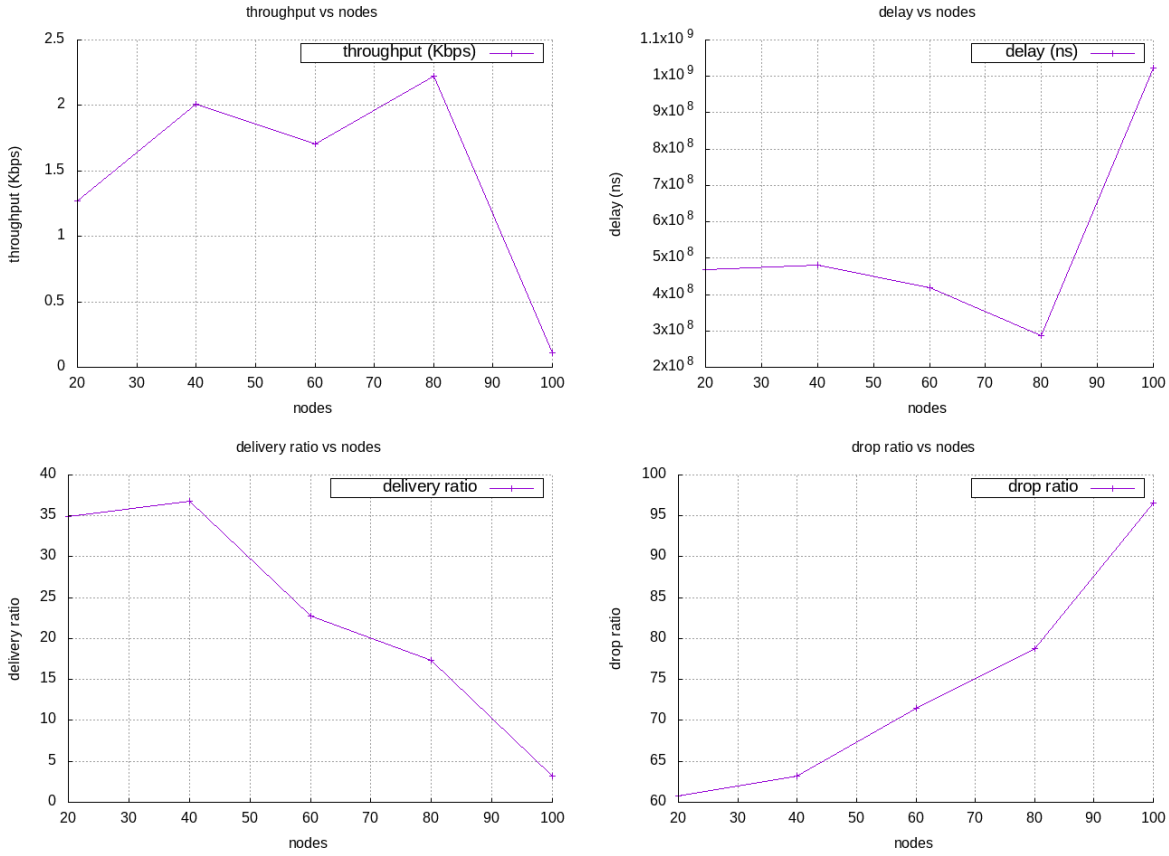


Figure 9: Varying Nodes VS Metrics

Explanation

Here low-rate wireless network is used in a dumbbell topology and this results in a low throughput and delivery ratio whereas drop ratio increased a lot compared to the Section 6.1.1. It seems the throughput increased somewhat as nodes grew but in the end dropped down below 0.5 Kbps. As this is low-rate, congestion seems to be at a minimum. That is why for some node increase, the throughput increased. But as nodes increased a lot, the congestion level increased and this being low-rate the throughput decreased a lot.
For the same reason, delay seemed to decrease until a certain threshold from where it skyrocketed.
Again, delivery ratio seemed to increase at first and then decrease by a lot. It points to the limited capability of transmission for low rate networks.
Unsurprisingly, the drop ratio always increased as the number of nodes grew. Increasing nodes seems to deteriorate the quality of the transmission.

6.2.2 Varying Flow

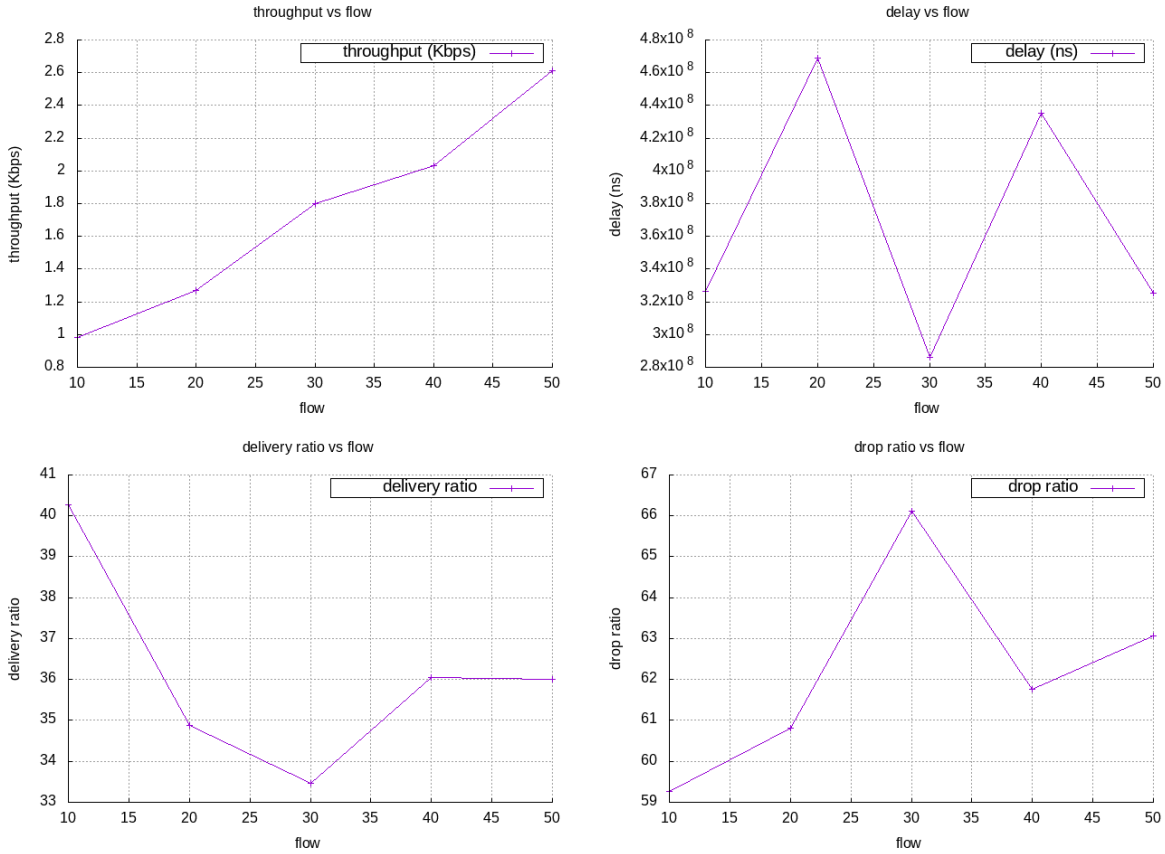


Figure 10: Varying Flow VS Metrics

Explanation

As the number of nodes are kept constant and number of flows are increased, this results in a much better throughput for the network. The plots are a little noisy, but if we see the trends we can infer that the delivery ratio decreases and the drop ratio increases as flows increase. Thus, more flows result in a better utilization of the bottleneck link but this results in a worse delivery rate. End to end delay increases and then decreases which points to the fact that the transmissions gradually result in congestion as the flow and throughput increased. There seems to be a certain number of flows in which the network utilizes the link layer the best.

6.2.3 Varying Packets Per Second

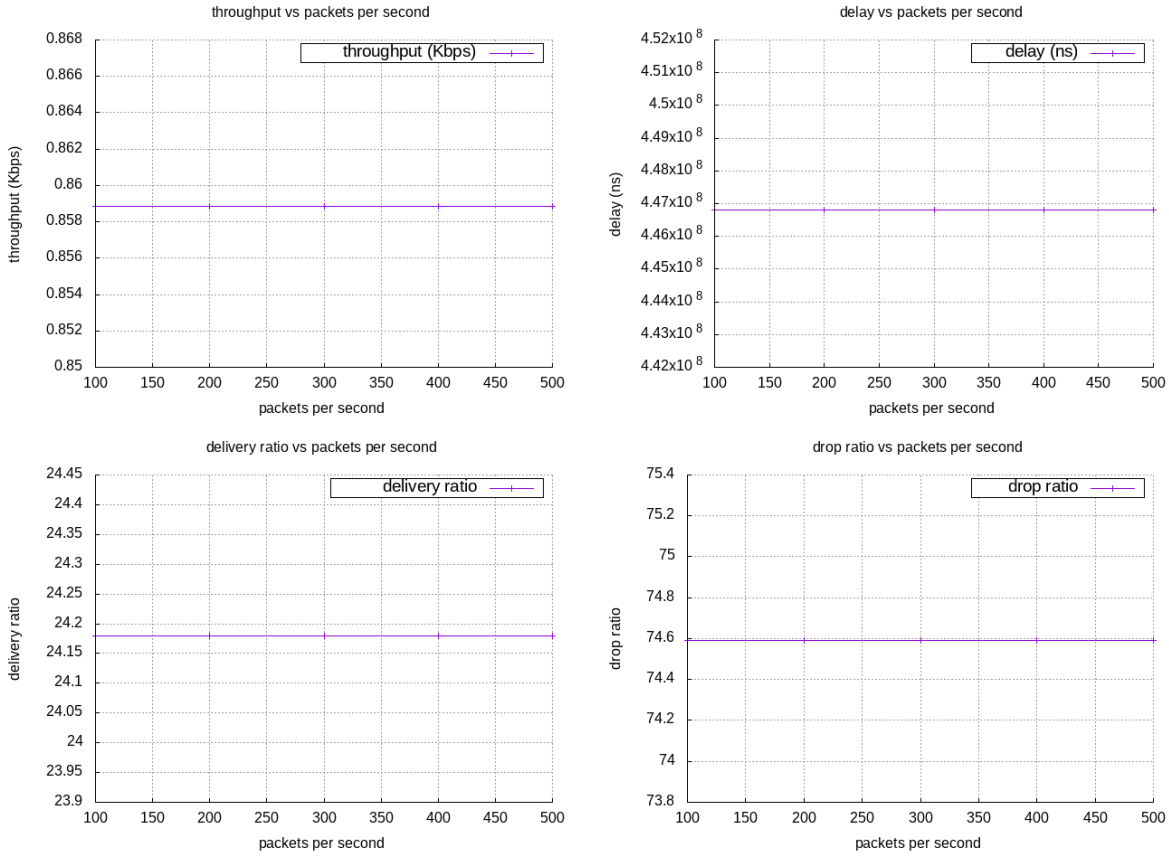


Figure 11: Varying Packets Per Second VS Metrics

Explanation

Here no matter how much the packets per second are varied, all the metrics remain the same. This can be explained using the fact that the congestion control algorithm will do its job incase the transmissions exceed the channel limit and slow down the transmission rate to a suitable one. That is why all the metrics remain the same.

6.2.4 Varying Coverage Area

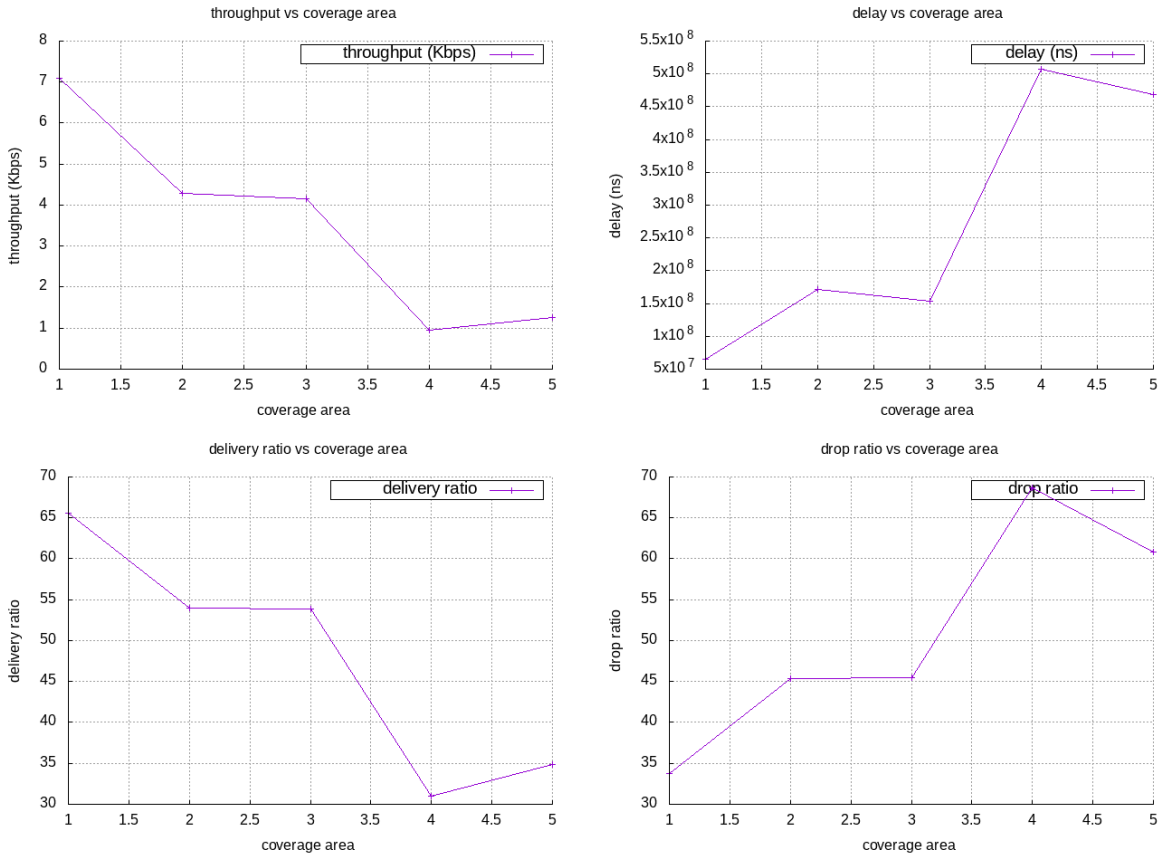


Figure 12: Varying Coverage Area VS Metrics

Explanation

Again the nodes were positioned using a **Grid position allocator** and as the number of nodes being able to communicate with each other depended on their individual coverage area, this resulted in an decrease of flows the less the coverage range became. For the case where there is supposed to be 20 flows, the following amount of flows were detected at each coverage area level:

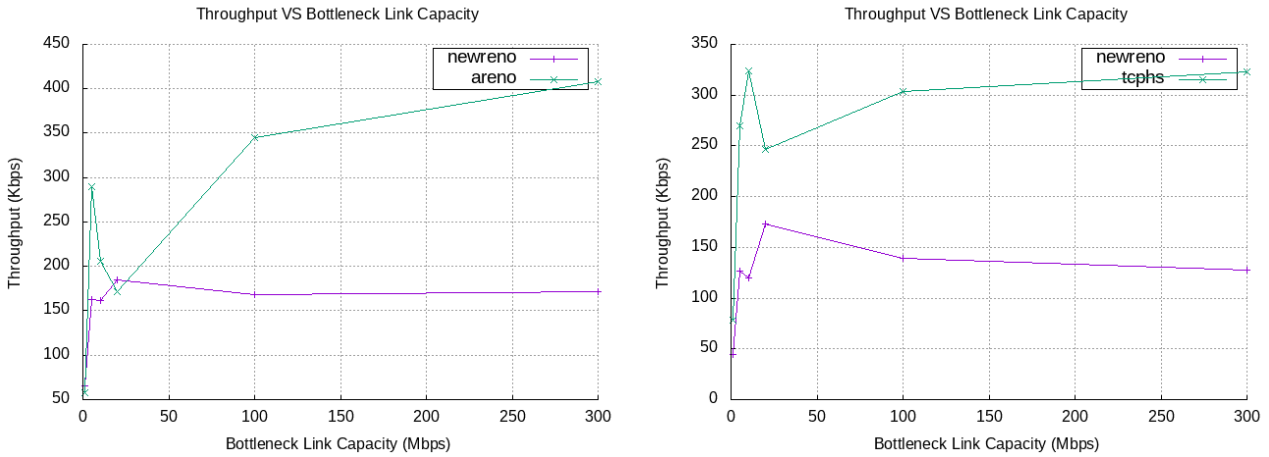
- 1*Tx_range : 13 flows
- 2*Tx_range : 18 flows
- 3*Tx_range : 20 flows
- 4*Tx_range : 20 flows
- 5*Tx_range : 20 flows

As no of flows increased, throughput decreased. Even when no of flows remained the same and coverage increased, throughput decreased. Thus low rate wireless tends to deteriorate as more flows increased. That is why end to end delay increased, delivery

ratio decreased and drop ratio increased as coverage area grew and more flows became active.

6.3 Task B

6.3.1 Throughput vs Bottleneck Link Capacity in Co-existing flows



(a) TCP-NewReno + TCP-AReno (b) TCP-NewReno + TCP-HighSpeed

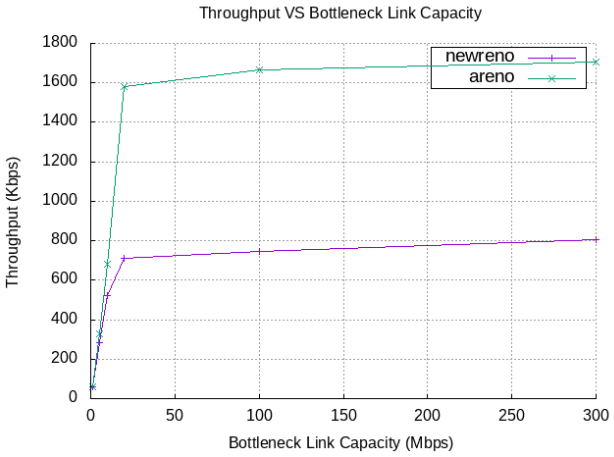
Figure 13: **Throughput VS Varying Bottleneck Link Capacity(no of Leaf=2)**

Explanation

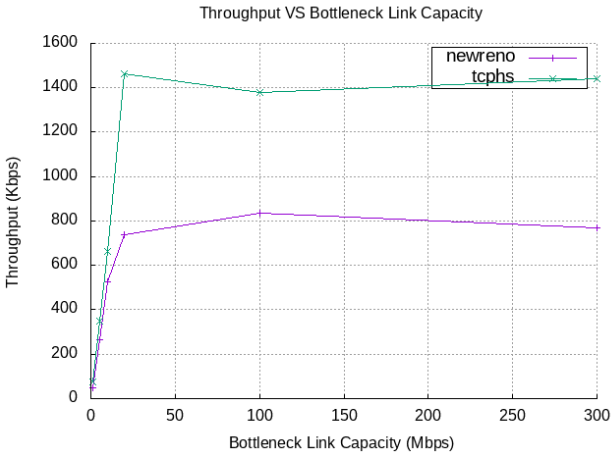
The experiment setup contains 2 leaf nodes on each side of the dumbbell, each containing a tcp congestion control algorithm variant. Thus the setup will contain co-existing tcp congestion control algorithms. By comparing the two pictures, two things are evident:

- Throughput increase of TCP-AReno is greater than both TCP-NewReno and TCP-HighSpeed. Thus this points to the fact that this variant of congestion control results in high throughput in high speed networks.
- If one compares the throughput from TCP-NewReno from the two cases, the increase is evident when TCP-NewReno co-exists with TCP-AReno compared to co-existing with TCP-HighSpeed.

Thus the proposed TCP-AReno has higher throughput and better friendliness to TCP-NewReno.



(a) TCP-NewReno + TCP-AReno



(b) TCP-NewReno + TCP-HighSpeed

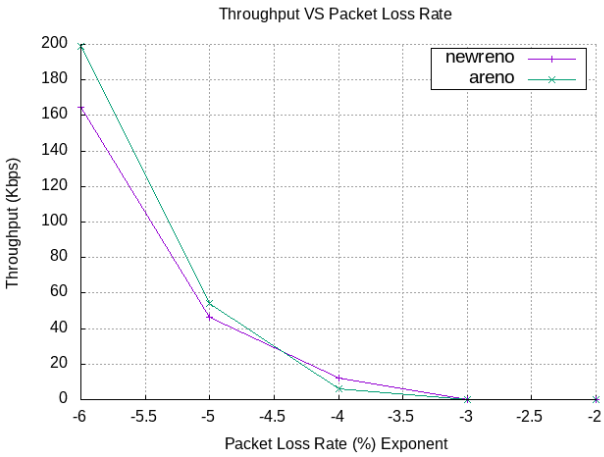
Figure 14: **Throughput VS Varying Bottleneck Link Capacity(no of Leaf=10)**

Explanation

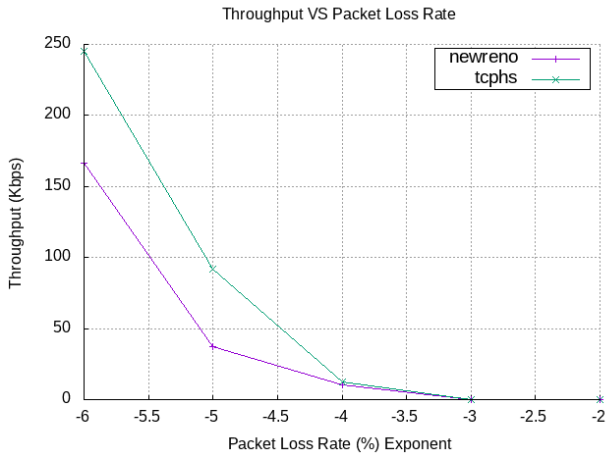
The experiment setup contains 10 leaf nodes on each side of the dumbbell. Here 5 leaf nodes in each side will contain one variant of the two co-exisiting congestion control algorithms. The output metrics of nodes related to each algorithms are aggregrated to present a case where multiple nodes have the two algorithms. It is evident that:

- Throughput increase of TCP-AReno is greater than both TCP-NewReno and TCP-HighSpeed. This again agrees with the fact that this variant of congestion control results in high throughput in high speed networks.
- If one compares the throughput from TCP-NewReno from the two cases, the increase is similar. Thus this experiment raises doubt whether the friendliness of TCP-AReno still holds when there are multiple nodes.

6.3.2 Throughput vs Random Packet Loss Rate in Co-existing flows

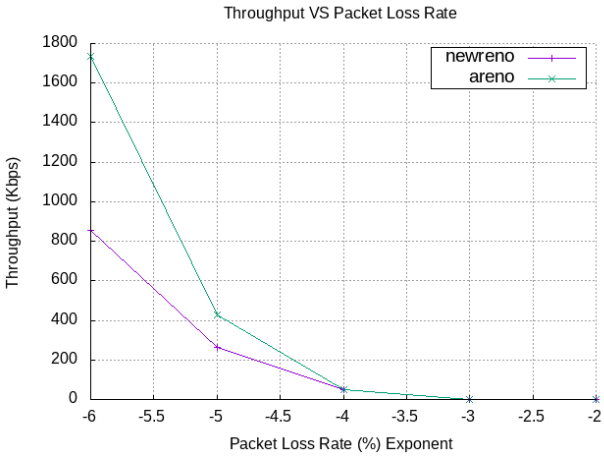


(a) TCP-NewReno + TCP-AReno

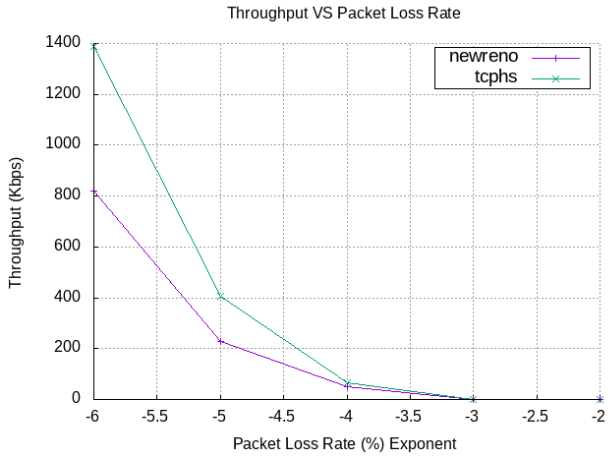


(b) TCP-NewReno + TCP-HighSpeed

Figure 15: **Throughput VS Varying Packet Loss Rate(no of Leaf=2)**



(a) TCP-NewReno + TCP-AReno



(b) TCP-NewReno + TCP-HighSpeed

Figure 16: **Throughput VS Varying Packet Loss Rate(no of Leaf=10)**

Explanation

For the first case where no of leaf is 2:

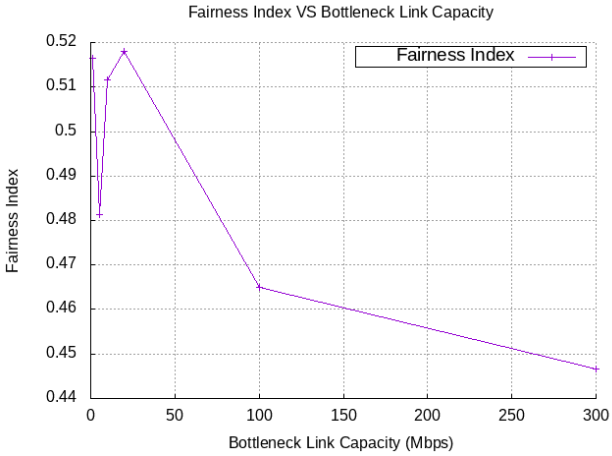
- Throughput of TCP-AReno still increases more than TCP-NewReno. But this is lower than the increase shown by TCP-HighSpeed when decreasing packet loss rate.
- In both cases the value of TCP-NewReno throughput seems to be the same. Thus it seems TCP-AReno doesn't show more friendliness in lossy environments compared to TCP-HighSpeed.

For the first case where no of leaf is 10:

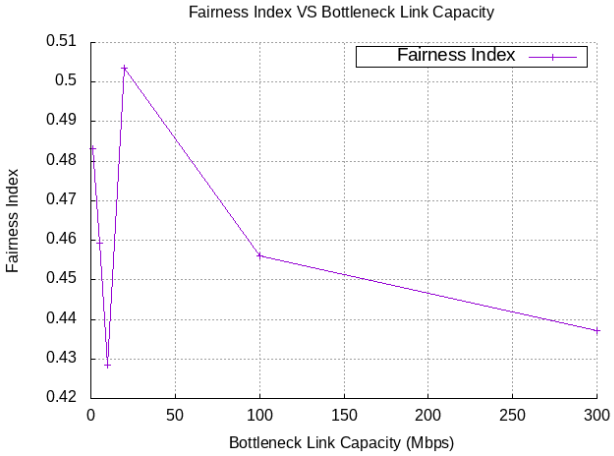
- In this case, the increase of TCP-AReno is more than the increase of TCP-HighSpeed as loss rate decreases. But when loss increases, they both show the same level of throughput.
- Again, in both cases the value of TCP-NewReno throughput seems to be the same.

Thus TCP-AReno doesn't seem to be more friendly towards TCP-NewReno in a lossy environment. But it definitely is more friendly in high speed environments.

6.3.3 Jain's Fairness Index vs Bottleneck Link Capacity in Co-existing flows

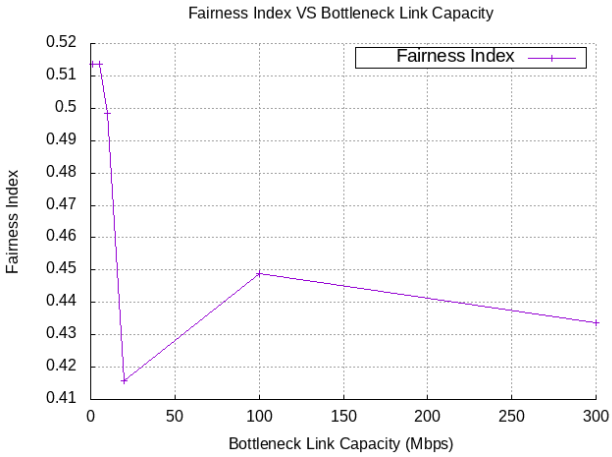


(a) TCP-NewReno + TCP-AReno

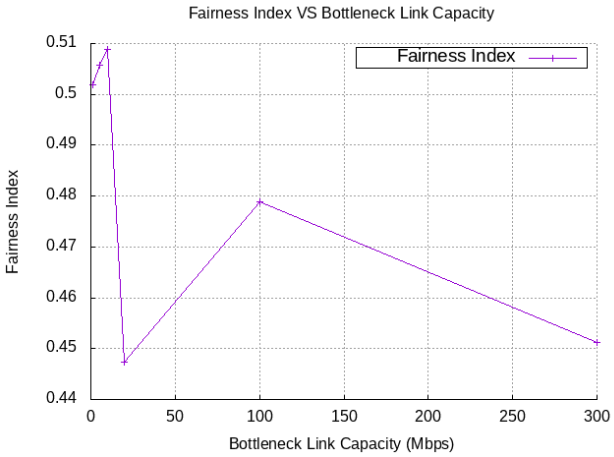


(b) TCP-NewReno + TCP-HighSpeed

Figure 17: **Fairness Index VS Varying bottleneck link capacity(no of Leaf=2)**



(a) TCP-NewReno + TCP-AReno



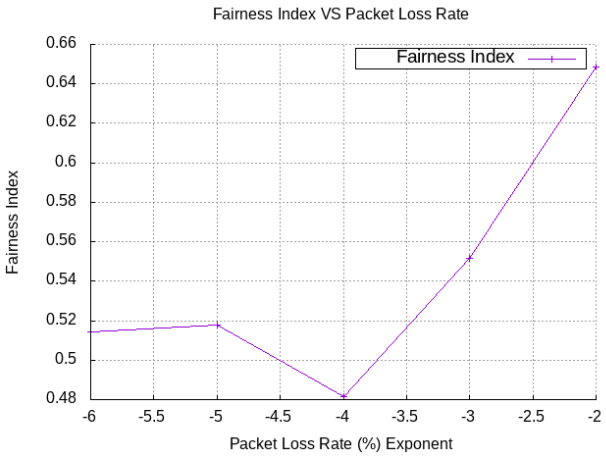
(b) TCP-NewReno + TCP-HighSpeed

Figure 18: **Fairness Index VS Varying Bottleneck Link Capacity(no of Leaf=10)**

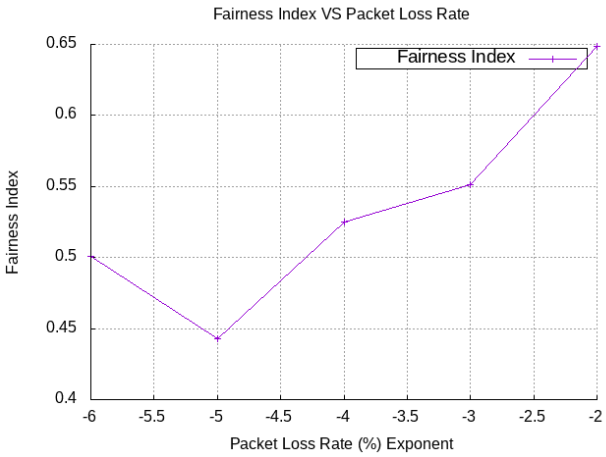
Explanation

For the no of leaf = 2 case, TCP-AReno seems to show more fairness in sharing resources compared to TCP-HighSpeed. But for the no of leaf = 10 case, TCP-HighSpeed seems to be more friendlier towards TCP-NewReno. For both cases, fairness decreased as bottleneck capacity link increased.

6.3.4 Jain's Fairness Index vs Random Packet Loss Rate in Co-existing flows

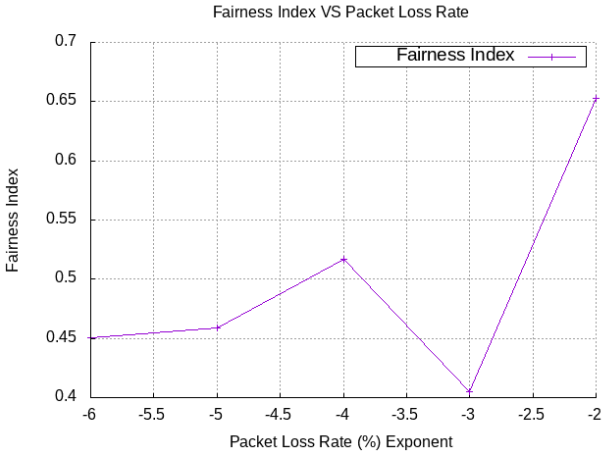


(a) TCP-NewReno + TCP-AReno

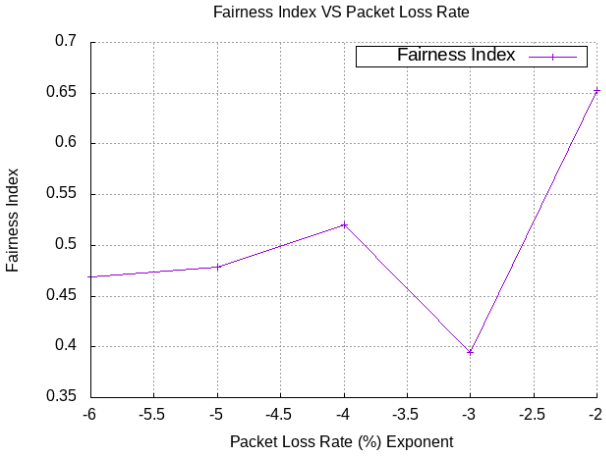


(b) TCP-NewReno + TCP-HighSpeed

Figure 19: **Fairness Index VS Varying Packet Loss Rate(no of Leaf=2)**



(a) TCP-NewReno + TCP-AReno



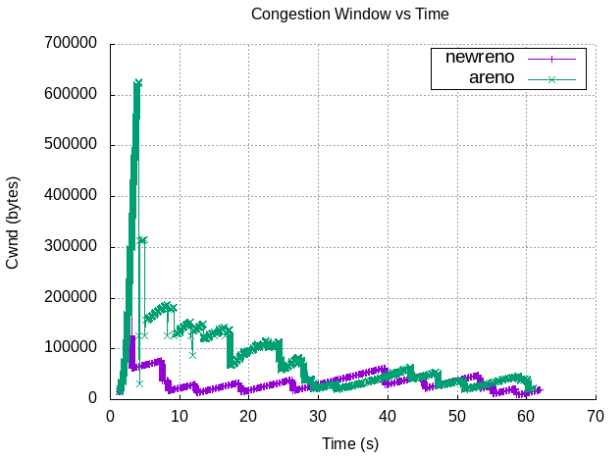
(b) TCP-NewReno + TCP-HighSpeed

Figure 20: **Fairness index VS Varying Packet Loss Rate(no of Leaf=10)**

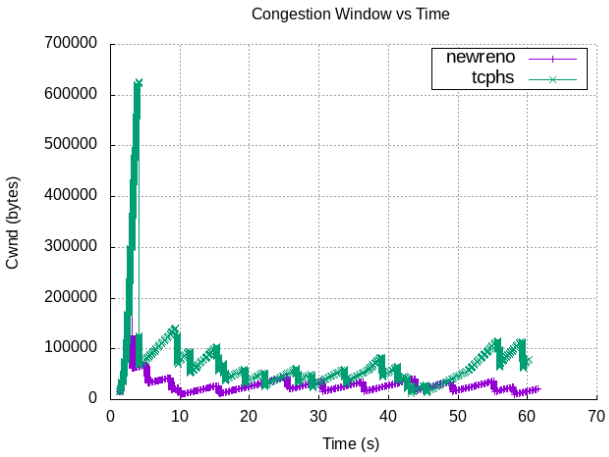
Explanation

Again for the no of leaf = 2 case, TCP-AReno seems to show more fairness in sharing resources compared to TCP-HighSpeed. But for the no of leaf = 10 case, TCP-HighSpeed seems to be more friendlier towards TCP-NewReno in low loss rate. In high loss case, both shows the same amount of fairness. For both algorithms, fairness increased as packet loss rate increased.

6.3.5 Congestion Window Comparison



(a) TCP-NewReno + TCP-AReno



(b) TCP-NewReno + TCP-HighSpeed

Figure 21: **Congestion Window vs Time (no of Leaf=2)**

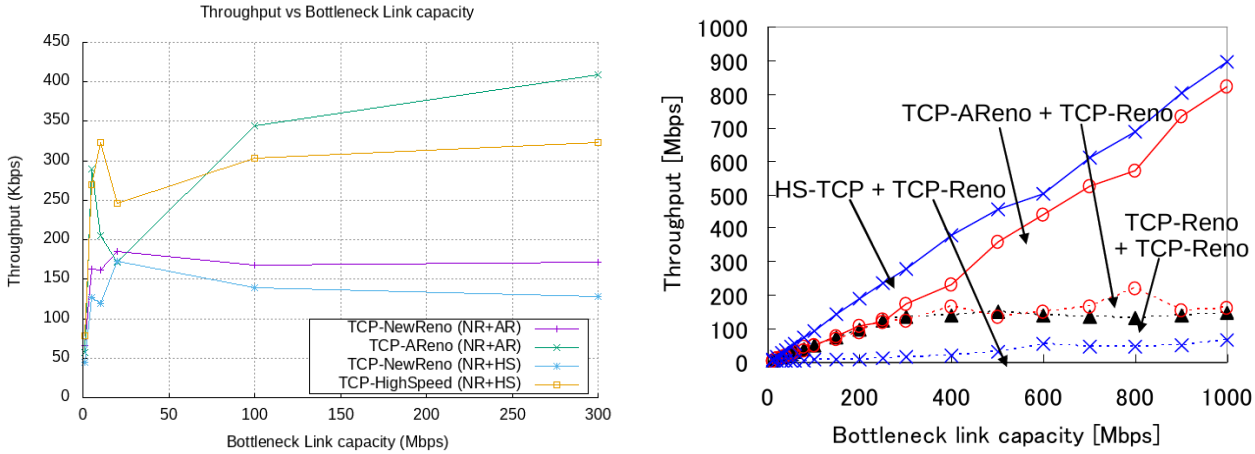
Explanation

It is evident from the graphs that the congestion window in TCP-AReno increases more than the one of TCP-HighSpeed. But the interesting characteristic that is displayed here is, as time increases and congestion increases, the window of TCP-AReno decreases and gives TCP-NewReno the capacity to catch up. Whereas the window of TCP-HighSpeed keeps increasing even though TCP-NewReno isnt getting any fair share of the bandwidth.

7 Analysis of Task B results

7.1 Comparison with paper results

7.1.1 Throughput vs Bottleneck Link Capacity in Co-existing flows



(a) My Simulation

(b) Paper Simulation

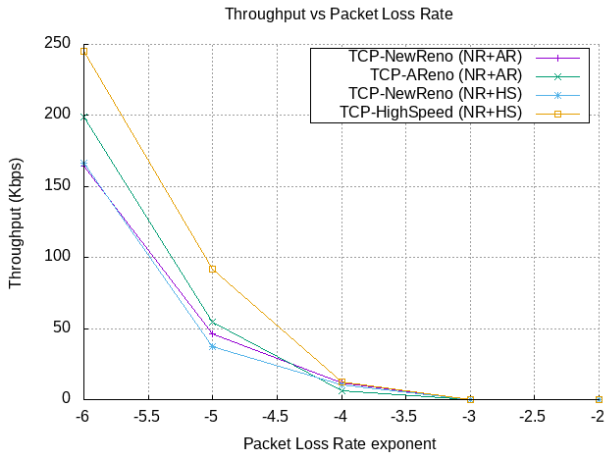
Figure 22: **Throughput VS Varying Bottleneck Link Capacity**

The settings of the two experiments are different in a few parameters, but the overall scenario from the plots are similar. In both simulations, TCP-AReno has higher throughput than TCP-NewReno/TCP-Reno. Furthermore the throughput of TCP-NewReno is higher when co-existing with TCP-Areno compared to TCP-HS (High-Speed).

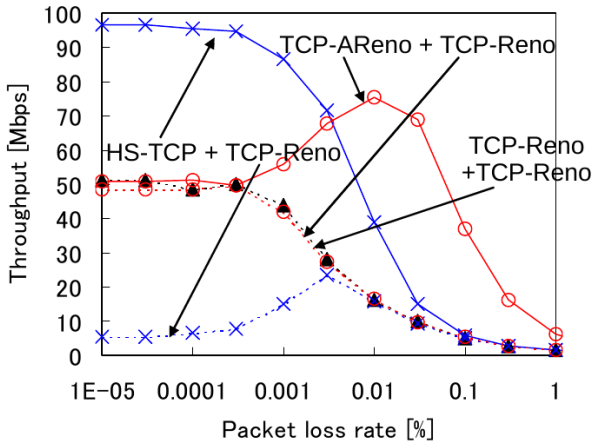
According to [2], a unicast-flow is considered TCP-friendly when it does not reduce the long-term throughput of any co-existent TCP flow more than another TCP flow on the same path would do under the same network conditions. As TCP-AReno instead increases the flow of TCP-NewReno compared to TCP-HS, it is friendlier.

Thus, my simulation agrees with the result with that of the paper in the case of high speed environment. Interestingly, in my implementation the throughput of TCP-AReno is even higher than TCP-HS.

7.1.2 Throughput vs Random Packet Loss Rate in Co-existing flows



(a) My Simulation

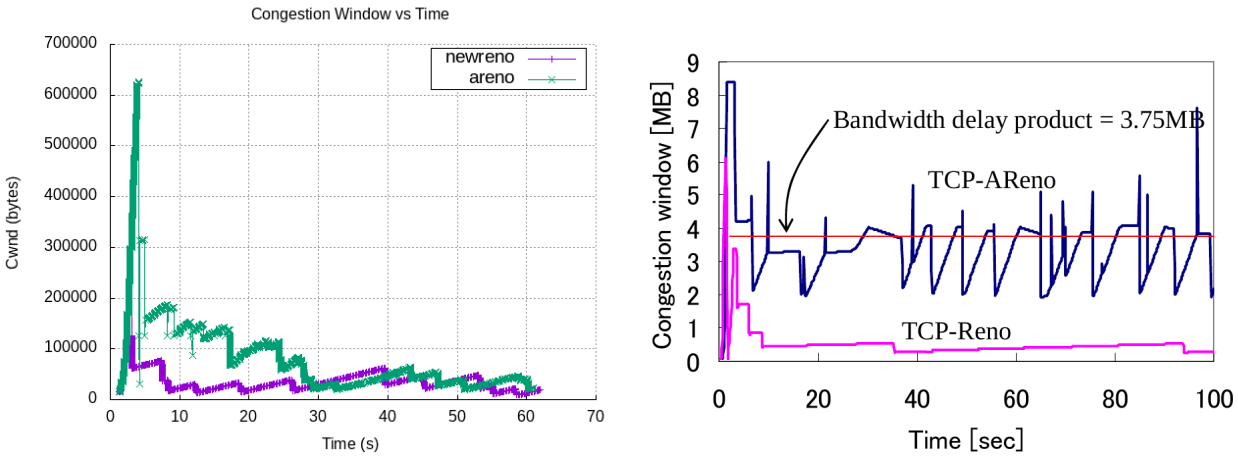


(b) Paper Simulation

Figure 23: **Throughput VS Varying Packet Loss Rate**

Here things get trickier. It seems in my simulation, the throughput of TCP-AReno never exceeds that of TCP-HS when increasing packet loss rate, though it does in the paper's simulation. But in both cases, the TCP-NewReno coexisting with TCP-AReno has a higher throughput when loss increases. Thus this agrees with the fact that TCP-AReno is still friendly in lossy environment. The fact that the throughput is lower than TCP-HS is due to the fact that the simulation was done with bottleneck link capacity being 10Mbps, whereas the paper simulation was done with the capacity being 100Mbps.

7.1.3 Congestion Window Comparison



(a) My Simulation (b) Paper Laboratory Simulation

Figure 24: **Congestion Window vs Time**

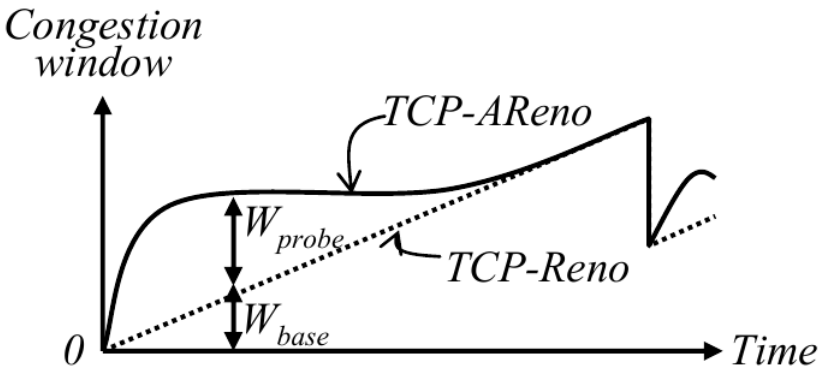


Figure 25: Theoretical shape of congestion window

The paper provided the congestion window change based on a physical laboratory experiment which cannot be done in the NS3 simulator, so a direct comparison is invalid. But we can get some ideas from the shape of the windows. In both cases the increase of the TCP-AReno window in the congestion avoidance phase is more than the increase of the window for TCP-Reno/TCP-NewReno. Both conform to the theoretical shape of the congestion window for TCP-AReno during congestion avoidance phase. Thus, it can be inferred from these observations that my implementation is a correct one.

7.2 Difference between the implementations

There are a few implementation changes from that of the paper which is listed below:

- The paper varied bottleneck link capacity from 100Mbps to 1Gbps. But due to some limitations of NS3's **Point to point Dumbbell Topology**, my implementation varies link capacity from 1Mbps to 300Mbps.
- Therefore I needed to tune a scaling factor M used in the function **EstimateIncWind** referred in Section 5. This value was tuned by the paper authors for their own network through trial and error, thus this shouldn't effect the result.
- Friendliness and co-existence experiments were done with TCP-NewReno instead of TCP-Reno from the paper. TCP-NewReno differs slightly from TCP-Reno and shares a large amount of characteristic similarities, thus the result of the my simulation should also apply to TCP-Reno as well.

8 Summary

From the simulations we can say that TCP-Adapative Reno has higher throughput then TCP-NewReno, is efficient in high network environments and is friendly towards TCP-NewReno and its variants. From the results, it cannot be said for certain whether TCP-Adaptive Reno has high fairness, but we can safely say it has high friendliness compared to TCP-HighSpeed. Thus my results agree with these claims from the paper.

References

[1] H. Shimonishi and T. Murase. "Improving efficiency-friendliness tradeoffs of TCP congestion control algorithm". In: *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.* (2005). DOI: 10.1109/glocom.2005.1577631.

[2] J. Widmer, R. Denda, and M. Mauve. "A survey on TCP-friendly congestion control". In: *IEEE Network* 15.3 (2001), pp. 28–37. DOI: 10.1109/65.923938.