CSE 322
ns-3 Project

A report on

TCP Faster Recovery Algorithm for Congestion Control

by

Ayan Antik Khan

1705036 (A2)

Supervisor

Md. Tareq Mahmood

Lecturer, Department of CSE

Bangladesh University of Engineering and Technology

February, 2022

Network Topologies Under Simulation

Task A(Part 1) and Task B

Wireless High Rate(802.11):

A dumbbell topology was used with Two Wireless Access Points. Each of these Access Points were connected by a pointTopoint link. These Access Point Nodes have networks of their own having a varied number of nodes ranging from 8 to 49 in each side.

In the modified algorithm. Nodes are fixed on each side of the AP nodes. Each AP node has 4 stationary nodes connected in a wireless network.

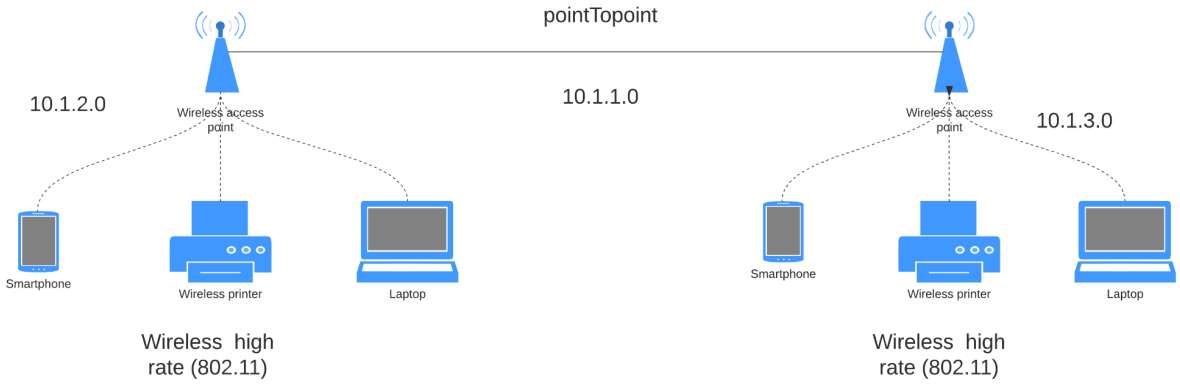

Fig: Topology used in Wireless High Rate

Task A(Part 2)

Wireless Low Rate(802.15.4):

A mesh topology was used. Each of the nodes can connect to any other nodes in the PAN network. Number of nodes varied from 20 to 100.

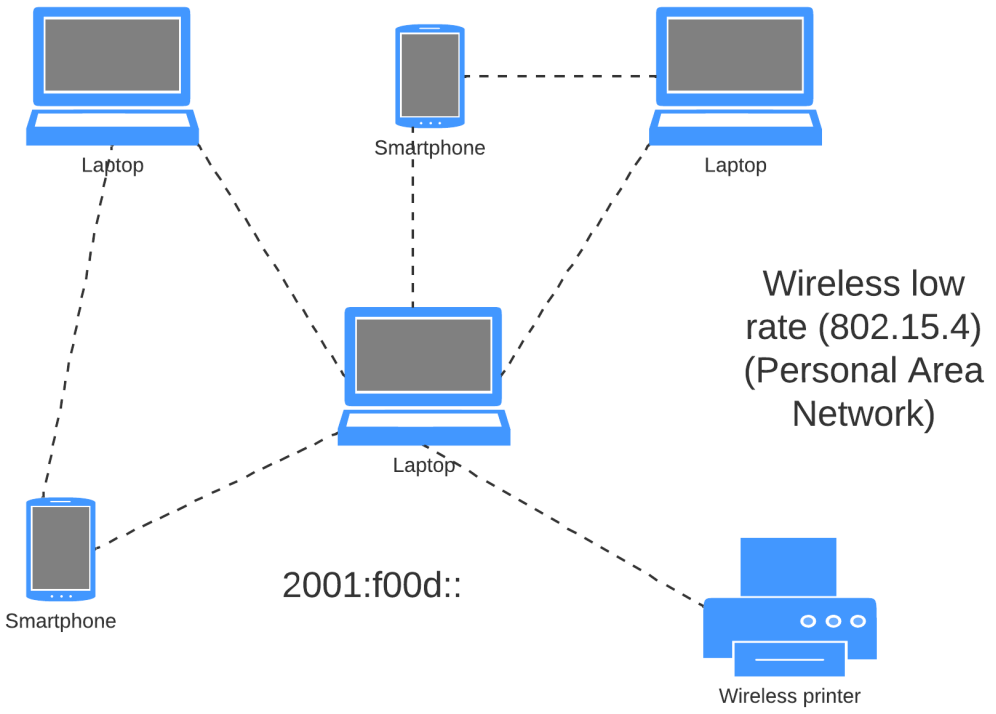

Fig: Topology used in Wireless Low Rate

Parameters Under Variation

In Wireless High Rate(802.11) -

- Number of Nodes
- Number of Flows in the Network
- Number of Packets sent Per second
- Coverage Area (Static Nodes)

In Wireless Low Rate(802.15.4) -

- Number of Nodes
- Number of Flows in the Network
- Number of Packets sent Per second
- Speed of Nodes(Mobile Nodes)

In Modified Algorithm (Wireless High Rate) -

- Number of Packets sent Per second

Overview of the proposed Algorithm

My proposed algorithm is a congestion control algorithm “TCP Faster Recovery”. The algorithm is an improvement upon the TCP Reno algorithm.

The algorithm proposes a new way to calculate the Slow Start Threshold. Upon finding a “Triple Dupack”, TCP Reno algorithm sets the *ssthresh* to *cwind*/2 i.e half the output rate when a triple dupack was found or the timer expired.

TCP-Faster Recovery algorithm has a cleverer approach, instead of “blindly” reducing the congestion window, it uses a **bandwidth estimate** value to set the new *ssthresh* and compute the congestion window.

Estimation of bandwidth:

TCP-Faster Recovery estimates the bandwidth upon receiving each ACK. This bandwidth estimation “ideally” converges into the actual bandwidth between the source and destination. The pseudocode:

```
if (ACK is received) {
 sample_BWE = pkt_size*8/(now - lastacktime);
 BWE = BWE*alpha + sample_BWE*(1 - alpha);
}
```

Calculation of *ssthresh*:

The estimated bandwidth is later used to calculate the *ssthresh.* Pseudocode:

- triple duplicate ACKS:
  $ssthresh = (BWE * RTT_{min})/a$
  $CWIN = ssthresh$
- coarse timeout expiration:
  $ssthresh = (BWE * RTT_{min})/a$
  $CWIN = 1$

Congestion Avoidance:

In congestion avoidance state, the GFR-TCP or “Gradual Faster Recovery TCP” algorithm improves upon the TCP Reno algorithm. It eliminates the linear increase of the congestion window and introduces a way to recognize when the output can be safely increased. The algorithm uses the estimated bandwidth calculated before to try and increase the *ssthresh* safely so that the network can enter slow start mode and increase the congestion window towards the available bandwidth faster than the linear way.

```
If (CWIN > ssthresh) AND (CWIN < BWE*RTT_min)
then
  ssthresh += (BWE*RTT_min-ssthresh)/2;
```

By using these improvements, the proposed algorithm performs significantly better than the TCP NewReno algorithm.

Modifications in simulator

In the ns-3 simulator, a number of modifications were made in the hope of implementing the TCP-FR algorithm. The details are as follows:

src/internet/model/TCPFrGfr.h

Variables:

- **double m_alpha -** Smoothing factor used in Bandwidth estimation. Value = 0.8
- **uint m_a -** Reduction factor used during *ssthresh* calculation. Value for small TCP networks: 1, Large TCP networks: 2
- **double m_BWE -** Bandwidth Estimate
- **Time m_lastacktime -** Time of previous packet acknowledgement
- **Time m_minRtt -** Smallest RTT recorded by TCP for this specific connection
- **EventId m_avoidance -** Scheduling congestion avoidance event handler

src/internet/model/TCPFrGfr.cc

The algorithm inherits TcpNewReno algorithm implemented in *Tcp-congestion-ops.cc*

Functions:

- **void PktsAcked() -** Called every time an ack is received. Bandwidth estimation according to the paper implemented here

- **void CongestionStateSet() -** Current Congestion State returned during state change. Actions regarding *ssthresh* update done here. Congestion Avoidance event also scheduled here

- **void IncreaseWindow() -** Checking if network should be in avoidance mode or slow start mode

- **void CongestionAvoidance()-** Congestion avoidance actions taken here, Updates carried out in *ssthresh* according to conditions in paper

- **void CongestionAvoidanceEvent() -** Scheduling of congestion avoidance event according to paper

Results with Graph

Task A (Wireless High Rate Static-802.11)

- Varying the number of Flows (10, 20, 30, 40, 50):

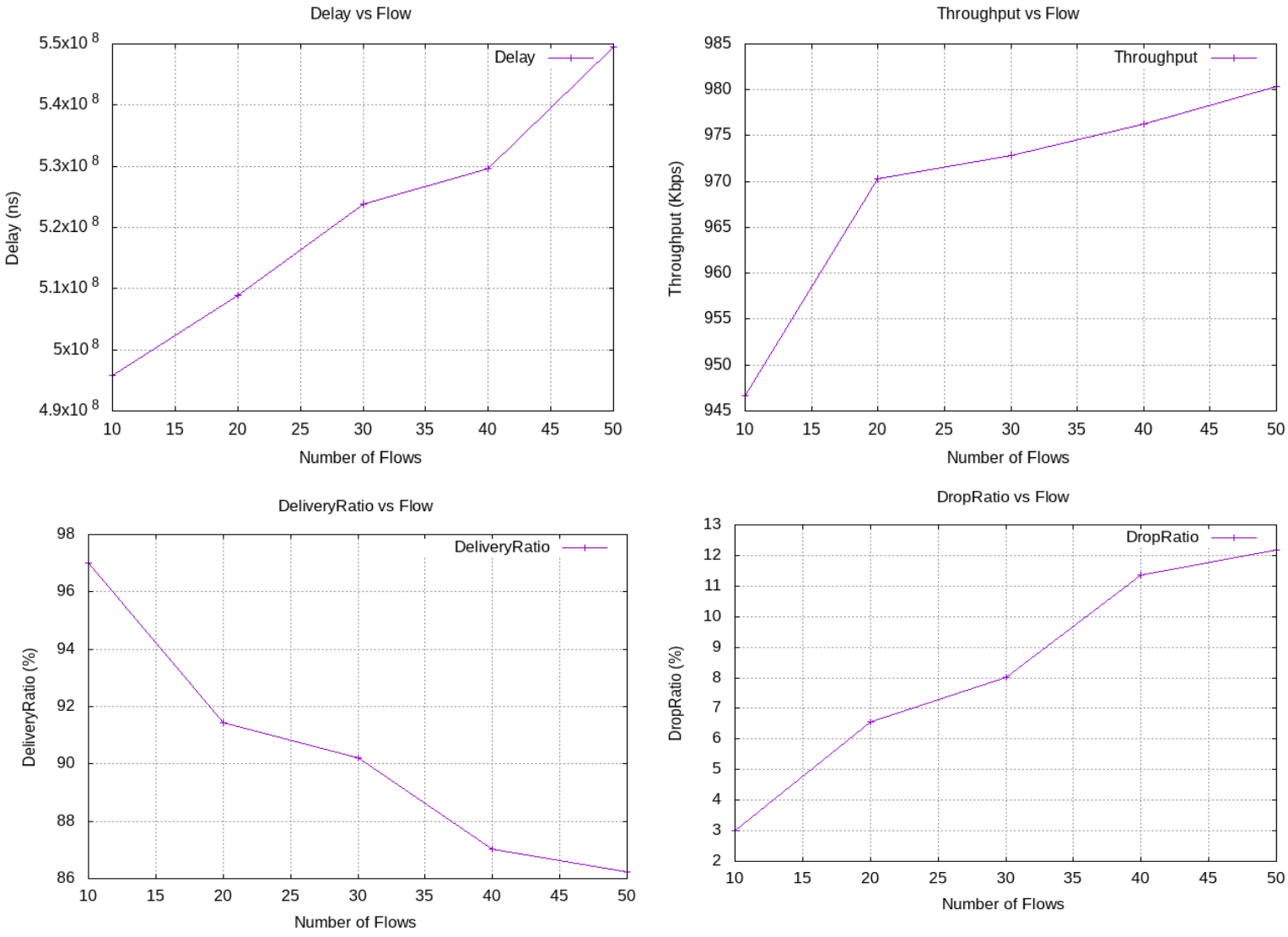

Fig: Flow vs Metrics

Expected results, Network delay and throughput increases with increase of Flow.
Packet delivery ratio decreases due to bottleneck and Drop ratio decreases

- Varying the Number of Nodes (20, 40, 60, 80, 100)

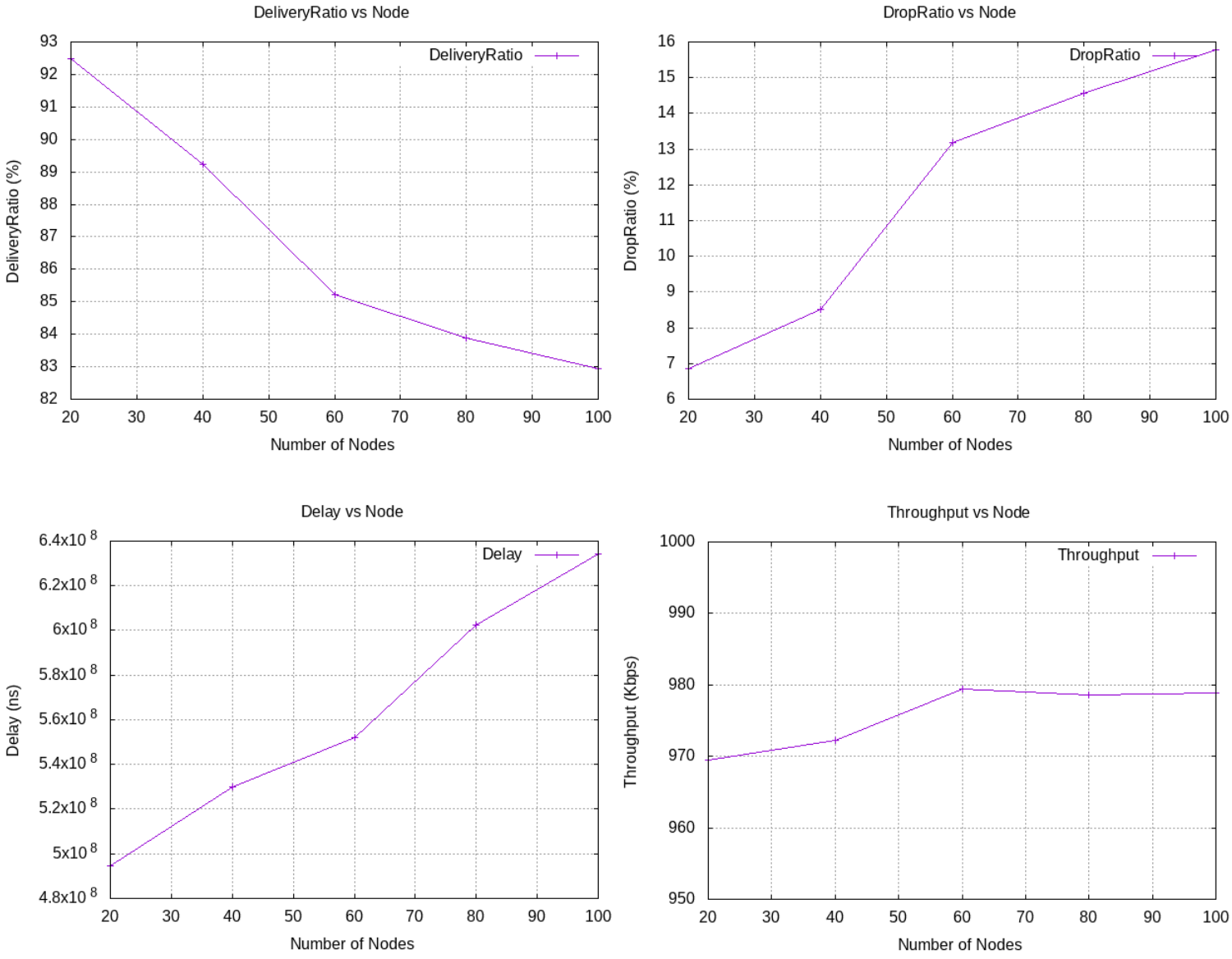

Fig: Nodes vs Metrics

Network throughput remains basically constant since Flows are also increased along with Nodes. Delivery ratio decreases due to bottleneck being constant. Drop ratio thus increases. Delay also increases due to more packets being transmitted.

- Varying the Number of Packets Sent Per Second (100, 200, 300, 400, 500)

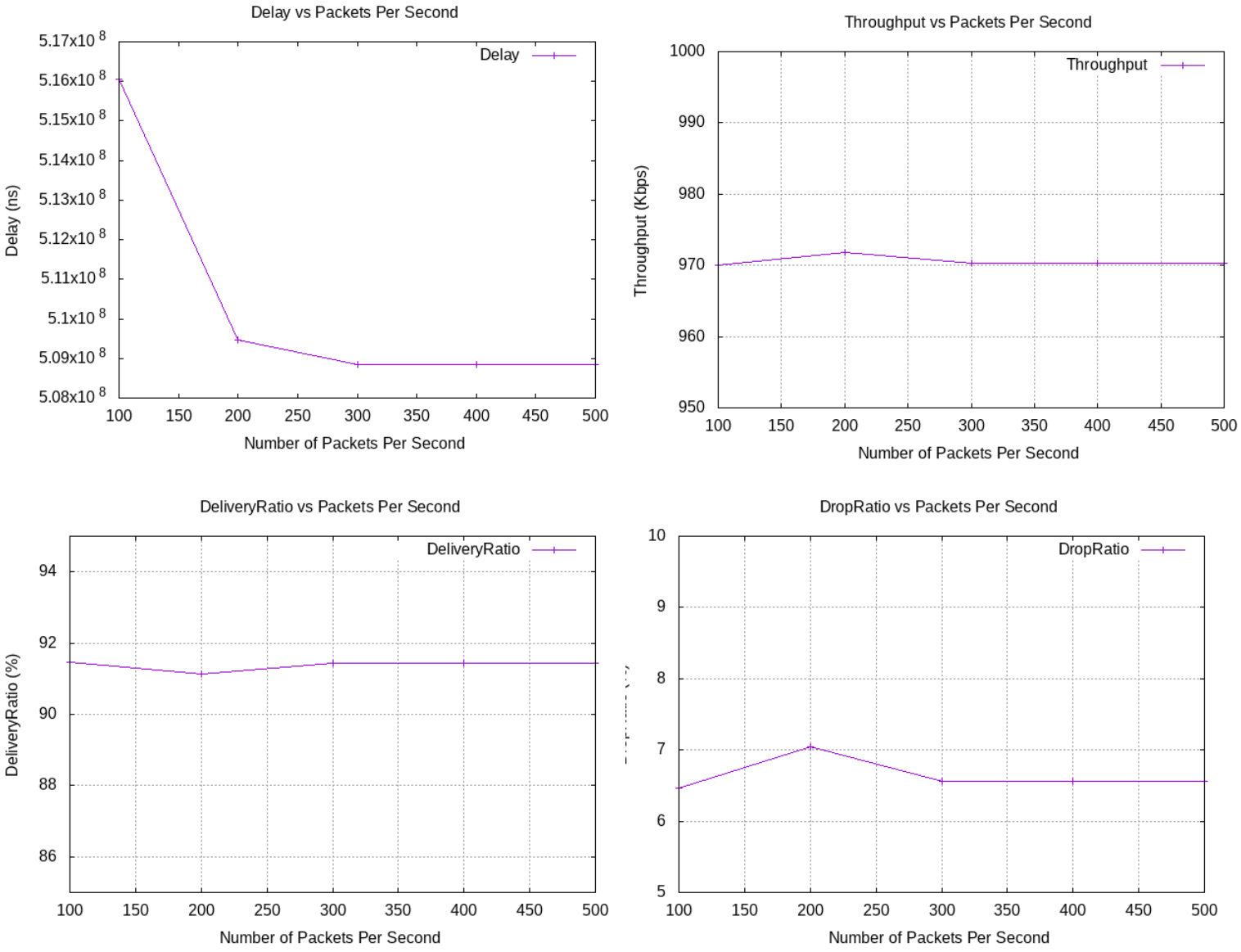

Fig: Packets Per Second vs Metrics

Network throughput remains the same. Same goes for Delivery and Drop ratios. But network delay slightly decreases with increasing packet number.

- Varying the Coverage Area (1, 2, 3, 4, 5)

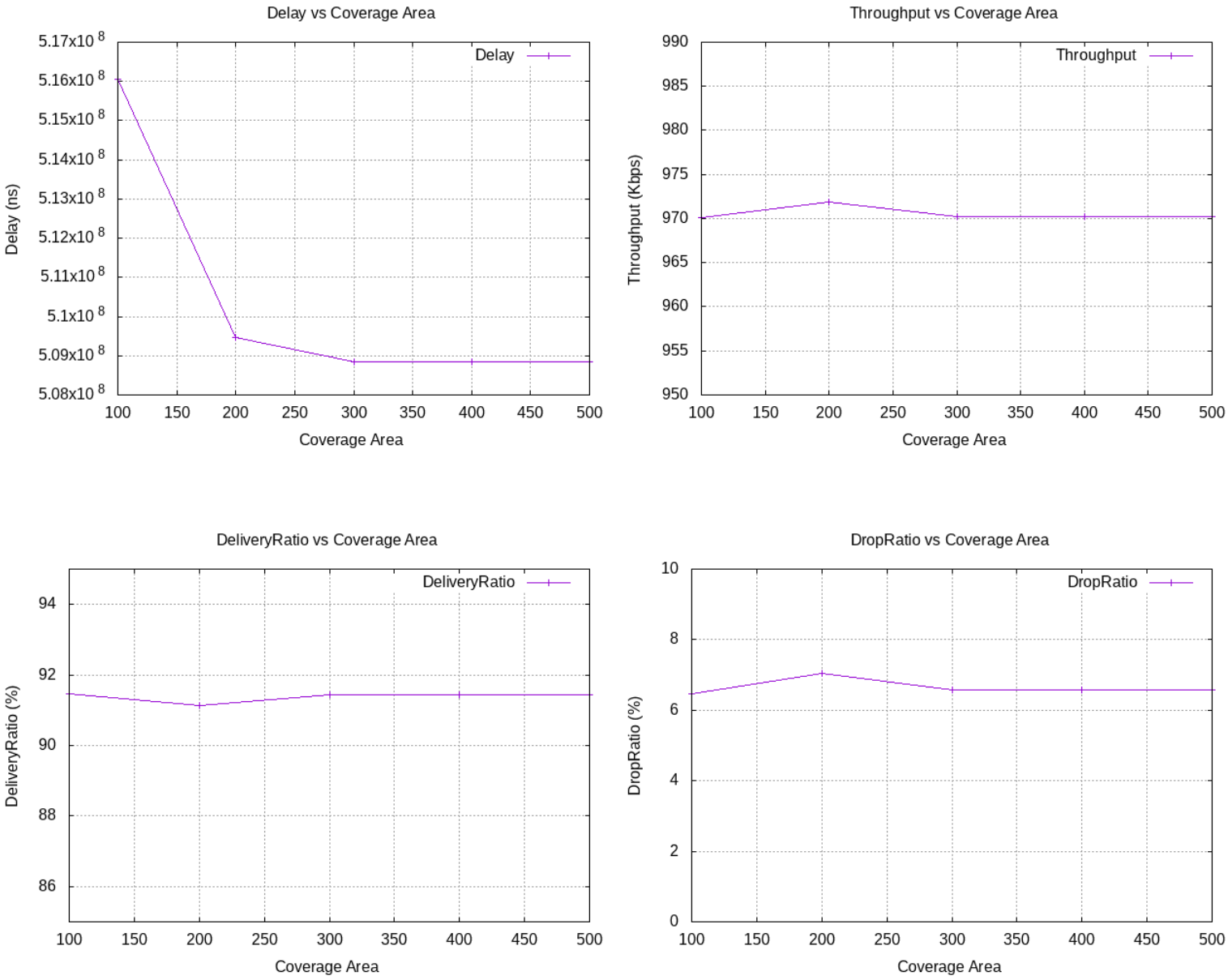

Fig: Coverage Area vs Metrics

The network behavior remains the same throughout the variance of Area. At first some delay is seen, but it decreases when area is increased and nodes are in range to receive packets. Then the network behavior becomes constant.

Wireless Low Rate(802.15.4)

- Varying the Number of Nodes (20, 40, 60, 80, 100)

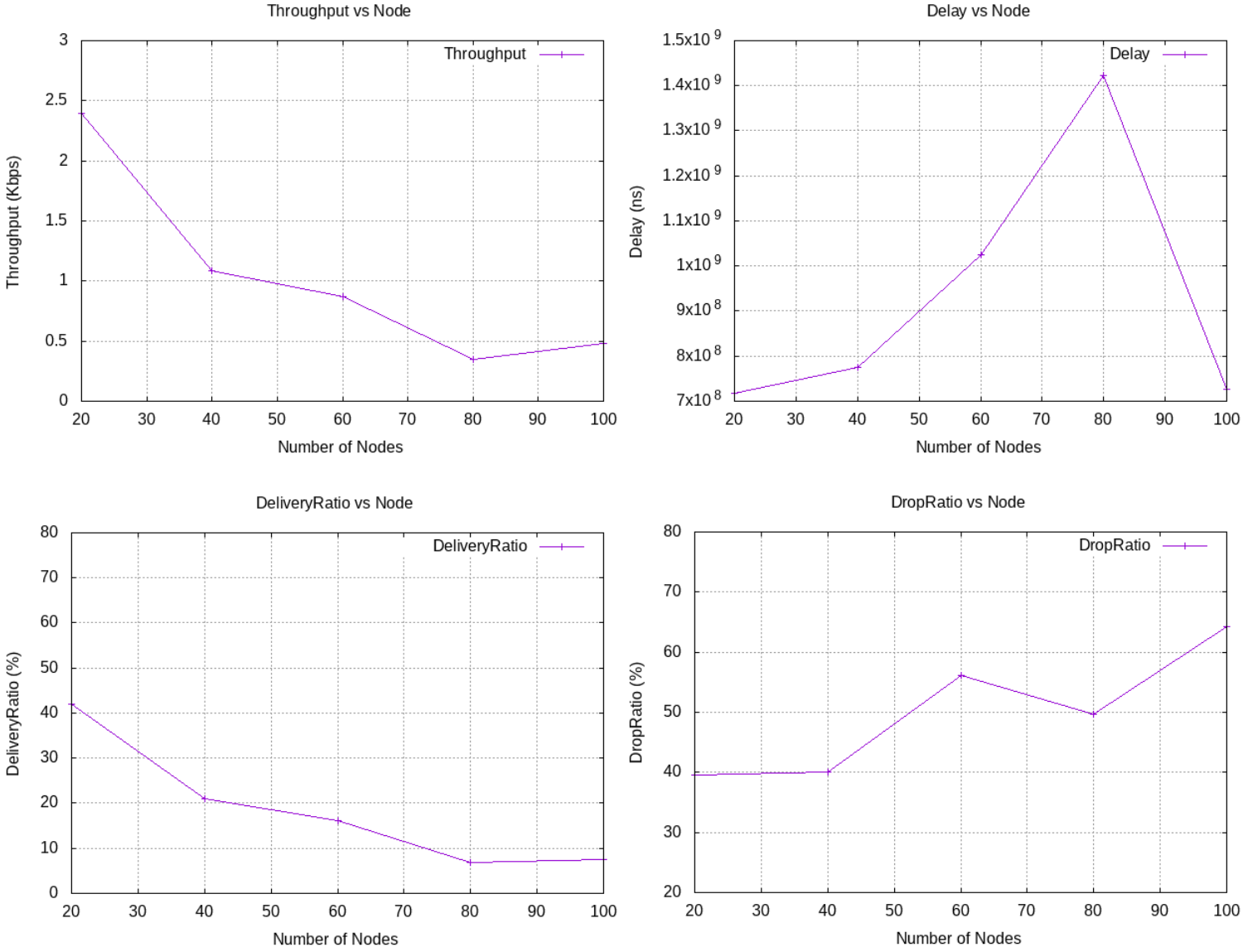

Fig: Nodes vs Metrics

Throughput decreases slightly with increased nodes even though flow is also increasing. Delivery ratio decreases with increased nodes due to increased packet loss in the network. Delay shows somewhat erratic behavior.

- Varying the Number of Flows (10, 20, 30, 40, 50)

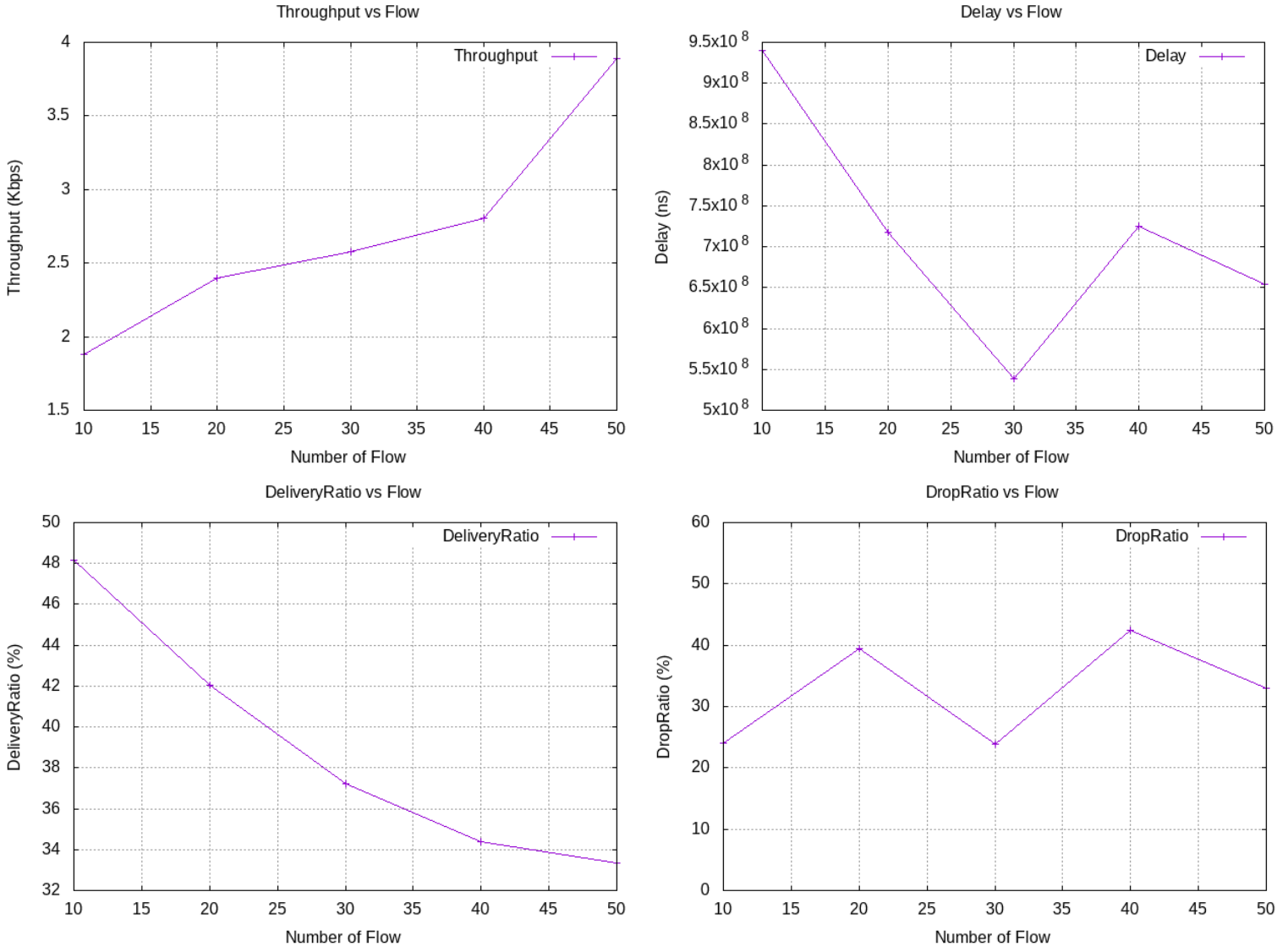

Fig: Flow vs Metrics

Throughput increases with increased number of flows as expected. Delay shows erratic behavior. Delivery ratio keeps decreasing due to increased packet loss.

- Varying the Number of Packets Per Second (100, 200, 300, 400, 500)

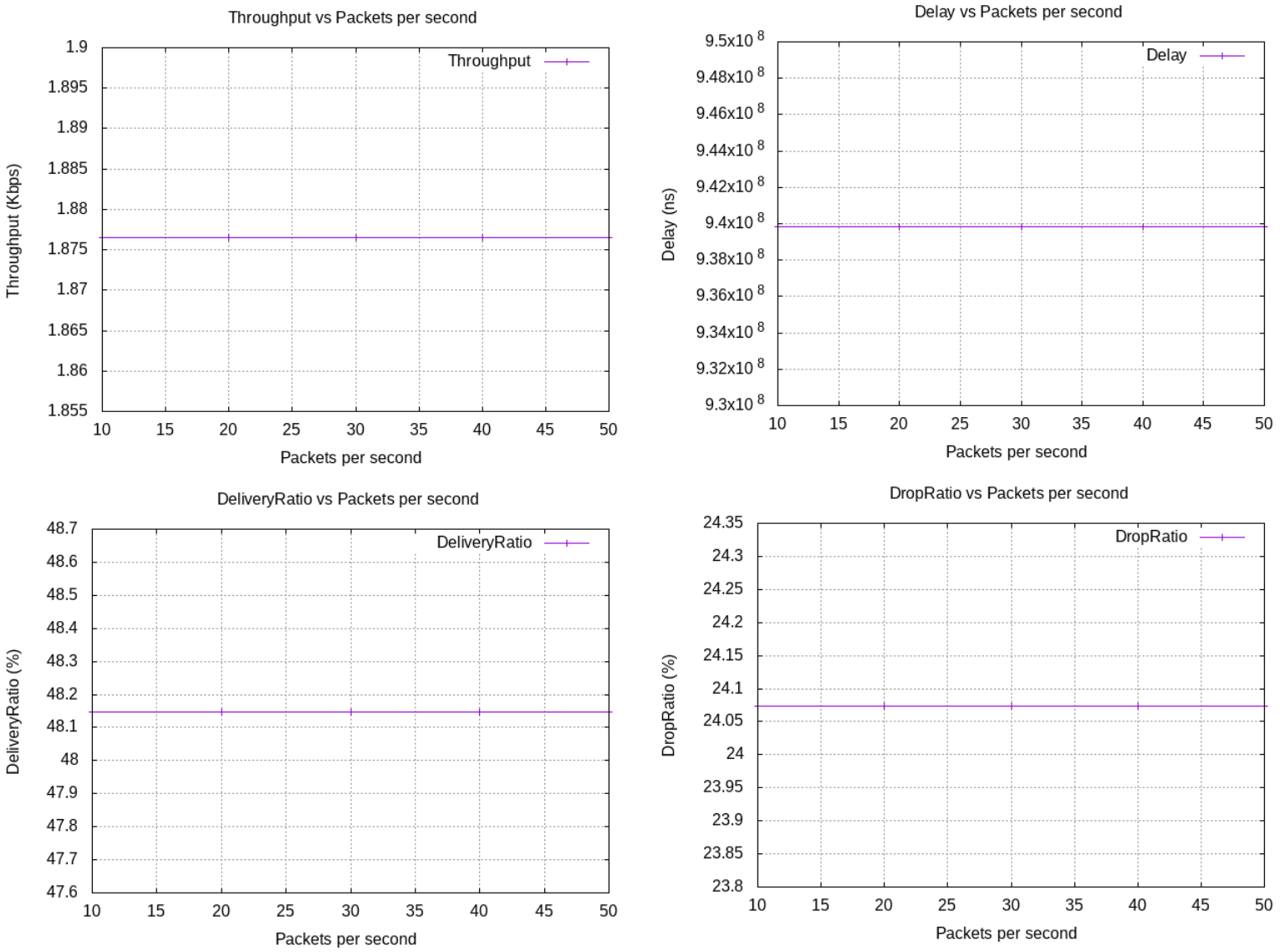

Fig: Packets Per Second vs Metrics

Change in the number of packets per second has almost no effect in the network since the congestion control algorithm takes control and keeps network behavior constant.

- Varying the Speed of nodes (2, 4, 6, 8, 10)

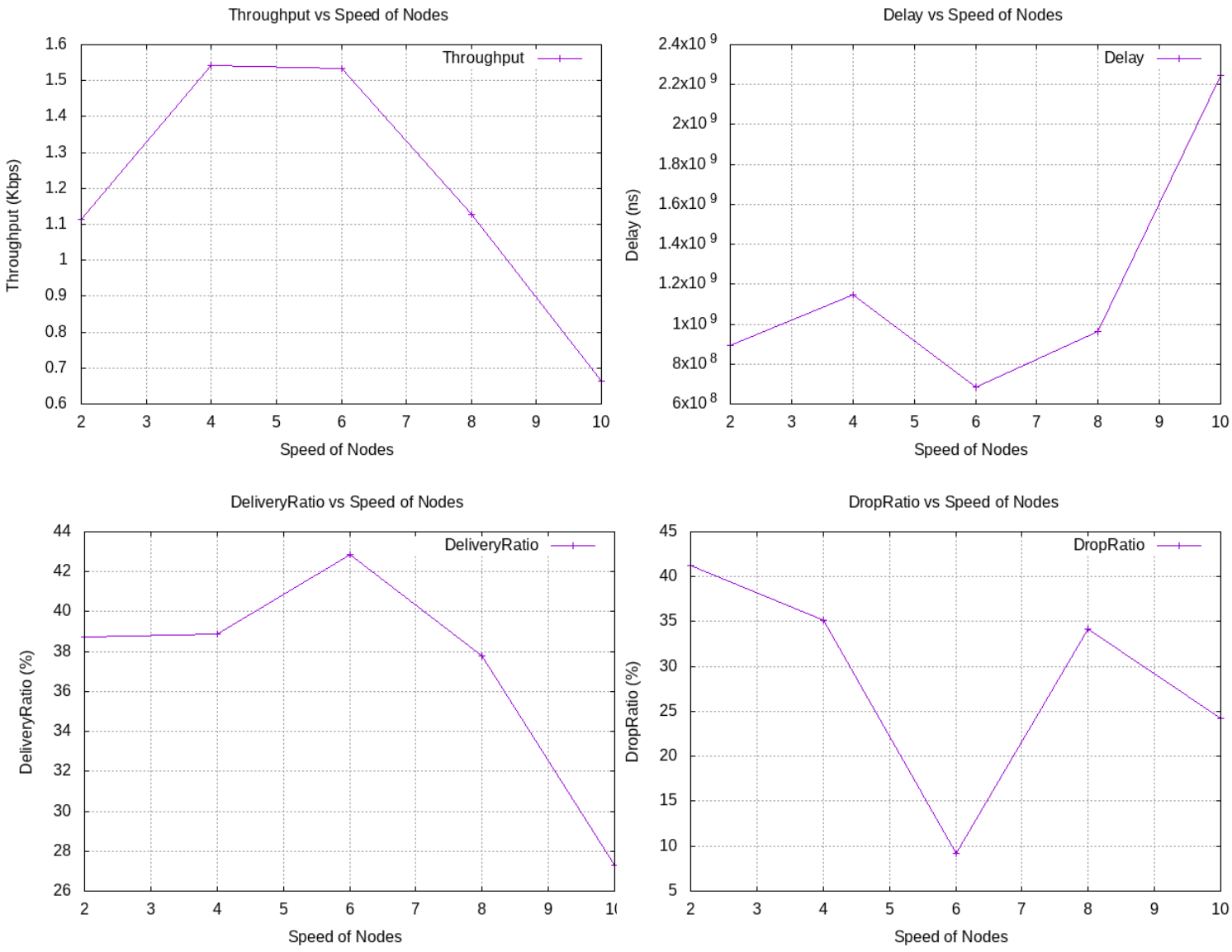

Fig: Speed of Nodes vs Metrics

Network shows erratic behavior with increased mobility of the nodes. But some linearity can be seen in case of throughput which encounters a gradual decline after an initial spike. Delay also increases somewhat linearly.

Task B (Modified Algorithm comparison with TCP NewReno)

- Varying the number of packets per second(100, 200, 300, 400, 500)

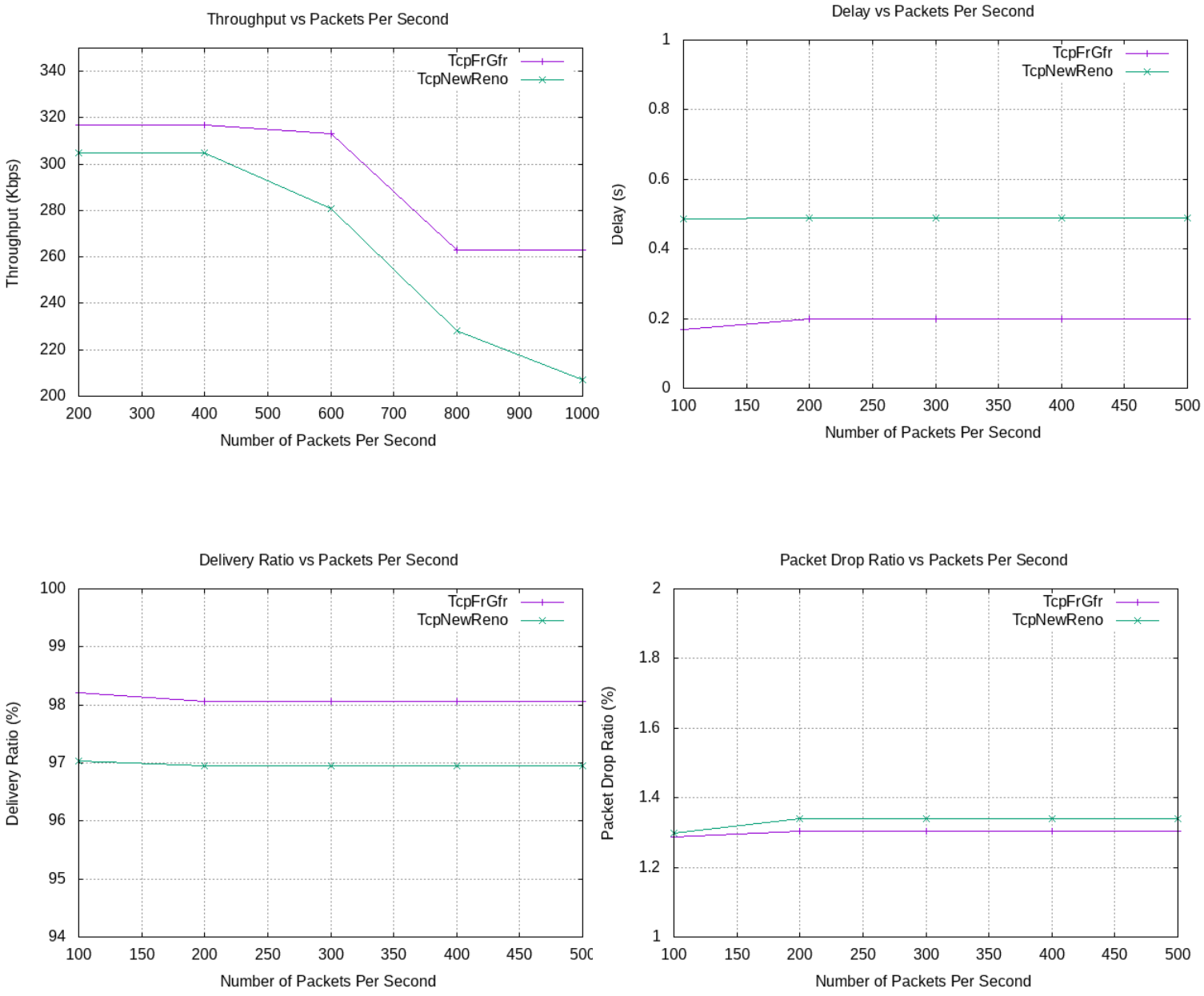

Fig: Comparison of Metrics in Modified TCP Algorithm

TCP Faster Recovery algorithm performs better in cases of delay, packet delivery ratio and packet drop ratio than TCP NewReno. The main reason for this is the intelligent use of estimated bandwidth in place of blind reduction of congestion window.

Under certain conditions, It also gives better Throughput than TCP NewReno. In the case of Throughput calculation, a different topology was used with fewer nodes. TCP Fast Recovery performed significantly better than TCP NewReno in that case. It is believed that this performance could have been further improved with more time and filtering.

Congestion Window of TCP- Faster Recovery

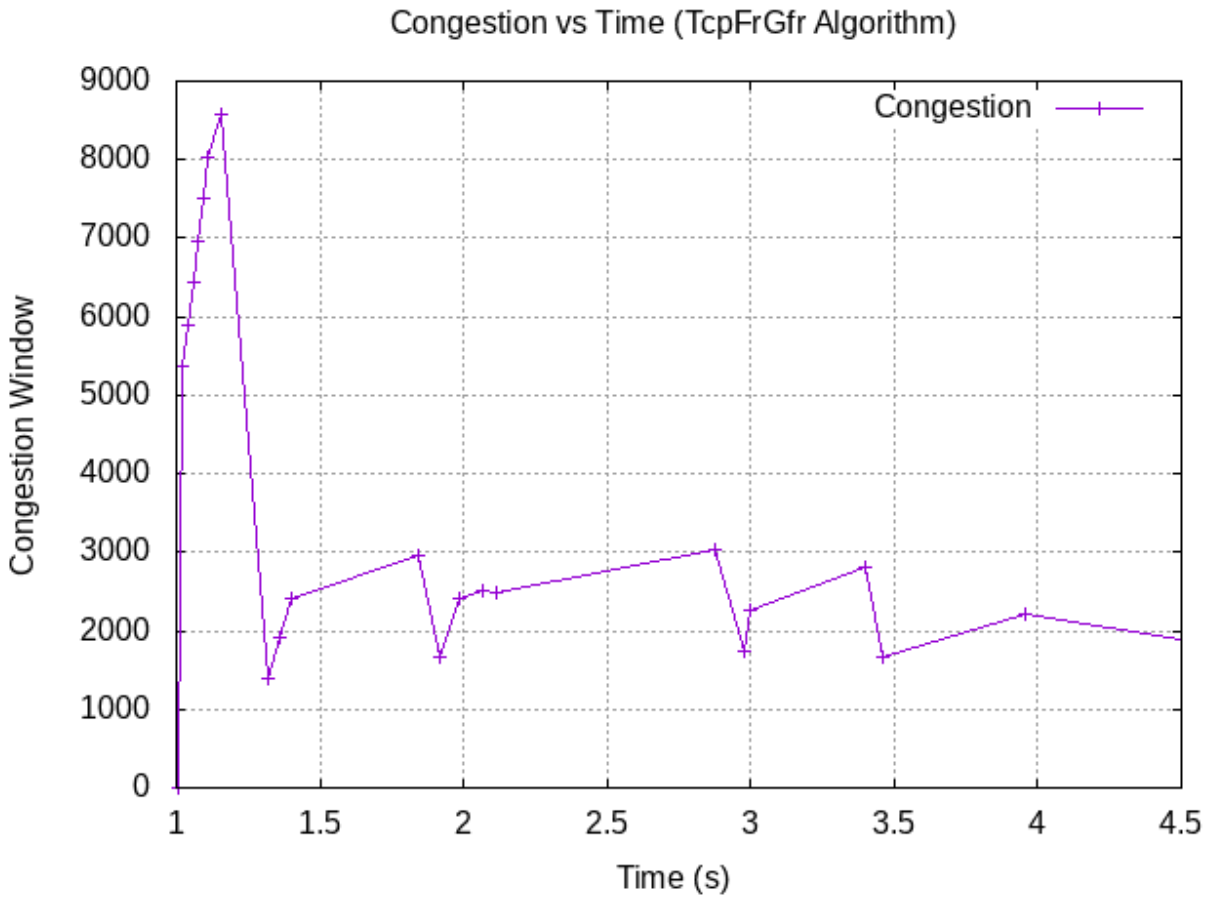

Fig: Congestion Window of Modified TCP Algorithm

The congestion window of TCPFrGfr Algorithm follows TCP NewReno in the slow start phase. Having exponential growth in the starting phase. After reaching the slow start threshold, the window decreases but then is increased within a short time. This increase is much better than the linear increase of the TCP NewReno algorithm. Which is why a better result in the network performance was observed.

The paper also claimed better performance of the modified algorithm under certain complex networks and lossy links. But due to the extreme shortage of time and complexity of the networks, those results could not be simulated.

Summary Findings and Conclusion

Detailed reasonings of the graphs have been included with them. From the high level, some findings:

Task A

Wireless High Rate (802.11) is generally better in performance than Wireless Low Rate (802.15.4) for packet transfer and as a network system. It can be easily claimed from the throughput, packet delivery, delay etc values found from the experiment. Although Wireless Low Rate has a different use case. It is used in Personal Area Networks which can be set up easily and is cost effective. Which is why the shortcomings are accepted in general.

Task B

Congestion control algorithms can always be improved with proper time and effort, the modification of a certain algorithm with proper logical implementation almost always results in better metrics for networks under certain conditions. Such was the case in this project. Proper modification and better implementation with more time would evidently enhance the results found in the project.

Problems in Task B:

During implementation, it was noticed that the network rarely went along with the congestion recovery condition needed to update the slow start threshold. It was probably due to the network setting which maybe was not completely ideal for this part of the algorithm.

The paper also talks about an idea to curb the aggressiveness of the algorithm and to cushion the effects of imprecisely estimated bandwidth. This part of the paper was not that detailed but I tried to carry out an implementation from the vaguely provided idea with Scheduling Event functionality but the network did not seem to have any particular improvement. So this part of the code was later made inactive.

Reference

Casetti, Claudio & Geria, M. & Lee, S.S. & Sanadidi, M.Y.. (2000). TCP with faster recovery. 1. 320 - 324 vol.1. 10.1109/MILCOM.2000.904968.