Improving Multipath TCP Congestion Control

CS 218 Project Final Report

Fall 2017

Cyrus Tabatabai-Yazdi, Zongheng Ma,

Kimberly Chou, Akshay Shetty, Akshay Raman,

Alan Tang, Sean Xiaowen Zhang

CONTENTS

1. Introduction

Today's evolving networks require a rethinking of current protocols. For example, mobile devices have multiple radio channels such as LTE and Wi-Fi. However, in today's Internet design, those multipaths are never used. To gain potential improvements in network performances and to fully utilize the capability provided by the current hardware, previous researchers proposed Multipath TCP (MPTCP) [3, 12]. One of the design challenges of MPTCP is its congestion control algorithm, similar to that of the traditional TCP protocols. In this project, we focused to improve the existing MPTCP congestion control algorithms. We designed and implemented a new algorithm and tested in the ns-3 network simulator.

To gain some insight on how to create an improved algorithm, we evaluated the existing MPTCP algorithms in our simulator. Some existing MPTCP algorithms we looked at include EWTCP, Coupled, Uncoupled, RTT Compensator[14]. For EWTCP, for each ACK that is received on path r, the window of path r, w_r, is increased by a/w_r, where a is 1/sqrt(n) and n is the number of paths. When a loss is detected, the window is cut in half. EWTCP is able to achieve fairness, but it is inefficient with its use of the bandwidth. For the Coupled algorithm, for each ACK received on path r, the window w_r is increased by $1 / w_{total}$, where w_{total} is the total window size. When a loss is detected, the window of path r is decreased by half the total window size. For Uncoupled, it follows the normal TCP congestion control algorithm with additive increase and multiplicative decrease. For RTT Compensator, for each ACK received on subflow r, the window increases by $\min\{a/w_{total}, 1/w_r\}$ where $a = 1/\sqrt{n}$, w_{total} is the total window size, and w_r is the window size of subflow r. It decreases the window size of subflow r by half when a loss is detected. We took inspiration from Compound TCP as well as these existing algorithms to create our own

algorithm, named “CS218_ALGO”. We introduce a delay window, which is a ratio of the best RTT and the last measured RTT. The purpose of this measurement is hasten the recovery of a path that has become less congested. To indicate when a path is less congested, we keep track of the number of consecutive ACKs received. In our algorithm, we set this number to 5. With this addition, we were able to see improvement in congestion control using our algorithm.

2. Motivation

The longevity of TCP has been a testament to its success as the primary transport layer protocol in use on the Internet today. There is no denying it has worked and will continue to work for many years. However, networks today are evolving. The prominence of multi-homed devices as well as data centers with multiple paths has led to the need for a new protocol that can leverage new hardware and software techniques for greater performance and efficiency. Since almost all mobile devices that access the Internet today are multi-homed since meaning they have multiple wireless interfaces to Wi-Fi or cellular networks, MPTCP can take advantage of these multiple interfaces to send traffic out of all interfaces simultaneously in order to increase throughput[12]. The increased throughput is great but without a good congestion control algorithm, havoc can be wrecked on the network. The goal of MPTCP congestion control is to utilize each subflow dependent on the congestion on the path. A layman's approach would be to transfer existing TCP congestion control mechanisms to each of the subflows, where each subflow calculates congestion independently. Since congestion control algorithms for regular TCP try to use their fair share of the available capacity, an MPTCP connection with two subflows would use twice its fair share. In addition, the goal is not to spread traffic equally among all available paths but rather use less congested paths dynamically over time, moving traffic from more congested paths to less congested ones. Thus, when designing a multipath congestion control algorithm, the goal is to give a connection at least as much throughput as a regular TCP connection and a subflow should take no more capacity on a path than it would if it were a normal TCP flow. However, traditional congestion control schemes neglect to factor in wireless network characteristics such

as channel quality or interference. In addition, we want to have faster recovery when a less congested path is detected rather than the slow start probing mechanism employed by TCP. Consequently, in designing our congestion control algorithm, we wanted to examine existing MPTCP algorithms to see their performance in different scenarios, particularly mobile ad-hoc networks. By analyzing the existing algorithms and realizing their shortcomings, we want to develop an algorithm that can improve upon the existing ones in certain aspects like throughput. Taking inspiration from existing TCP congestion control algorithms like Fast TCP and Compound TCP, our algorithm takes into account RTT and the number of consecutive ACKS received to dynamically change the rate at which the congestion window grows to more effectively utilize available network resources.

3. Methods

3.1 Overview

For this project, we mainly deployed our implementation inside ns-3, a open-source network simulator[9]. Ns-3 supports multiple types of networks including wireless like Wi-Fi and LTE. It also allows researchers to develop their own network models and define their own topologies. In addition to ns-3, we utilized a open-source model to simulate MPTCP networks inside ns-3 developed by previous researchers[1-2]. This model included an implementation of MPTCP to allow researchers to test and develop their own experiment using MPTCP. This model is capable of virtualizing multiple subflows through one physical interface. In other words, multiple subflows can easily be created without a one-to-one correspondence with physical interfaces. There are multiple 802.11 networks being created logically, meaning the subflows are not sharing the same channel and congestion happens on orthogonal Wi-Fi channels. Essentially, each node in the topology will have 8 logical interfaces where data can be sent and received. This is all taken care of by the model. Overall, this structure allows for easier testing of algorithms for path management, congestion control, and connection management for a large number of subflows without needing to create a device with multiple interfaces[1-2]. The multiple paths will be simulated due to the fact that the subflows are logically sent out of the same physical interface(different logical) at the same time and may go through different paths because of dynamic routing. However, we acknowledge that this not represent a true multipath as would be found in the real world, but suffices to design and test our own algorithms in a sandbox. We designed and implemented our own congestion control algorithm and several topologies and tested our algorithm along with the existing algorithms on all of the topologies.

3.2 Idea and Solution

3.2.1 Current scenarios

In the paper "Design, implementation and evaluation of congestion control for multipath TCP" , we are given brief overview of existing multipath TCP algorithms[4]. While the majority of the solutions implement and satisfy the sum of throughputs for all subflows of MPTCP is greater than a single TCP connection , the emphasis is more on fairness and reliability. The algorithms try to achieve the fairness goal that sum of all subflows in a link is less than or equal to a single TCP connection throughput experiencing the same loss or congestion. Consider the alpha introduced by the paper which is used by RTT Compensator and Linked Increase algorithm where n is number of subflows

$$alpha = cwnd_{total} * \frac{MAX(\frac{cwnd_i}{rtt^2_i})}{(\sum_{i=1}^{n}(\frac{cwnd_i}{rtt_i}))^2} \quad (1)$$

Equation (1) takes into account the RTT times upon successful acknowledgement. Now if we see cwnd increase using the Linked Increase algorithm,

$$min(\frac{alpha * bytesAcked * MSS_i}{cwnd_{total}}, \frac{bytesAcked * MSS_i}{cwnd_i}) \quad (2)$$

Equation (2) provides great incentive on fairness and wouldn't any subflow to impact throughput of other TCP connections. However, consider a high speed path where there is no congestion and requires fast data transfer. If we follow the above algorithms, the growth of the cwnd is slow and the path can be severely underutilized. If we want to make MPTCP as a mainstream, we need to work on a solution that would mitigate these scenarios.

3.2.2. Delay based mechanism addition

In the paper “A Compound TCP Approach for High-speed and Long Distance Networks”, it was found that only using loss based congestion algorithms have poor path utilization and RTT fairness. On the other hand , a pure delay based congestion control algorithm might be too aggressive on a path. The balance is maintained we include both loss-based (TCP-Reno) and delay based components like below

Window = CWND + DWND

Where

CWND - congestion window based on loss based approach of TCP- Reno

DWND - delay based window introduced by Compound TCP

The inclusion of both the components allows us to have a rapid window increase in the case of underutilized paths and graceful reduction in sending rate once a bottleneck is observed in a link.

3.2.3 Our solution

We have implemented an approach similar to Compound TCP . We have added a delay window to quickly increase the congestion window in the case of an underutilized and uncongested fath path. However we differ in the way , we have implemented the delay window. Compound TCP was designed for single TCP connection but we are implementing it to make use of the MPTCP subflows.

The solution is based on the assumption that continuous ACKs received is an indication of a less congested subflow. Upon a set of runs we did on ns-3, we converged on the idea of using 5 as the ideal value for the least number of continuous ACKs thereafter which we need to add our delay

window to the total window. This way we rapidly recover on a less/uncongested subflow. Otherwise the behavior is similar to normal RTT Compensator or Linked Increase Algorithm.

```
if(ackCount > 5) {
      cwndTotal += DWND;
}
```

To keep in sync with existing protocols, we have kept the reduction algorithm same as others. That is , upon packet loss , the window size is reduced by two .

For subflow r, congestion window Wr is given

$$W_r(new) = \frac{W_r(old)}{2}$$

To determine the value of DWND to be added , we have to decided to consider the following fraction

$$\frac{Best\ RTT}{Last\ Measured\ RTT}$$

where Last Measured RTT is the last measured value of RTT of the subflow in milliseconds.

Best RTT value is the least value of the measured so far for the subflow in milliseconds.

We combine this fraction with the alpha that is used in RTT Compensator and Linked Increase protocols.

$$DWND = alpha * (\frac{Best\ RTT}{Last\ Measured\ RTT})$$

The overall algorithms for congestion window reduction and increase looks like this

1. Congestion Window Reduction upon packet loss.

   $$ackCount = 0$$

   $$W_r = \frac{W_r}{2}$$

2. Congestion Window increase upon ACKs

$ackCount++$; // Increase ack_count upon receiving ACK

$DWND = 0$; // Reset the delay window

$$alpha = cwnd_{total} * \frac{MAX(cwnd_i/rtt_i^2)}{(\sum_{i=1}^{n}(\frac{cwnd_i}{rtt_i}))^2};$$

```
if(ackCount > 5) {
   DWND = alpha * (Best RTT/Last Measured RTT);
}
```

$CWND = alpha + DWND;$

```
if(Last Measured RTT < Best RTT) {
   Best RTT = Last Measured RTT;
}
```

3.2.3.1 Best and worst cases

The best scenerio is when LastMeasuredRTT is greater than or equal to the current BestRTT value. Consider the scenario when LastMeasuredRTT is equal BestRTT

$$\frac{Best\ RTT}{Last\ Measured\ RTT} = 1$$

$$DWND = alpha * 1 = alpha$$

In this case

$$CWND = 2 * alpha$$

The cwnd is increased by twice the amount added in RTT compensator or Linked Increase algorithms.

Worse case when the LastMeasuredRTT value is too high , the path is congested but we are still getting ACKs. In the case the fraction goes to near zero.

$$\frac{Best\ RTT}{Last\ Measured\ RTT} \approx 0$$

$$DWND = alpha * 0 = 0$$

$$CWND = alpha + 0 = alpha$$

As we can we approach the existing RTT Compensator and Linked Increase protocol behavior upon receiving high value of RTT . This helps maintain the fairness aspects of the MPTCP.

3.2.3.2 Fairness considerations

While our solution wants to prevent aggressive fairness , we still want to make sure that under high congestion, we want to maintain the fairness to all subflows and other existing TCP connections.

Following measures have been taken to maintain fairness aspects

1. We add DWND only when the number of ACKs received is greater than 5 . This way the delay window only kicks in when the path is less or not congested. This also decreases the chances of cycling.
2. The value of Last Measured value of RTT provides a good insight into the current status of the link congestion.
3. The algorithm provides similar approach to RTT Compensator and Linked Increase protocols when the RTT values are too high or we are getting repeated packet drops.

3.3 Topologies

We have successfully designed and implemented 2 topologies in ns-3 simulator, and developed two others that encountered problems due to bugs within the ns-3 model and the interaction of MPTCP with regular TCP flows and LTE.

Each topology contains one UDP server node, one MPTCP server node, and several UDP client nodes as well as several MPTCP client nodes. For the MPTCP server node, it runs a packet sink

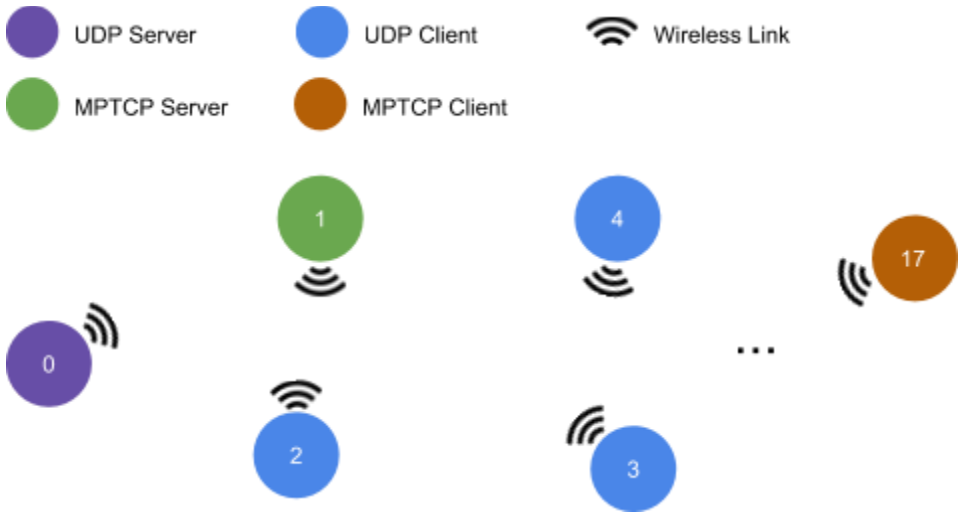
application that receives MPTCP packets. For the MPTCP client nodes, they run bulk-send applications in ns-3. For the UDP nodes, they run simple echo applications in order to generate congestions among the links.

Below are the detailed explanations of the setup of each topology.

3.3.1 Wi-Fi MANET

In this topology, we create a Mobile Ad-hoc Network (MANET). As shown in the graph below, there are 18 nodes in total. There is one UDP server node, one MPTCP server node, 15 UDP client nodes, and one MPTCP client node. Every node is connected with each other via wireless links. We use the ns-3 mobility model here so the nodes are randomly moving. The bandwidth is 5 Mbps. We also tested using a scenario where this is no mobility.

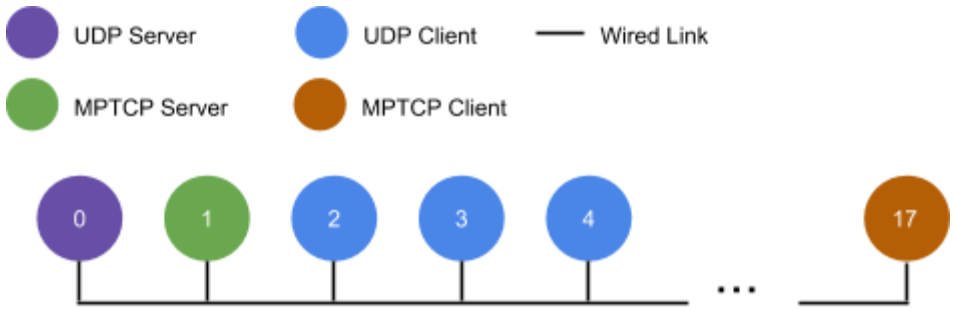
We use nodes that are running UDP clients and sending continuous UDP packets to create congestion in the links, so that we can test our congestion control algorithm.



3.3.2 Wired CSMA

In this topology, we create a wired network in which nodes are sending and receiving packets based on CSMA protocol on a wired link. We use this topology to simulate Ethernet. Since real

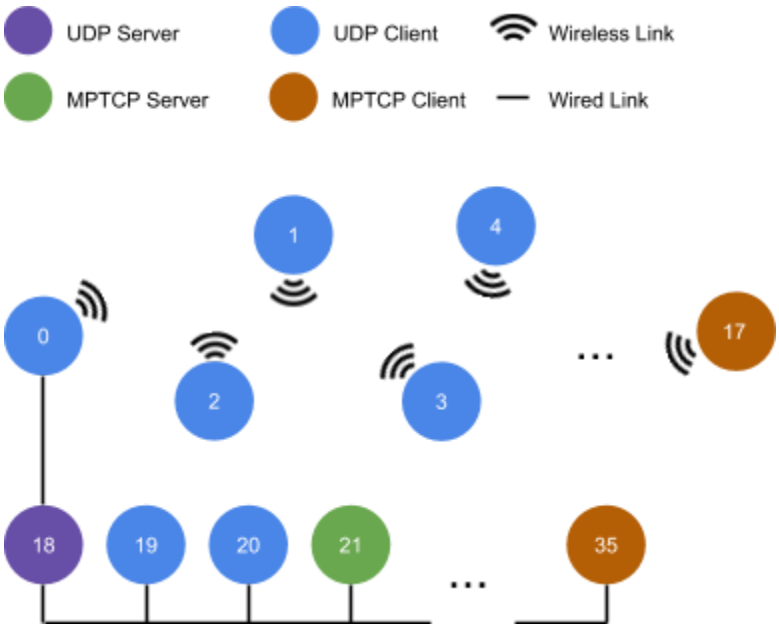
Ethernet is not supported in ns-3, we use CSMA because it behaves similar to Ethernet. Similar to topology *3.3.1*, there are 18 nodes in total. There is one UDP server node, one MPTCP server node, 15 UDP client nodes, and one MPTCP client node.



3.3.3 Wi-Fi and CSMA

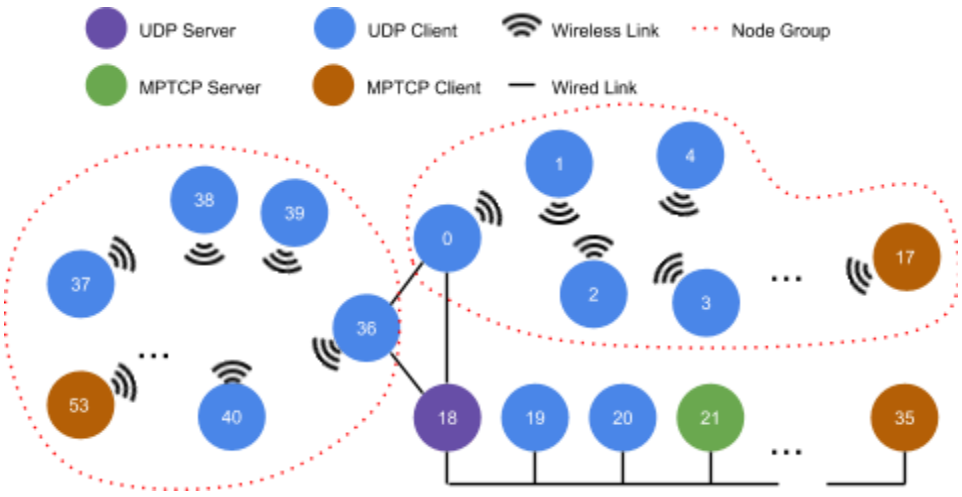
In this topology, we combine the previous two topologies(*3.3.1* and *3.3.2*). We have 36 nodes in total. Similar the previous topologies, we have one UDP server node, one MPTCP server node, and many UDP client nodes (32 in total). Unlike the previous topologies, we have two MPTCP client nodes this time; one Wi-Fi MPTCP client node, and one CSMA MPTCP client node.

We have implemented this topology in ns-3. However, the behaviour of the subflows was unreasonable when we have both Wi-Fi and CSMA in the topology. We have not found the cause of this problem yet and we did not use the results we got from this topology.



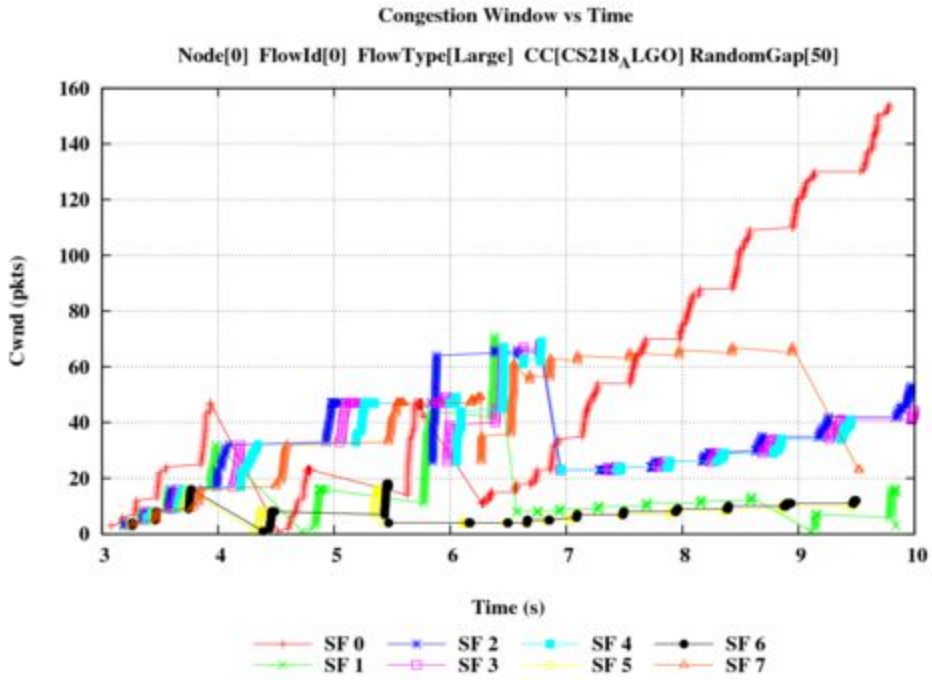
3.3.4 Full Mesh

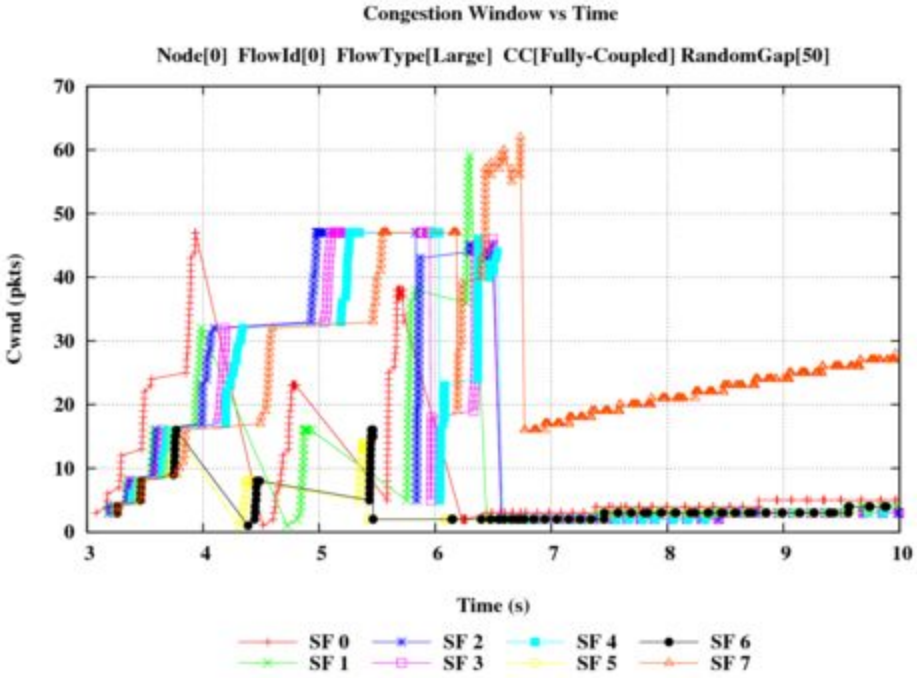
In this topology we added another set of Wi-Fi MANET nodes to topology 3.3.3. There are totally 54 nodes. As shown in the figure blow, there is one node running MPTCP client application in each group of nodes. The MPTCP service application is running in Node 21, which is in the CSMA network. Node 36 and Node 0 servers as access points of the two Wi-Fi node sets.

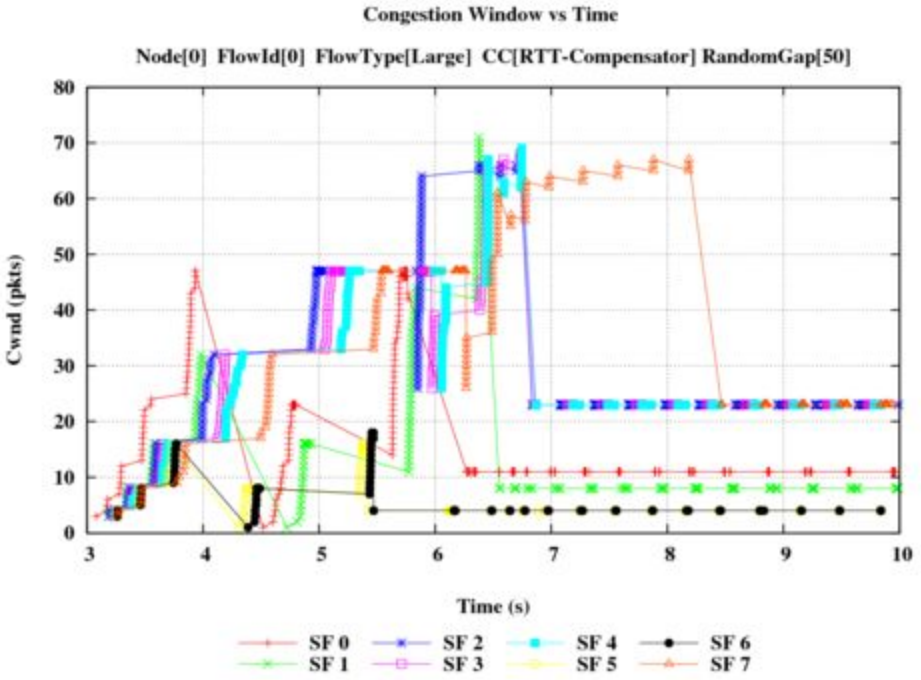


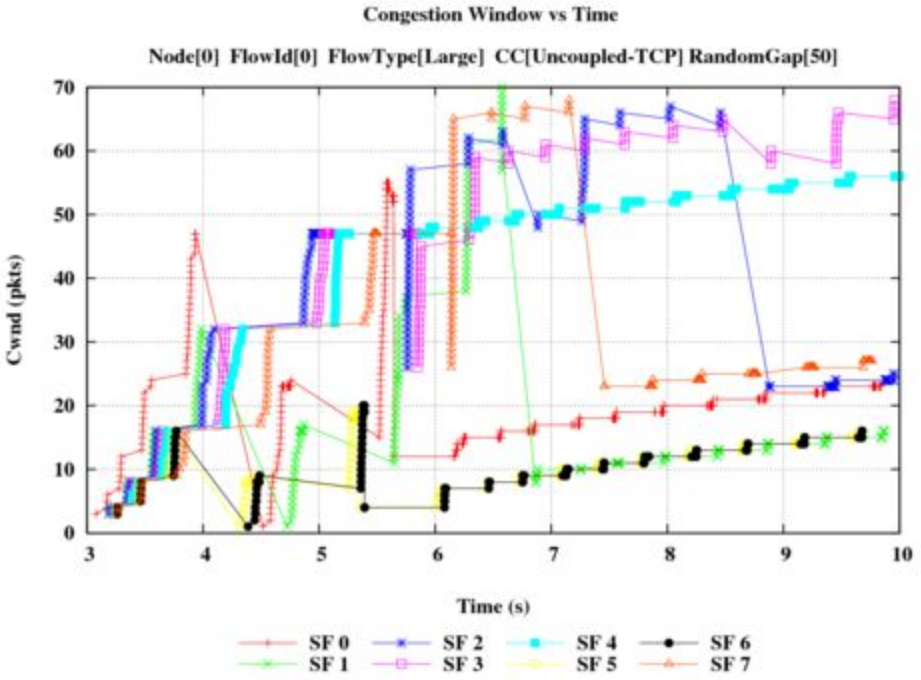
As we can easily see, if a MPTCP client node in the Wi-Fi networks wants to send a packet to the MPTCP server node in the CSMA network, it will have two physical routes to choose from. Similar to topology 3.3.3, we also have the issue here in this topology. We think there is a bug in the MPTCP model for ns-3, but due to time limitations, we have not solved this bug.

4. Results and Evaluations



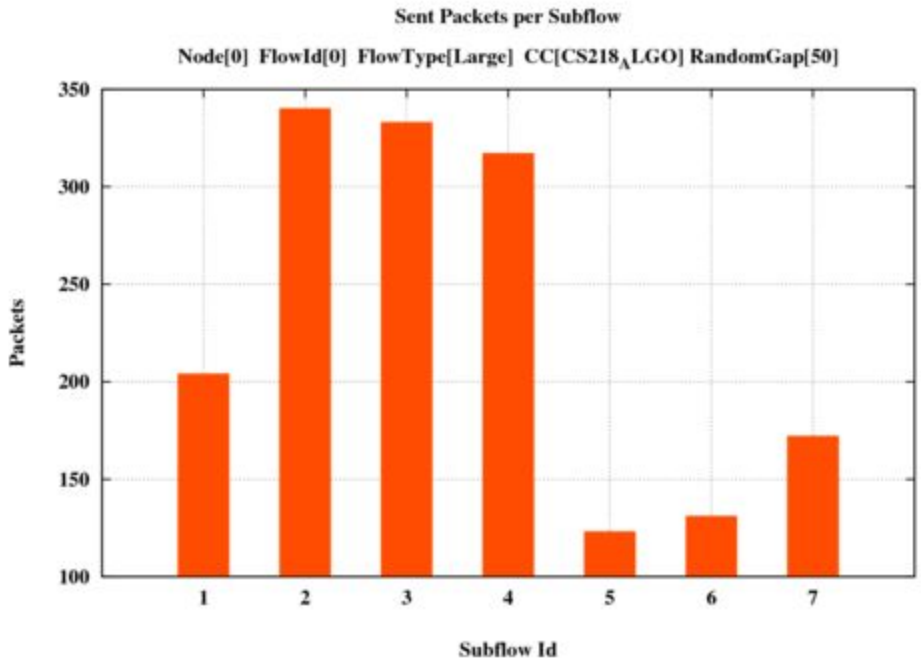


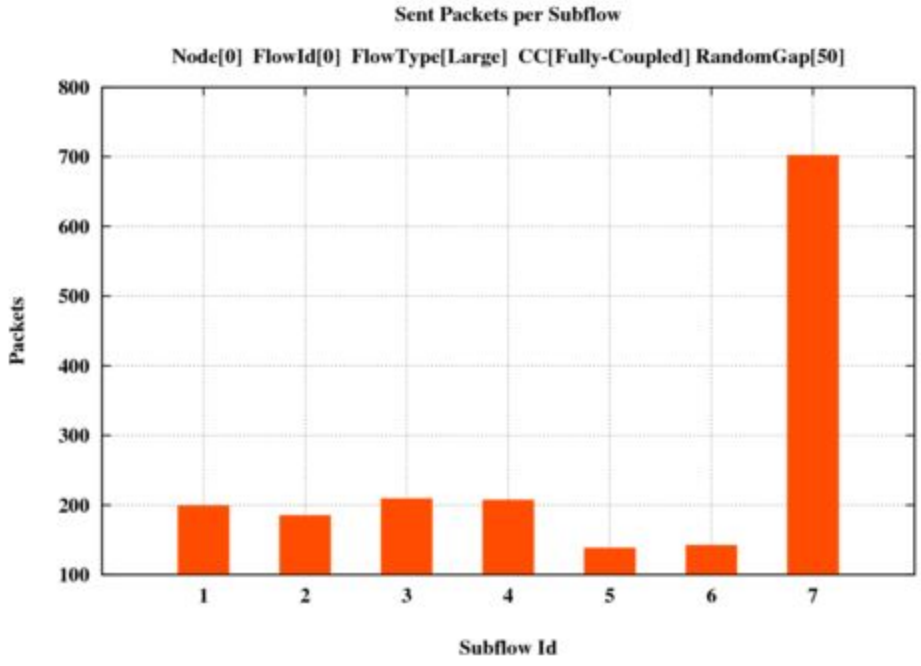


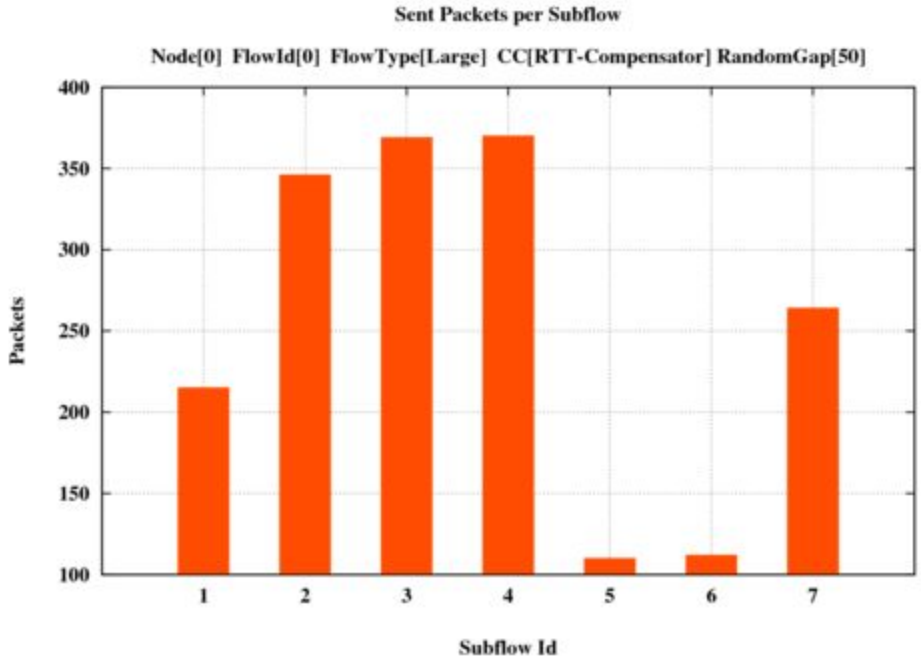


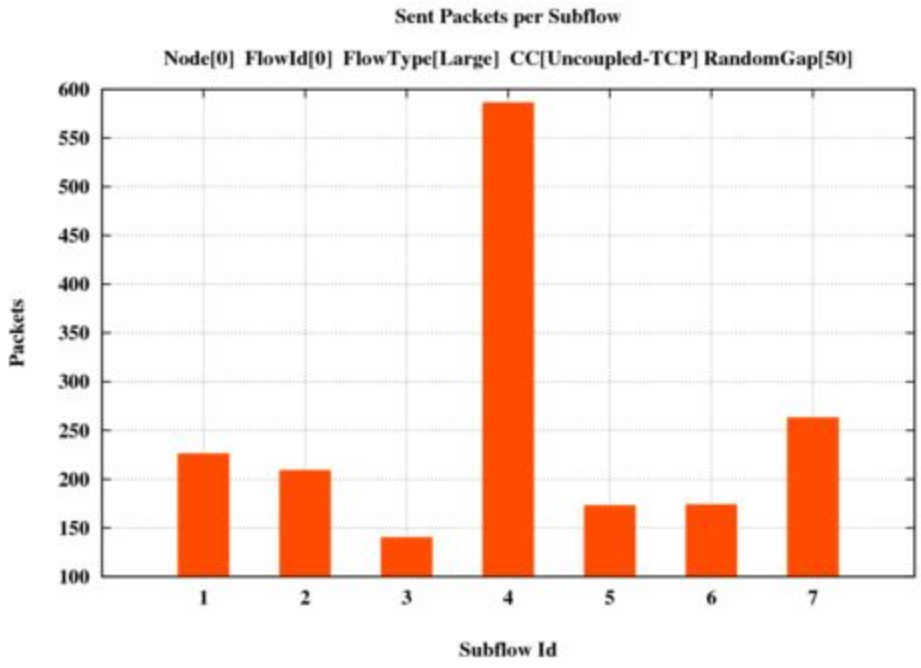
The above four graphs depict the congestion window vs time for 3 existing MPTCP congestion control algorithms as well as our own custom one. In the first graph, our algorithm, you can see that subflow 1's congestion window grows after the 6 second mark while the other subflows'

congestion windows decrease and stabilize. This means subflow 1's path is the least congested which is why it is growing as more ACKS are being received, indicating less congestion. This follows the MPTCP congestion control philosophy that traffic should be moved from more congested paths to less congested ones.




Sent Packets per Subflow
Node[0] FlowId[0] FlowType[Large] CC[Fully-Coupled] RandomGap[50]
Packets
800
700
600
500
400
300
200
100
1
2
3
4
5
6
7
Subflow Id


Sent Packets per Subflow
Node[0] FlowId[0] FlowType[Large] CC[RTT-Compensator] RandomGap[50]
Packets
400
350
300
250
200
150
100
1
2
3
4
5
6
7
Subflow Id

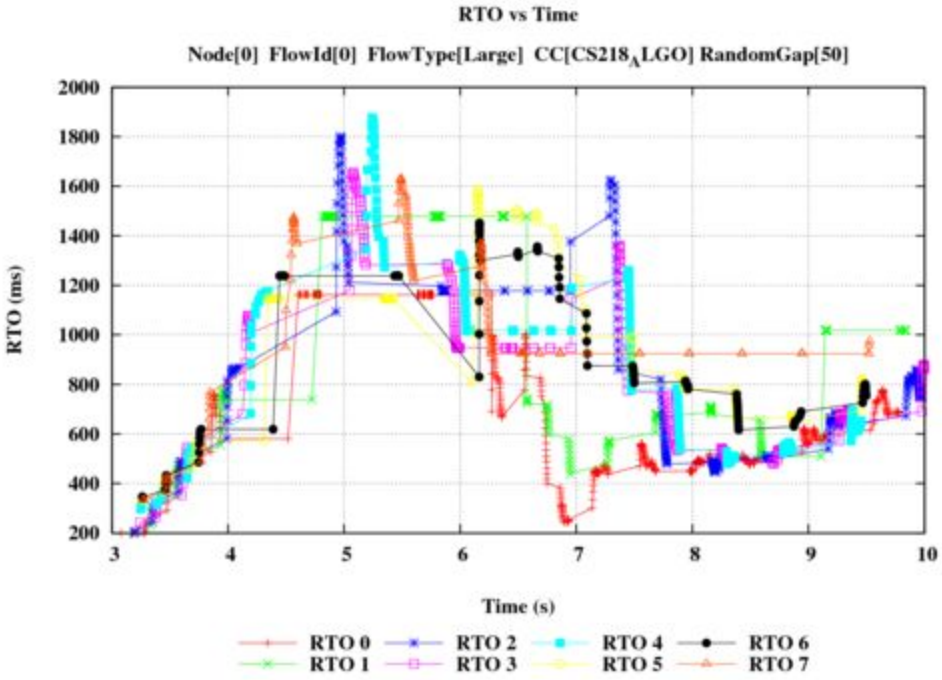


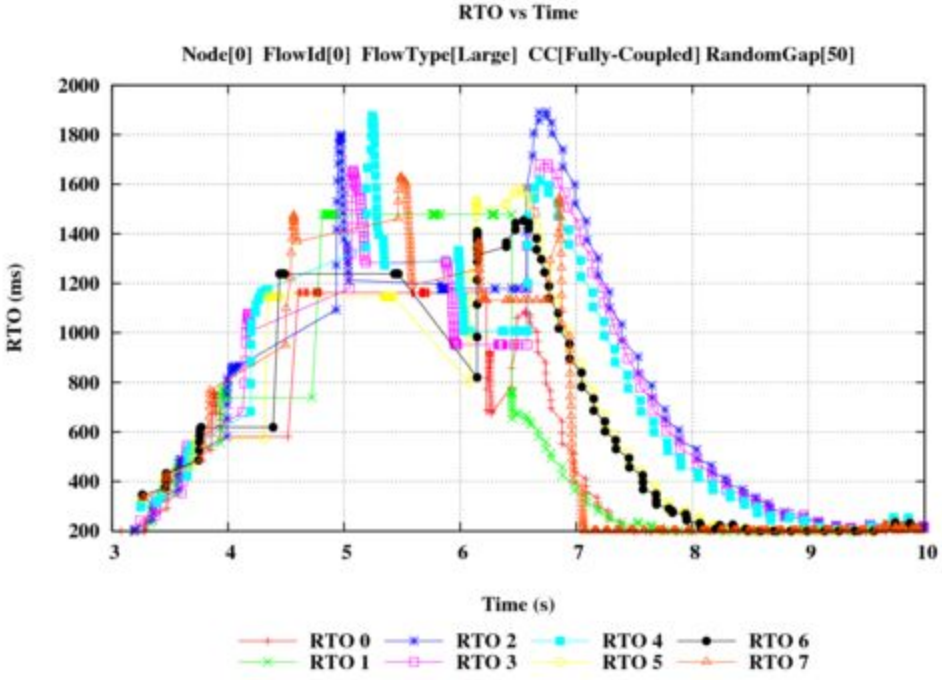
The above four graphs show the number of packets sent per subflow for our algorithm and the three existing ones. To aggregate the stats, this below graph shows the approximate throughput over 10 seconds for the four different algorithms for a summary of what the histograms are saying. For clarification the bandwidth for each channel is 5 Mbps.

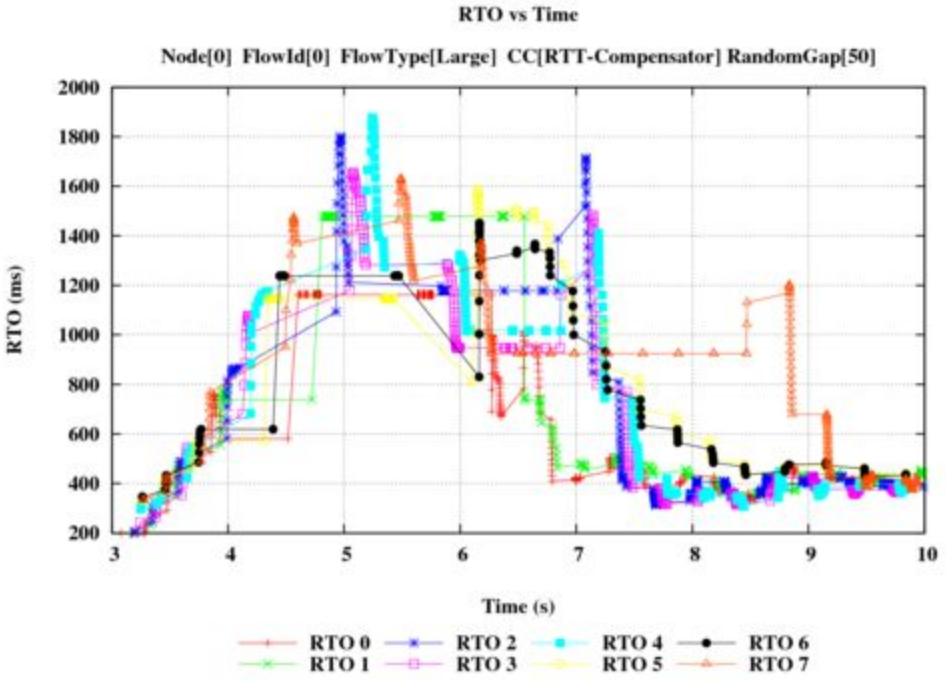
Average Throughput over 10 runs

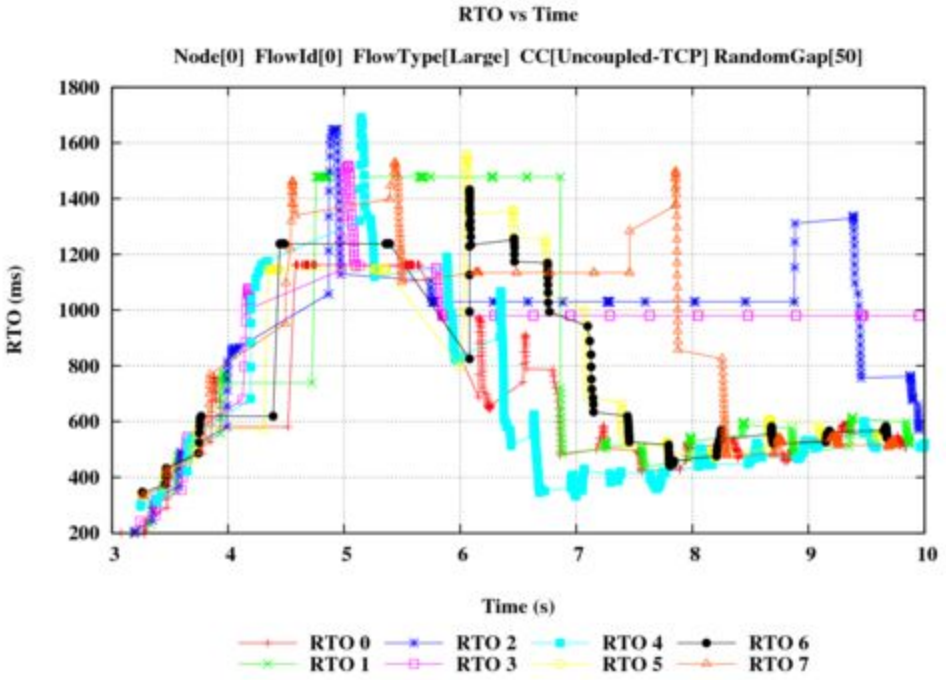
Algorithm	**Throughput**
CS218 ALGO	25.04 Mbps
Fully Coupled	24.06 Mbps
Uncoupled	24.89 Mbps
RTT Compensator	24.52 Mbps

As you can see, our algorithm improves upon throughput slightly.


RTO vs Time
Node[0] FlowId[0] FlowType[Large] CC[CS218_ALGO] RandomGap[50]
RTO (ms)
Time (s)
RTO 0
RTO 1
RTO 2
RTO 3
RTO 4
RTO 5
RTO 6
RTO 7

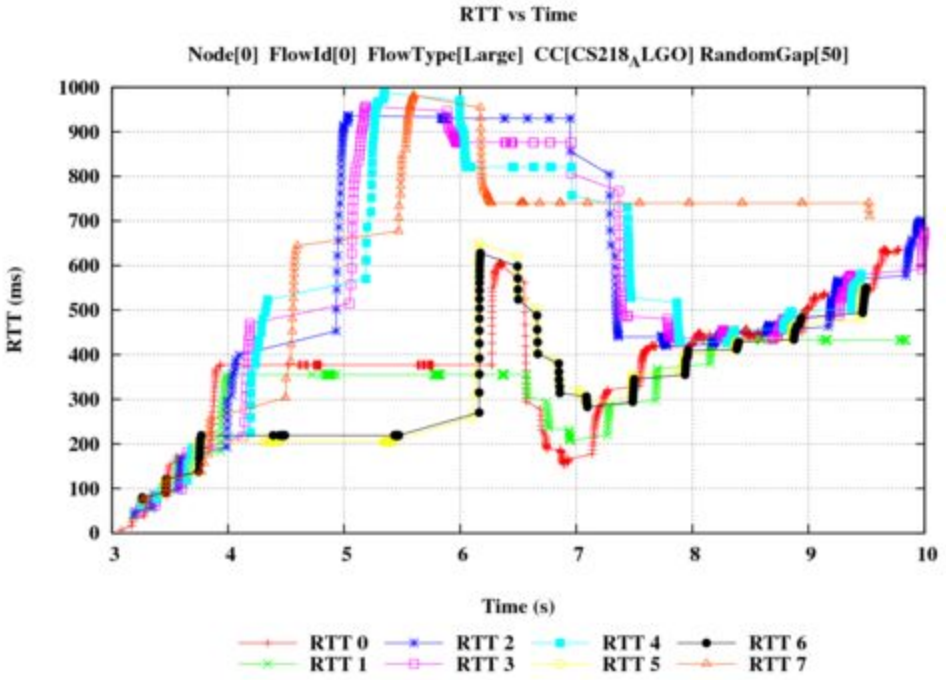

RTO vs Time
Node[0] FlowId[0] FlowType[Large] CC[Fully-Coupled] RandomGap[50]
RTO (ms)
Time (s)
RTO 0
RTO 1
RTO 2
RTO 3
RTO 4
RTO 5
RTO 6
RTO 7

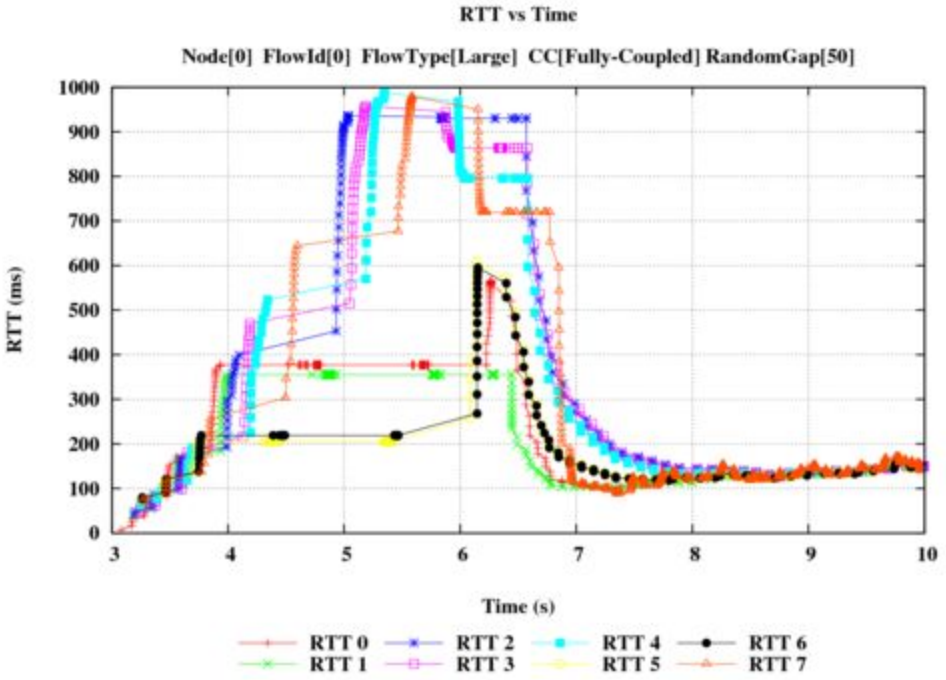


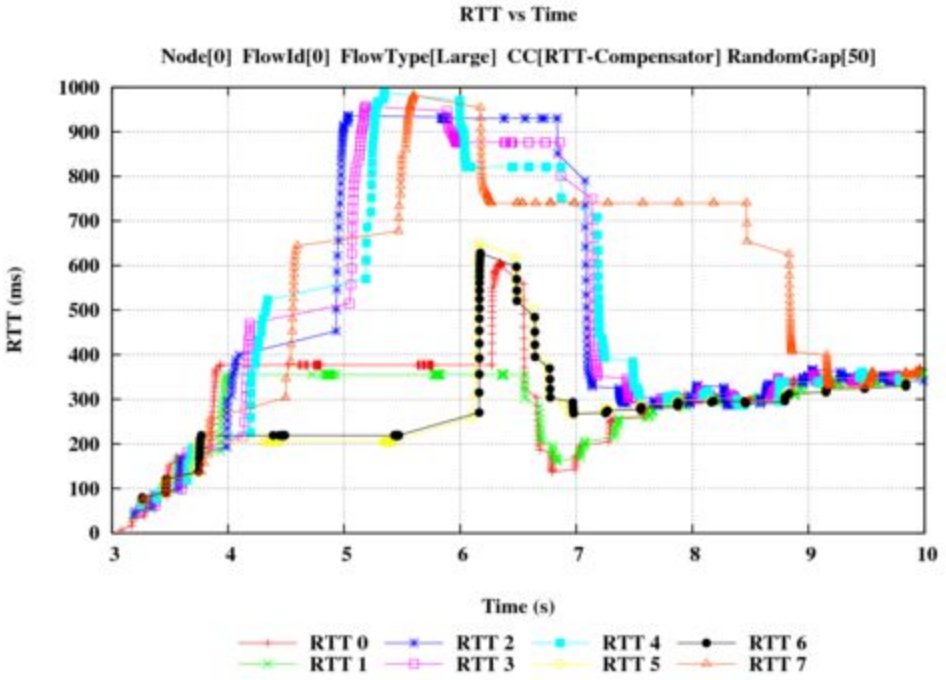


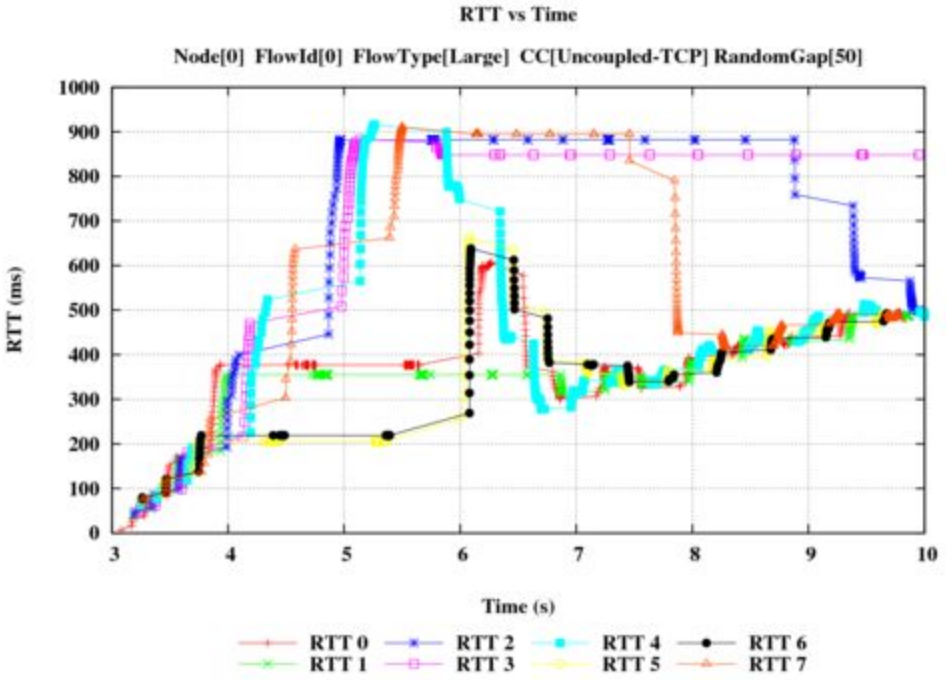
The four graphs above show the RTO for the different algorithms. Our algorithm performed slightly worse in this regard, but that is a consequence of higher throughput. Because we are

sending more packets, logically there should be more packets lost just due to increased traffic which will result in the RTO increasing. Our algorithm does not worsen congestion but because of more packets being sent, there are more lost packets and timeouts that happen.



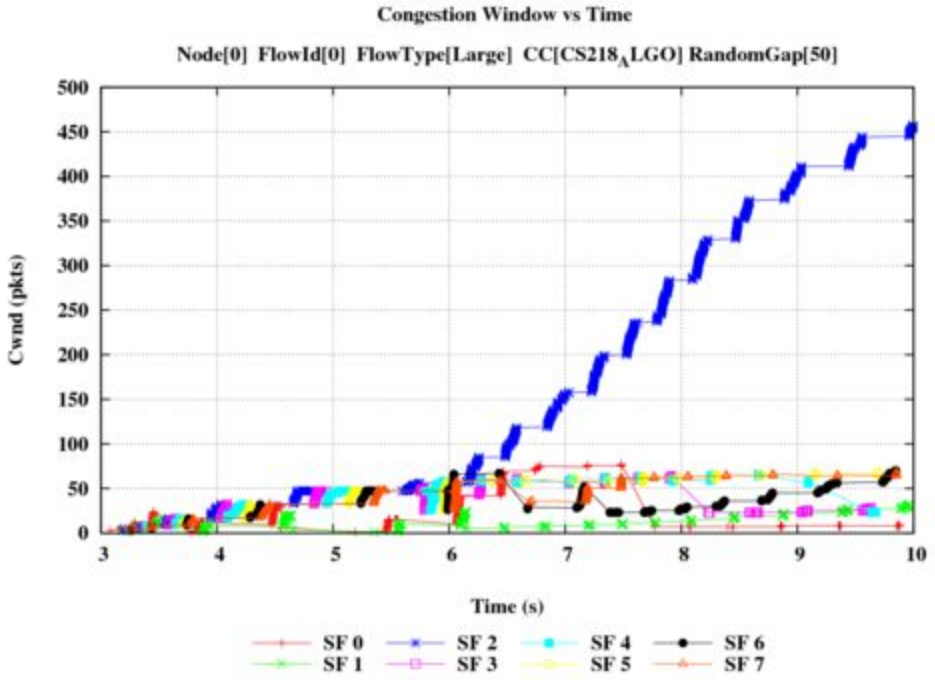


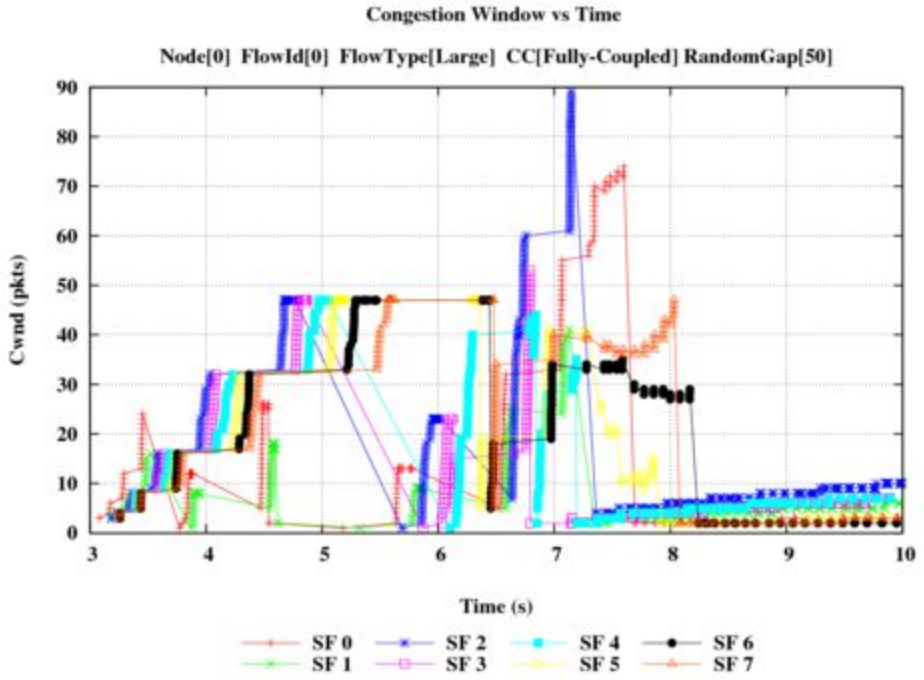


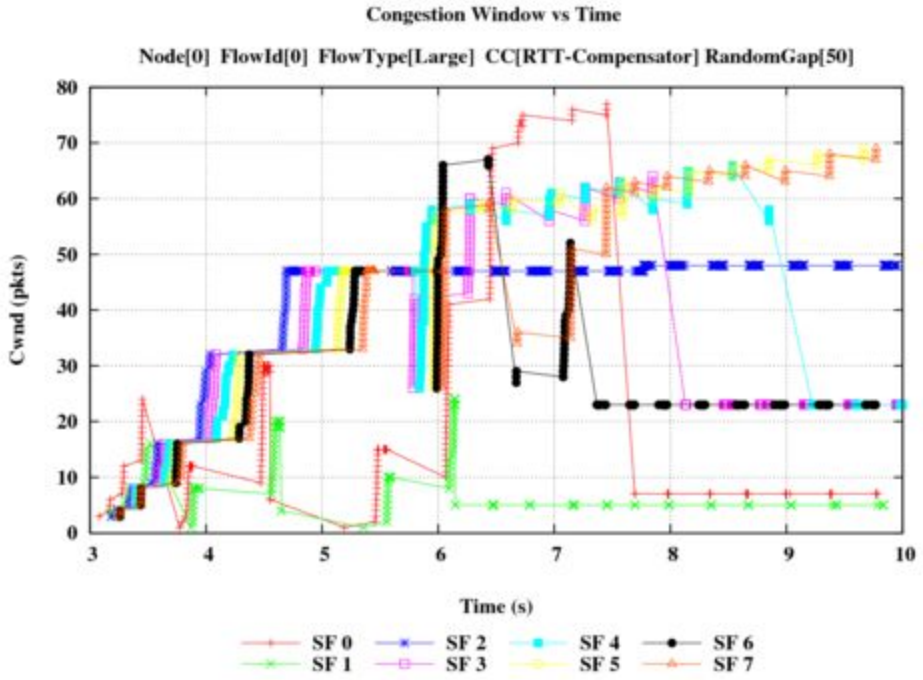


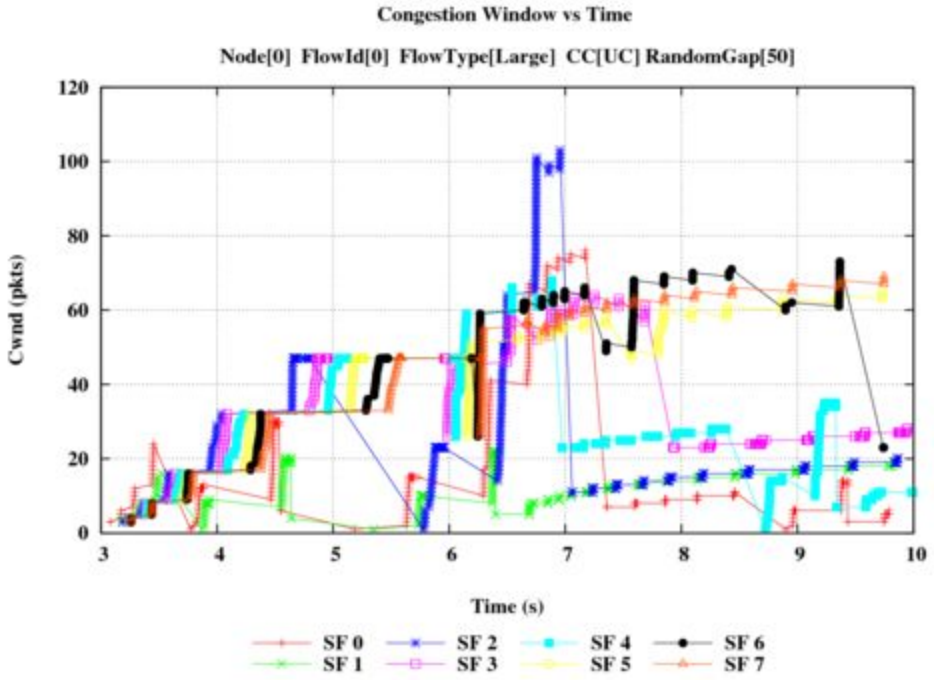
The four graphs above show RTT for the different algorithms. For RTT, our results were not as good as the other algorithms as it increased slightly. We believe this is because more packets are sent due to higher throughput, meaning there will be more delay between the send and receive.

The following sets of graphs are for a MANET with no mobility meaning all the nodes are static. We wanted to test out to see how the existing algorithms and our algorithm performed in a much more stable environment. An application for this could be a sensor network.



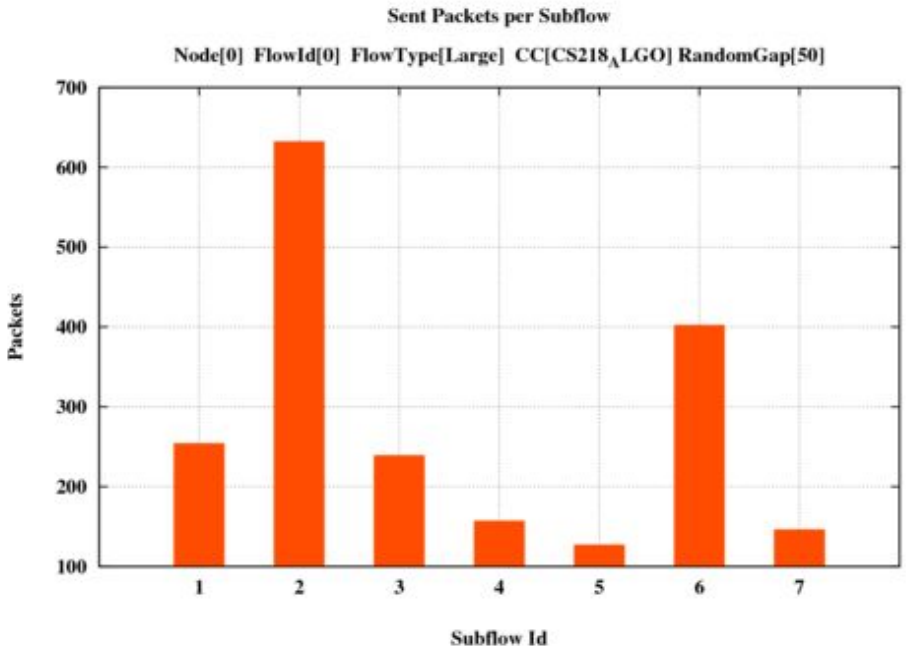


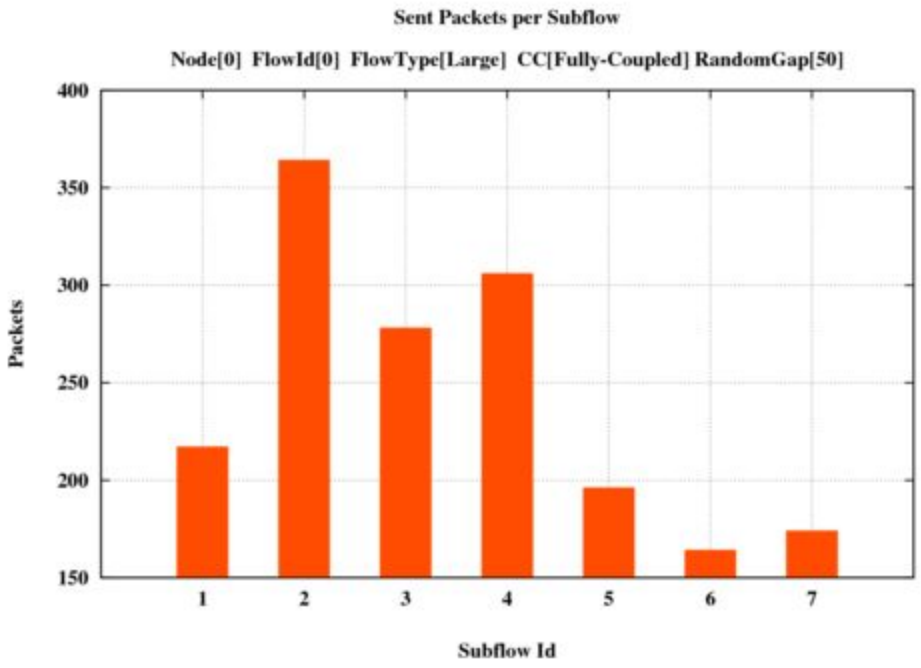


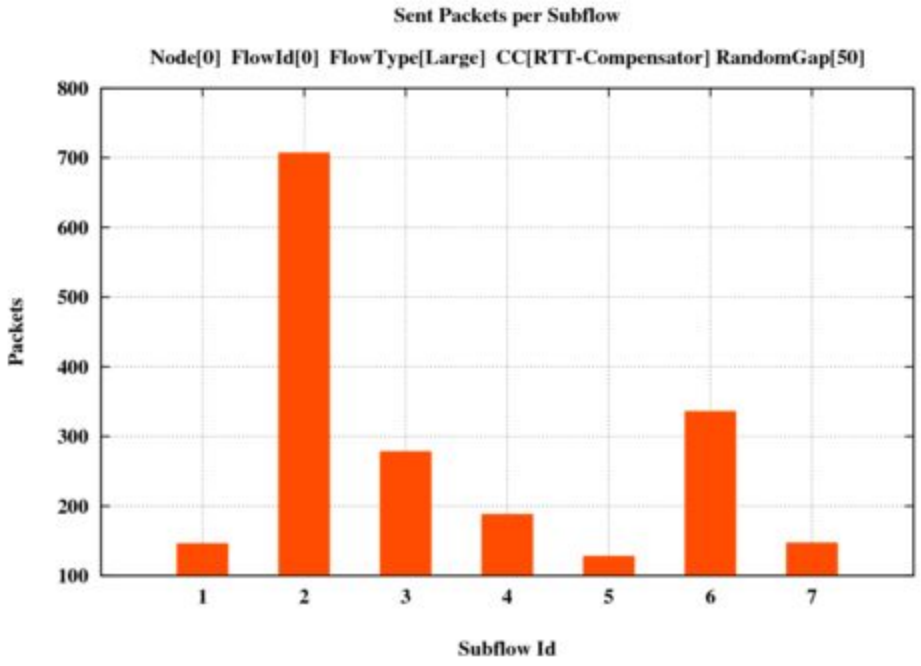


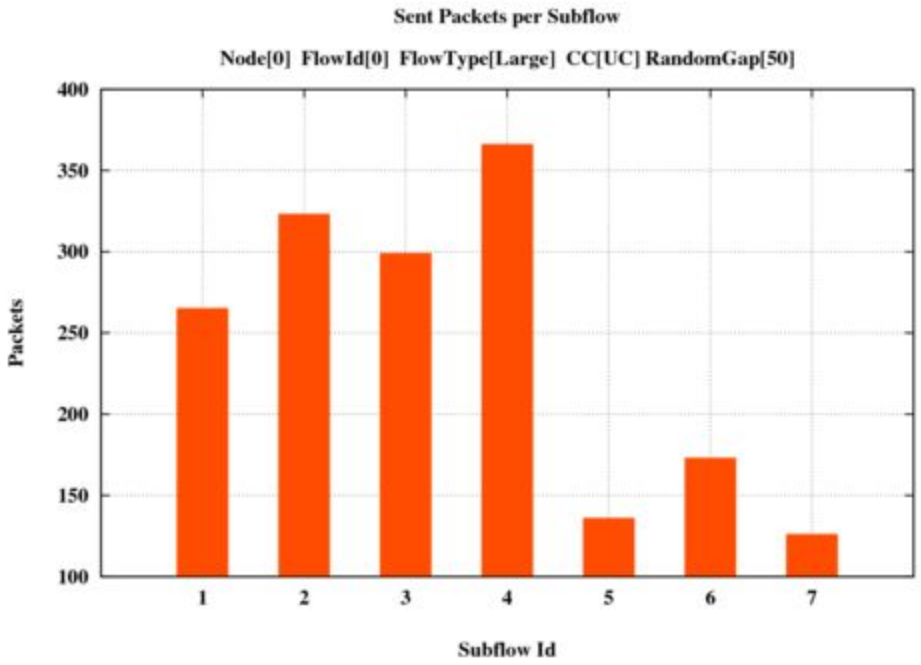
As you can see, for our algorithm the congestion window for subflow 2 is the greatest and grows faster than any other subflow in any other algorithm. Due to the stable nature of the topology, this path is the one with least congestion and it does not change which is why more packets are

being sent on this path. One might expect more packets being sent to induce congestion but a likely explanation is the lack of cross traffic. The other algorithms attempt to balance congestion among the subflows but fail to optimally utilize the resources to gain higher throughput as will be shown in the following graphs.



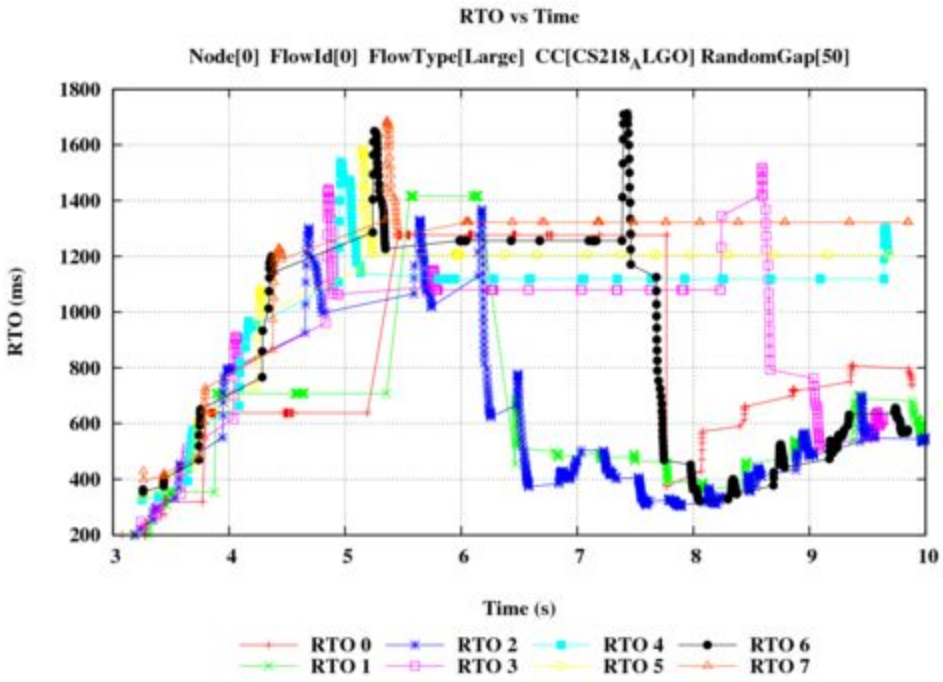

Sent Packets per Subflow
Node[0] FlowId[0] FlowType[Large] CC[Fully-Coupled] RandomGap[50]
Packets
400
350
300
250
200
150
1
2
3
4
5
6
7
Subflow Id

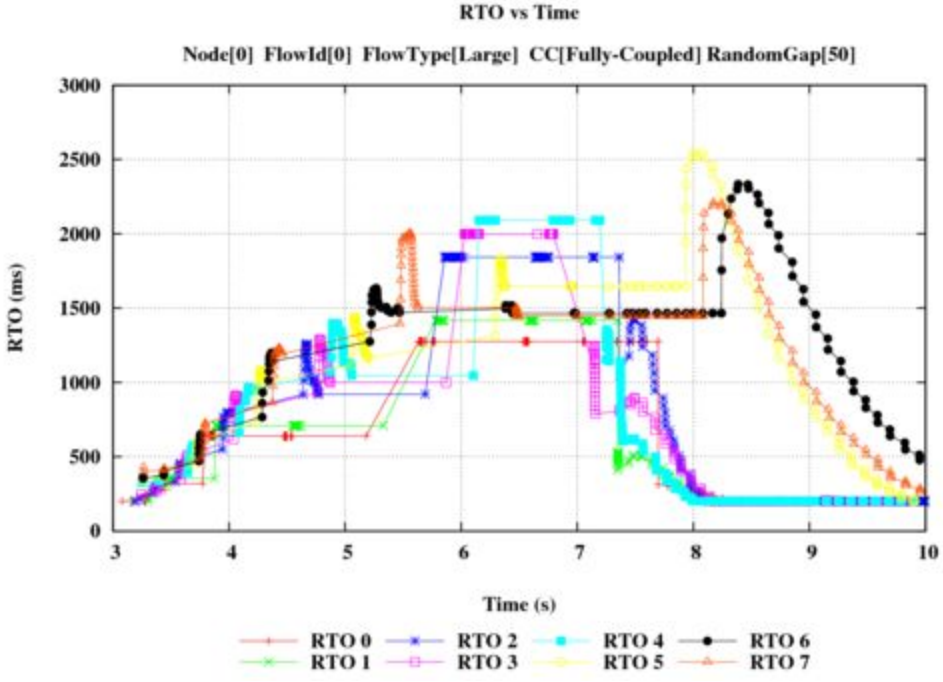

Sent Packets per Subflow
Node[0] FlowId[0] FlowType[Large] CC[RTT-Compensator] RandomGap[50]
Packets
800
700
600
500
400
300
200
100
1
2
3
4
5
6
7
Subflow Id

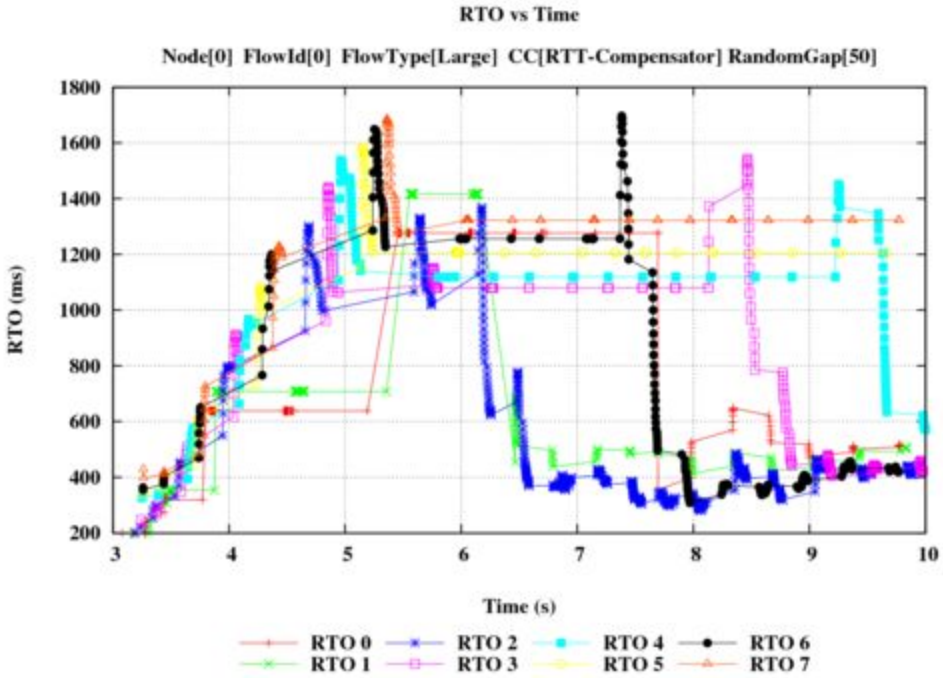


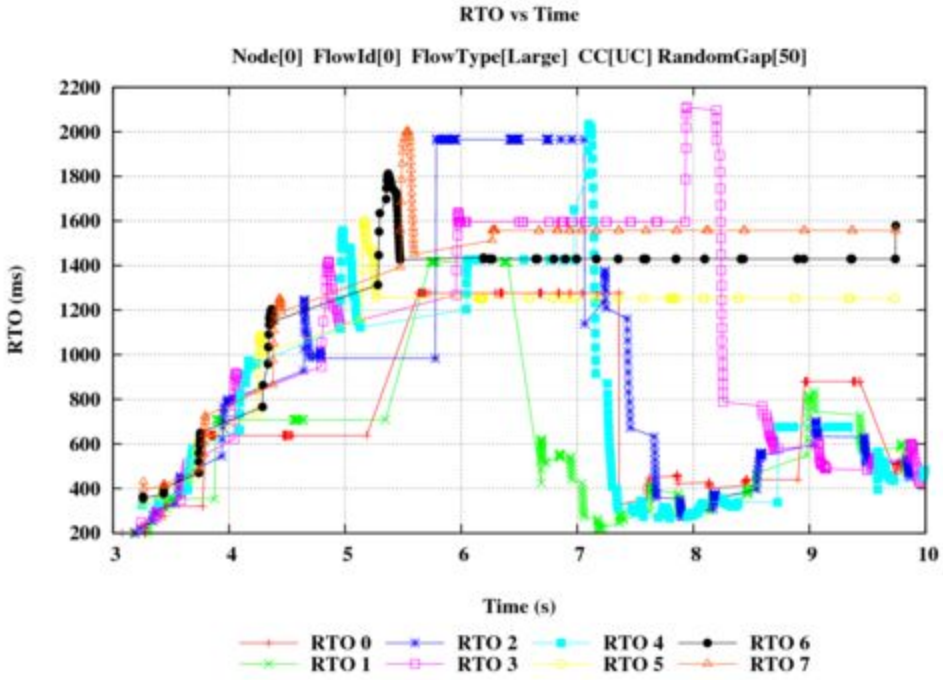
As you can see below, the throughput for our algorithm was highest averaged over 10 runs.

Algorithm	**Throughput**
CS218_ALGO	25.05 Mbps
Fully Coupled	22.3104 Mbps
RTT	24.74 Mbps
Uncoupled	22.3328 Mbps

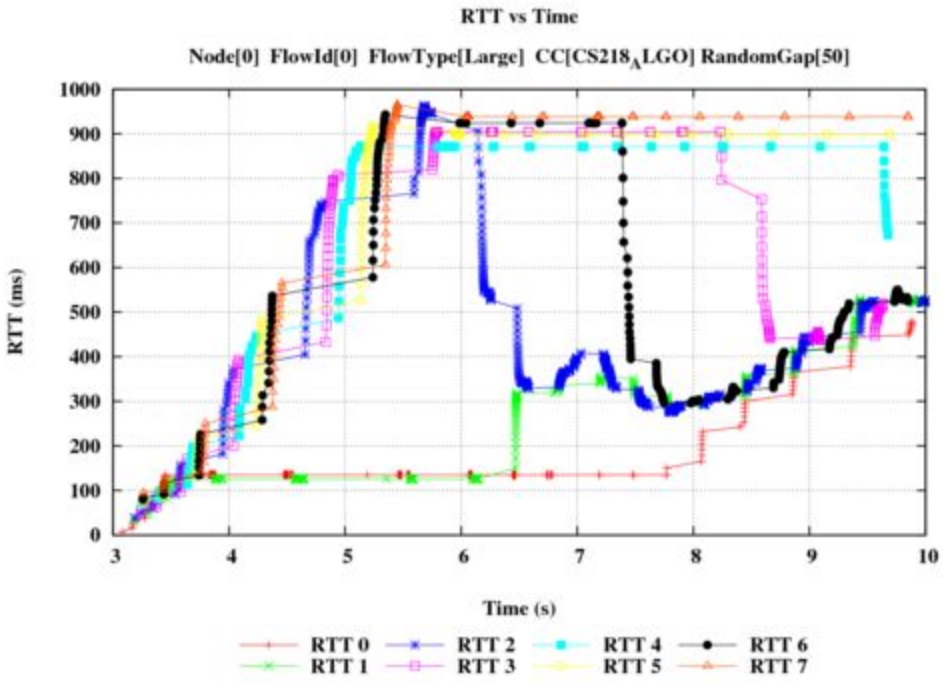

RTO vs Time
Node[0] FlowId[0] FlowType[Large] CC[CS218_ALGO] RandomGap[50]
RTO (ms)
Time (s)
RTO 0
RTO 1
RTO 2
RTO 3
RTO 4
RTO 5
RTO 6
RTO 7

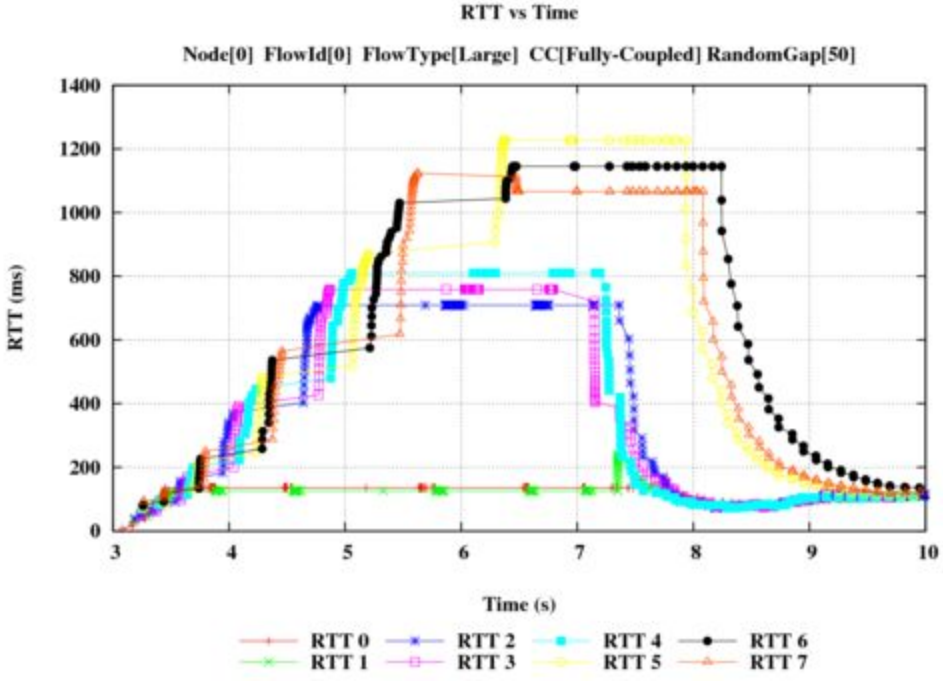

RTO vs Time
Node[0] FlowId[0] FlowType[Large] CC[Fully-Coupled] RandomGap[50]
RTO (ms)
Time (s)
RTO 0
RTO 1
RTO 2
RTO 3
RTO 4
RTO 5
RTO 6
RTO 7

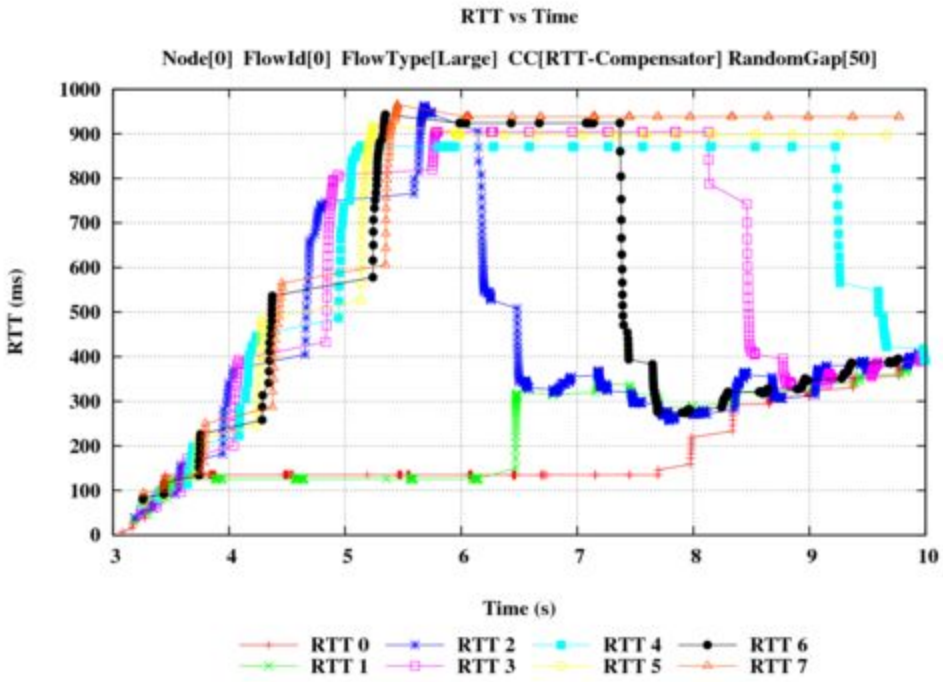


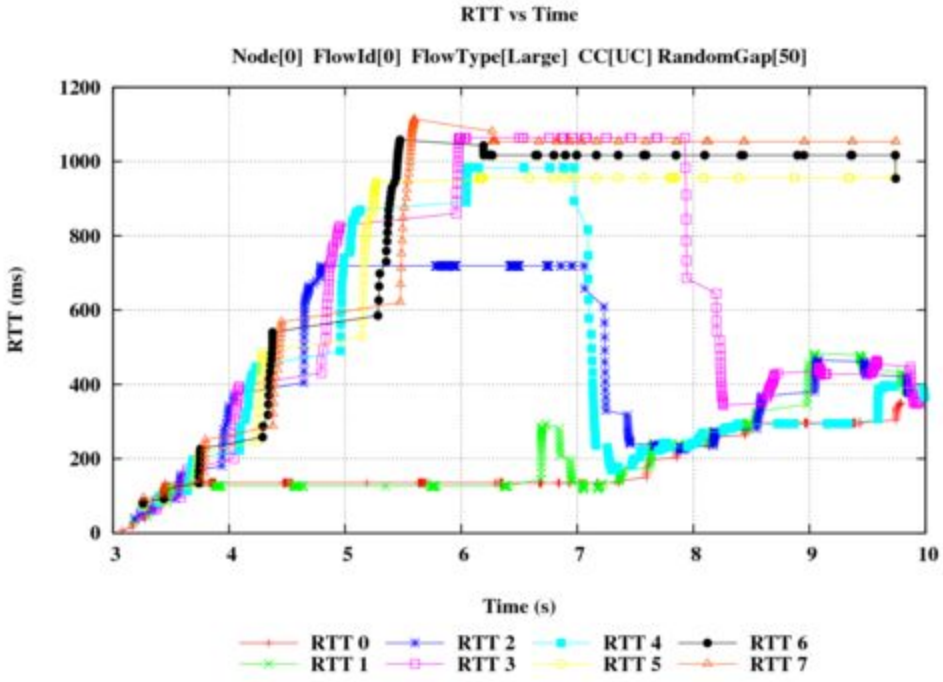


For RTO, our algorithm performed similarly to RTT Compensator and Uncoupled but not as well as Fully Coupled. An explanation could be that Fully Coupled sends the least amount of packets meaning there are less likely to be timeout events so the RTO does not increase as much.


RTT vs Time
Node[0] FlowId[0] FlowType[Large] CC[CS218_ALGO] RandomGap[50]
RTT (ms)
Time (s)
RTT 0
RTT 1
RTT 2
RTT 3
RTT 4
RTT 5
RTT 6
RTT 7


RTT vs Time
Node[0] FlowId[0] FlowType[Large] CC[Fully-Coupled] RandomGap[50]
RTT (ms)
Time (s)
RTT 0
RTT 1
RTT 2
RTT 3
RTT 4
RTT 5
RTT 6
RTT 7





Once again, our algorithm did not perform the best in RTT measurements but this could be explained by the fact that there are more packets being sent so more traffic hence and more delay between sending and receiving.

5. Future Work

There are a few ways our project can be expanded upon for further research. Firstly, the topologies we used do not accurately reflect real-world MPTCP usage scenarios. The most common use case envisioned for MPTCP is for mobile devices where both cellular and Wi-Fi interfaces can be leveraged to improve throughput by either sending out of both simultaneously or sending out of the best one if power consumption is considered. Thus, as our topologies fail to include an LTE network and more realistic dual-homed devices due to problems and bugs we encountered, a future work could be to create a more practical topology consisting of LTE and Wi-Fi networks. Also, at a glance, none of our topologies include a true multipath nature. This was mainly due to bugs we encountered due to not being able to include multiple interfaces of different types on devices. When we tried to include dual homed devices, only one of the interfaces was being utilized. Our model includes only logical 802.11 interfaces.Thus, our project was limited by the staleness of the model we were using that precluded including more realistic scenarios. In the future, our own MPTCP implementation could be created to overcome the problems and shortcomings we had using this existing one. In addition, the topologies we created consisted of one or two subnets. A future work would be to create larger scale topologies consisting of multiple subnets of different network technologies to better model the Internet. Also, our main usage scenario for our experiments was a MANET but the growth of connected cars means evaluating MPTCP in a VANET could be an interesting research direction. Lastly, there are hundreds if not thousands of TCP congestion controls out there. Not all of them have been tested with MPTCP. Despite mentioning that traditional TCP congestion control

mechanisms will not usually work, tweaking existing ones to account for the multipath nature could be fruitful. MPTCP has a lot of potential in the future as evidenced by Apple adopting it as its mechanism for Siri, and continued analysis and evaluation can only serve to justify its promise in an evolving Internet.

6. Summary

In this project we have studied the congestion algorithms used in Multipath-TCP and explored other congestion algorithms used in different variants of TCP. We implemented our own congestion algorithm to increase the throughput of the connection by incrementing the congestion window faster, in order to obtain a higher utilization of the bandwidth in cases where packet loss is not caused by congestion. We have implemented such an algorithm and tested its performance on several topologies as mentioned above using ns-3. The results have shown that our algorithm reaches a slightly higher and fairer throughput (packets sent per subflow) while having a slight negative impact on the RTO and RTT. In addition, we have analyzed the constraints and performance differences under different scenarios, and we have proposed a few aspects of this project that remains open to improvement.

BIBLIOGRAPHY

1. Craig, K. Compound TCP in ns3. https://web.cs.wpi.edu/~rek/Adv_Nets/Fall2014/TCP_Compound_Project.pdf
2. Coudron, M., Secci, S. (2017). An implementation of multipath TCP in ns3. Computer Networks, Volume 116.
3. Hadji, A. (2017). From TCP to MPTCP Brief Explanation. Nov. 5, 2017. https://www.slideshare.net/akremgegawatt/from-tcp-to-mptcp-a-brief-explanation-81616545?next_slideshow=1.
4. Handley, M., Wischik, D. Raiciu, C. Coupled Congestion Control for MPTCP. https://www.ietf.org/proceedings/77/slides/mptcp-9.pdf
5. Kheirkhah, M. (2014). MPTCP. https://github.com/mkheirkhah/mptcp
6. Kheirkhah, M., Wakeman, I., and Parisis, G Multipath-TCP in ns-3. https://arxiv.org/pdf/1510.07721.pdf
7. Lochert, C., Scheuermann, B., Mauve, M. (2007). A Survey on Congestion Control for Mobile Ad-Hoc Networks. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.7225&rep=rep1&type=pdf
8. Murari Sridharan, Kun Tan, Jingmin Song and Qian Zhang - "A Compound TCP Approach for High-speed and Long Distance Networks" https://www.microsoft.com/en-us/research/publication/a-compound-tcp-approach-for-high-speed-and-long-distance-networks/
9. Ns-3. https://www.nsnam.org/
10. Paasch, C., Bonaventure, O. (2014). Multipath TCP. acmqueue, Volume 12.
11. Raiciu, C., Handly, M., Wischik, D. - "Coupled Congestion Control for Multipath Transport Protocols" [RFC 6356] https://tools.ietf.org/html/rfc6356
12. Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., Handley, M. (2012). How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. NSDI 2012.
13. Raiciu, C., Wischik, D., Handley, M. Linked Congestion Control. https://www.ietf.org/proceedings/76/slides/mptcp-8.pdf
14. Wischik, D., Raiciu, C., Greenhalgh, A., Handley, M. (2011). Design, implementation and evaluation of congestion control for multipath TCP. NSDI.