

Division of Engineering and Applied Science

Department of Electrical Engineering

CS/EE 143: Communication Networks

**Network Simulation Project**

Anyesha Ghosh, Cat Le, David Mace

December 5, 2016

**1 Introduction**

In the past few decades alone, society has iterated through several generations of network architectures, including congestion control, routing update algorithm. In congestion control, there are two major approaches: UDP and TCP; the most stable of which, TCP Reno, has now been adopted by nearly all major network operators across the world. As of recent years, there has been an enormous growth in the amount of users on the network, creating an incentive to experiment with new methods of network expansion. Therefore, the main goal of the project is to simulate the modern network, analysis the performance, to understand the meaning and calculation behind what people are using in network today. The students are also required to perform the theoretical calculation in each test cases and compare with the simulation results. The simulation is required to implemented in two different congestion controls. Only one routing update algorithm is required, though it is recommended to perform in both Bellman Ford algorithm and Dijkstra Algorithm. In the simulation, the properties of each nodes (host, link, router) can be modified, such as host can have the congestion control algorithm instead of links. The main goal of the project is to simulate the network so that students can obtain the real-time result of the communication of the actual network.

**2 Methodology**

The network simulation in this project is implemented in Python, that based on process-based event simulation to imitate real-time and accurately how the network’s nodes, such as hosts, links, routers, communicate with each others. Based on multithread programming, this network simulation is garanteed to work in real-time and process in parallel. The output of the simulation is the logger and plots that refect the time of took to deliver a packet of information across the whole network. It also shows the effectiveness of the links through link rate, or the loss rate when packet is dropped due to full buffer’s occupancy. This simulation are also a test cases for different method of congestion control (TCP Reno versus FAST TCP), routing update algorithm (Bellman Ford versus Dijkstra), and full duplex versus half-duplex.

**3 Design and Setup**

**3.1 Packet Class**

The packet class represent information or data which is sent between nodes (hosts, routers,…) in the network. Each packet has a header and a payload, which carries actual data. The header is the information to direct the packet when it was in the network.

The header includes:

* SOURCE (src): the sender of the packet, usually not change over the network, except in some cases where there is packet loss.
* DESTINATION (dst): the destination which sender want to send to packet to.
* TYPE: packet possibly be in 3 types: DATA, ACK, Routing. DATA is the actual information; ACK is the acknolegement sent when the host receive something; there is ACK of the ACK or three-way handshake in TCP to generete the connection between 2 hosts; Routing packet is specific for routing table update; it is used when router want to update its neighbor in the network about a new path to send the packet.
* SIZE (DATA, Routing: 1KB or ACK: 64B): determine how long it take to send it through a link, or how much buffer space it need when packet is in the queue.
* PACKET NUMBER (pktnum): the parameter to keep track of each packet in order to response when there is packet loss (retransmit the lost packet only) or to keep the packet in correct order.
* OPTIONAL FLAGS: used to detect error within the packet data. In this simulation, we do not focus on the correctness of data, but only the sucess of the transmission. In actual   
    
  network, the data can be transmitted successfully, but the data inside might be corrupted, which make the receiver asked for retransmit, even though the packet is received correctly.

As mentioned above, there are three types of packet:

* DATA (or standard packet): used to imitate the communication between host and host on a network. In this simulation, each DATA packet has the standard size, which is 1KB or 1024 Bytes. This includes both the header and the payload (actual data).
* ACK (acknowlegement): used to notify the sender about the successful receive of specific packet. It is specified by the packet number. In the beginning of the communication, there is three-way hanshake between two host: one host will send a opening packet with no actual data inside; another host will send an ACK when it receive it, otherwise, the sender will resend the opening packet after a certain timeout. Finally, the sender will send an ACK to acknowledge the ACK of the reciver. When the communication is complete, the sender will send the closing packet, the receiver host will send an ACK+closing packet, and when the sender host receive it, it send another ACK. This is when the network between these two hosts actually closed.
* Routing Update Packet: used to communicate dynamic routing information between routers to routers across the network. This packet has a size of 1KB or 1024 Bytes. The information in this packet is generated by the routing update algorithm, which functions to find a best path for a specific packet to travel across the network. This type of packets are generated and propagated to all nodes on the network.

**3.2 Host Class**

Host Class represents the end-users in the network. It generates the packet and send to other host that request for the information, or it will receive the packet and read the information inside. When receiving a DATA packet, the host will have to send an ACK to the source of the packet to acknowledge the receiving packet. If the receiving packet is an ACK, the host will notified its timeout section about the arrival of DATA packet and not to retransmit another DATA packet. In acutal network, the host also responsible for propagate the routing update packet.

In this simulation, the host is the most complicated and high-level implementation. While having the link setup, packet generator, ACK mechanism, and timeout mechanism, it also includes the congestion control, which is actually based on the link or router in real network. This setting helps to minimize the complexity of the Link and the Router, while guarantee the performance of the network simulation compared to the actual network.

* Link Setup:
* Packet Generator:
* …

In this project, the Congestion Control methods which were implemented are TCP Reno and FAST TCP. The main purpose of the congestion control is to minimize the packet drops during the network transmission. It will modify the window size for specific time in order to maximize the network performance. Whenever there is a packet drop, the congestion control will reduce the window size and set a new limit of it.

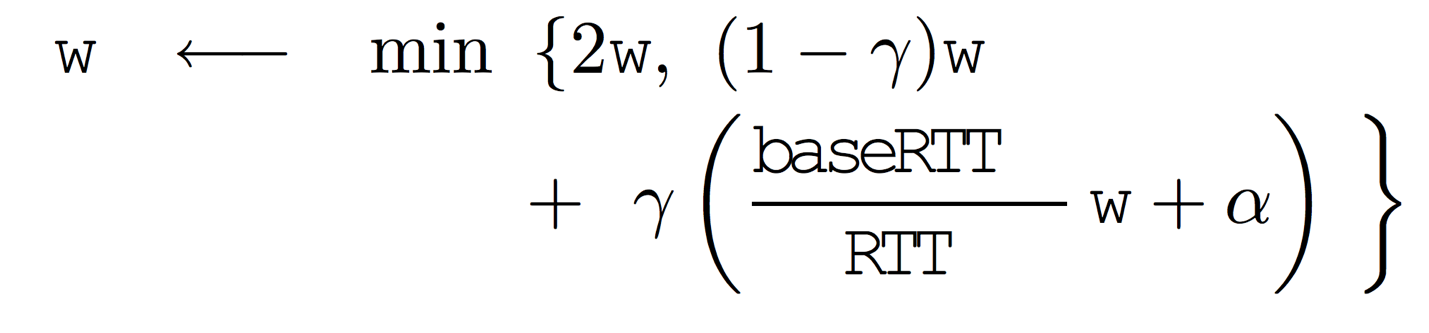
The congestion control also remember the packet number for retransmitting purpose. The packet drop is detected by the certain timeout, with the base timeout set at the beginning and modified overtime. After timeout, if the sender still have not yet received the ACK, the congestion control will notify it as a packet drop and retransmit the packet.

The packet drop is also triggered by a three-duplicate ACK. When the receiver receive the packet whose packet number greater than the missing packet, it will send the ACK of the previous, in-order packet, notifying the out of order packets have been received. When 3 consecutive ACKs like this happen, there is no need to wait for timeout to trigger the packet drop.

In TCP Reno there are 3 states: Slow Start, Congestion Avoidance, or Fast Recovery. At each state, the window size is changing in different speed. While in FAST TCP, the window is updated by calculating the RTT of the packet and saving the minimum RTT encountered.

While TCP Reno is the most well-known and most popullar congestion control, FAST TCP, which was developed by Microsoft has proved itself with the speed of processing and the stability in network.

The formula of window size in FAST TCP:



with base parameters and , the convergence in window size will happen. If this parameter is not fit with the network, change them by 1 unit and test the network again. In our simulation, the value of parameters are and .

**3.3 Buffer Class**

Buffer Class representes the number of packets for which a link queue can hold while transmitting others packet. In this simulation, all links’ buffer are set to be at 64KB, which can hold 64 DATA or Routing Update packet. While the link in this simulation is half-duplex, which means only one direction of flow is available at one time, the buffer of this link is shared for both directions. In other word, the buffer of a specific link is a same buffer for back and forth directions.

The buffer works based on two main functions:

* put(): add the packet into the queue, in first-in-first-out (FIFO) order if there is sufficient space available in the buffer; if there is not enough space, drop the packet (cause packet loss)
* get(): pull/retrieve the packet from the buffer in correct order and get ready to send it over the link.

At the beginning, the buffer is assumed to be empty, which minimizes the packet loss – causes by buffer overflow.

**3.4 Link Class**

Link Class represents the medium to transmit packets between exact two nodes of the network, for example host-to-host, or host-to-router, or router-to-router. In this simulation, the link rate, link delay, and buffer size are specified by the users. The link rate determines how fast it take to process and send 1 bit of data. The link delay is the time it takes for the packet to travel across the link. In order to simualate this delay, a packet is set to stay at the host for a certain amount of time appropriate to the link characteristics before process, and send it through link. In this simulation, the link class also determine the size of the buffer, which unlike the actual network, where the buffer is located at the hosts’ end or the routers’ end. At the buffer, the packet is waiting in FIFO order to be processed and sent while the link is busy sending other packets. The buffer has specific capacity and the packet loss happens whenever the buffer is out of space. When a certain packet is dropped, the buffer will notify the congestion control to reduce the window size in order to prevent further drops. The link operates as half-duplex link, and it will put the packets in its buffers whenver it is currently sending a packet in either directions.

The link operation can be divided into three functions:

* onreceive(): place a received packet in the buffer whenever the link is busy sending other packets.
* propPkt(): simulate the propagation delay of the packet by set a timer for the packet to stay at the sender for a certain amount of time.
* sendPkt(): pull/retrieve the packet from the buffer in FIFO order and send it over the link

**3.5 Router Class**

Router Class is similar to Host Class, except they do not have responsibility for the congestion control. Instead of that, Host Class determine the routing path of the packet and routing table is located in all router across the network. Whenever the router receive a packet, it will only read the header, but not the payload of the packet, except for the routing update packet. The main function of the router is to forward the packet to the next hops on its best path to the destination. It does not need to look inside the payload DATA or ACK packets. While looking at the header of packet, the router will identify the source and the destination of the packet, and then looking up the current routing table and forward the packet to its next hop/destination. If the router receives a routing update packet, it will read and update its own routing table, then parallelly send the routing update packet to its immediate neighbor routers. The routing update packet is propagated through the network until all router received and updated their own routing table.

In this simulation, the routing update algorithm is Bellman Ford algorithm. The routing table is updated whenever these two situations happen:

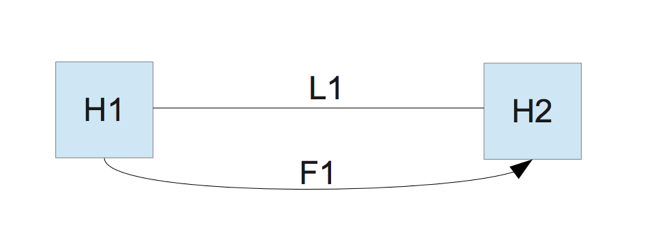
* A new shorter path appears when the propagated routing update is received by certain router.
* The origiral path went through the sender becomes longer than calculated before.

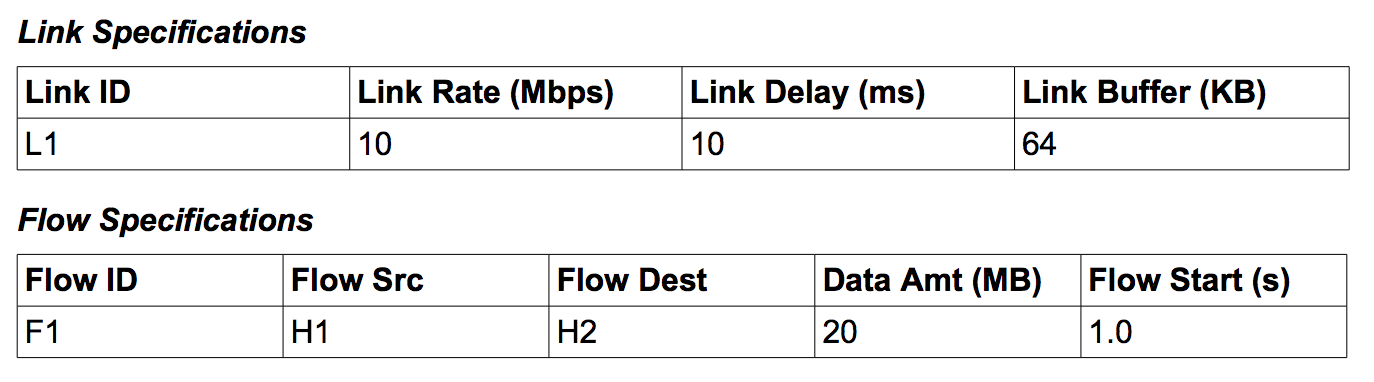
In general, the router can be represented as following functions:

* routing\_update(): update routing table for immediate neighbor links whenever receiving a routing update packet; then send out this packet to the network.
* update\_table(): change the routing table to reflect the information in the router; then send the packet across the link.
* send\_table(): propagate the routing update packet to its neighbor routers.
* route(): send the packet to the next link on its best path to destination
* receive(): receive a packet from a host or other routers; then process to read the header(for DATA and ACK packets) or the entire packet (for Routing update packets)

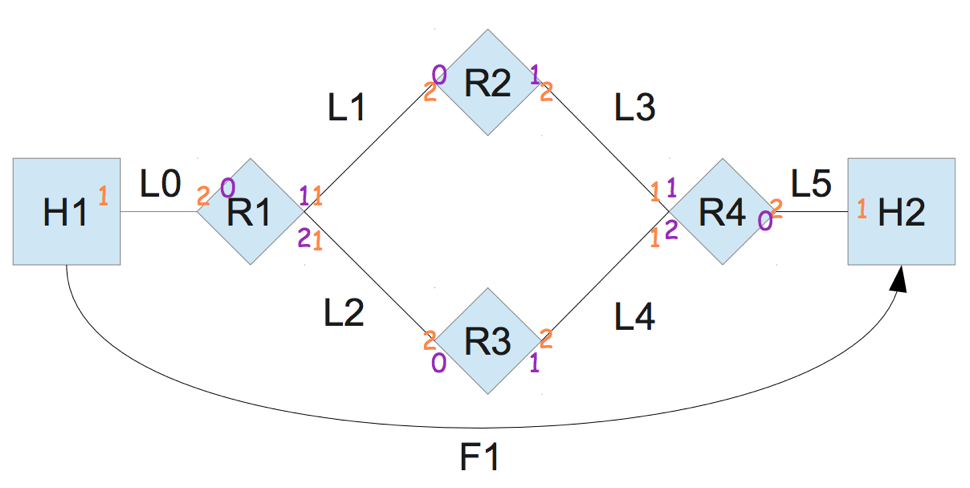
**3.6 Test Cases**

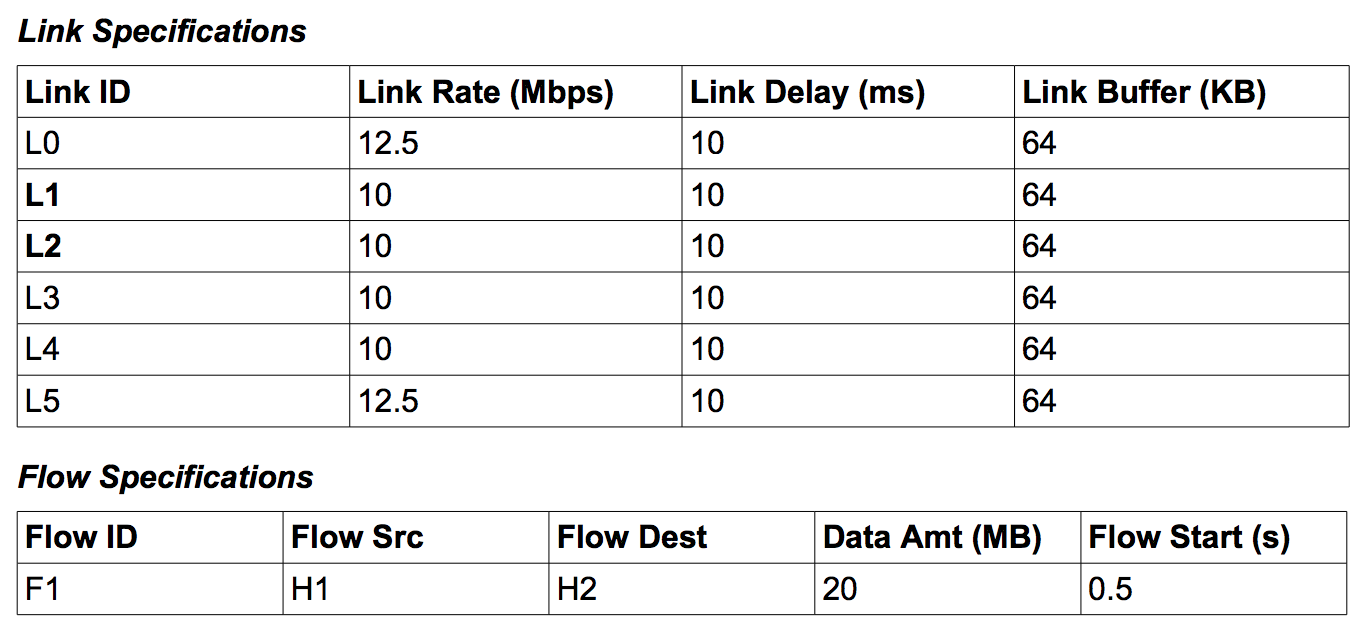
**Test Case 0**



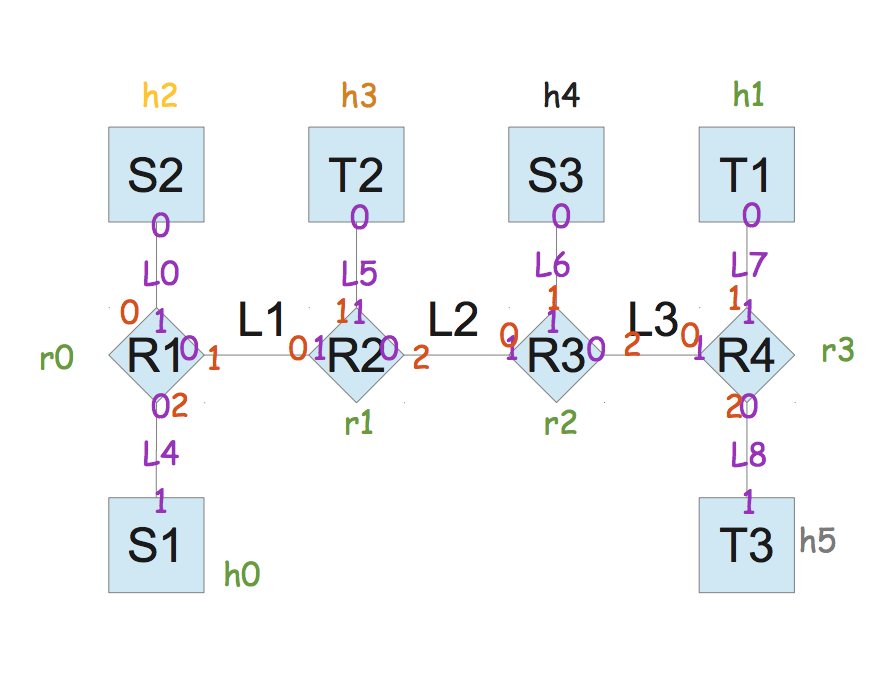


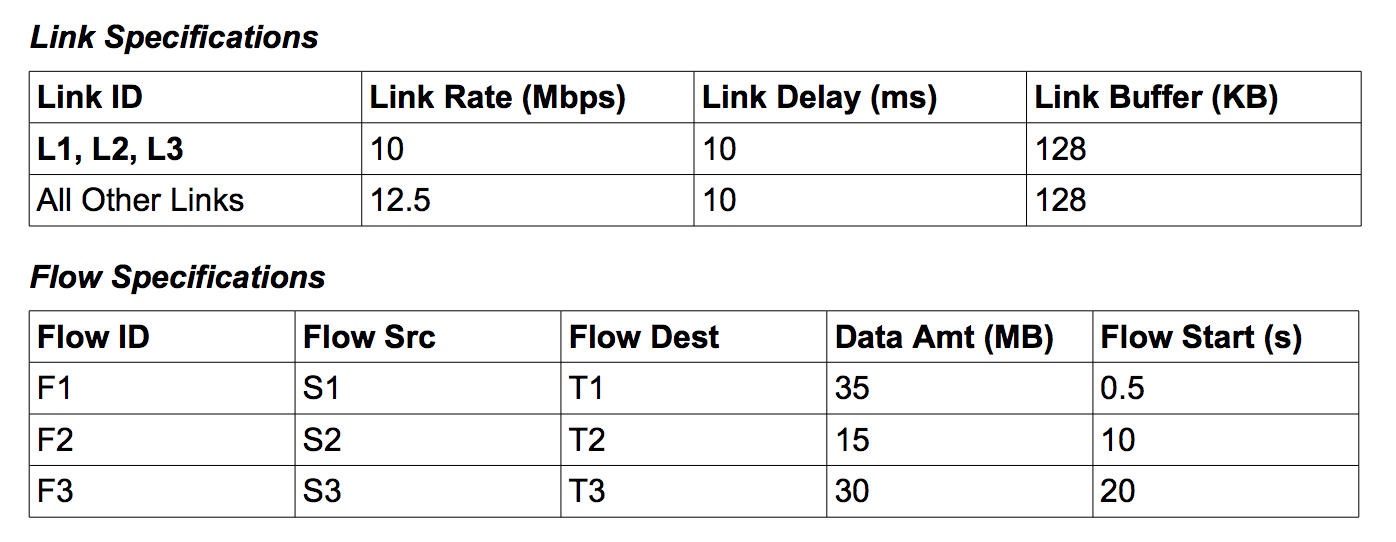
**Test Case 1**





**Test Case 2**





**4 Theoretical Analysis**

**Test Case 0**

In this test case, there is only one link and one flow between two hosts. The link rate L1 is set to be at 10Mbps, the link delay is 10ms and the buffer is 64KB. The flow from H1 to H2 is the only flow to used up all the capacity of L1. The packet generator is 1024 bytes or 1KB. The throughput of the link is 10,240 packets/s. The queuing delay is calculated by alpha divided by the throughput of the flow, which is equal to approximately 4.88 ms, while the alpha in FAST TCP is 50. The average queuing packets is calculated by the throughput times the queuing delay which equals to 50 packets.

**Test Case 1**

Similar to test case 0, there is one flow from H1 to H2. There are six links in this test cases: L0, L1, L2, L3, L4, L5 connecting in between of H1 and H2. These links create two path, and there are two ways to go from H1 to H2 each via one of the path. Links L0, L1, L3, L5 create the first path. Links L0, L2, L4, L5 create the second path. F1 will uses up all the capacity of one of these path

As we calculated in test case 0, throughput of the flow is 10240 packets/s. Since the link rate of L0 is 12.5 Mbps, greater than the link rate of both L1 and L2, which is 10 Mbps, there will be bottleneck at node that has links L1 and L2. On the other hand, link L5 has link rate of 12.5 Mbps, greater than link rate of L3, L4. As a result, the packet travel thoroughly these links without getting delay. In other word, only L1 and L2 will have a queue. The queuing delay is calculated by alpha divided by the throughput of the flow, which is equal to approximately 4.88 ms. The average queuing packets is calculated by the throughput times the queuing delay which equals to 50 packets.

**Test Case 2**

In this test case, there are three flows:

* Flow 1: S1 🡪 T1 starts at t = 0.5s
* Flow 2: S2 🡪 T2 starts at t = 10s
* Flow 3: S3 🡪 T3 starts at t = 20s

At steady state, the window size is the same. The rate at steady state is:

* At time t = 0.5s to 10s, there is only F1 active. F1 will use the capacity of L1, L2, L3 to connect S1 to T1. The steady state throughput is 10240 packet/s. The queuing delay at L1 is

The queue of L1 is 50 packets. There is no queue for L2, L3 (L2, L3 queues = 0 packet).

* At time t = 10s to 20s, there are F1 and F2 active. There will be a bottleneck at L1, since both F1 and F2 are using it. In order word, the link capacity of L1 is shared between F1 and F2. The steady state throughput is 10240 packet/s. The queuing delay at L1 is

The queue of L1 is 440 packets. There is no queue for L2, L3 (L2, L3 queues = 0 packet).

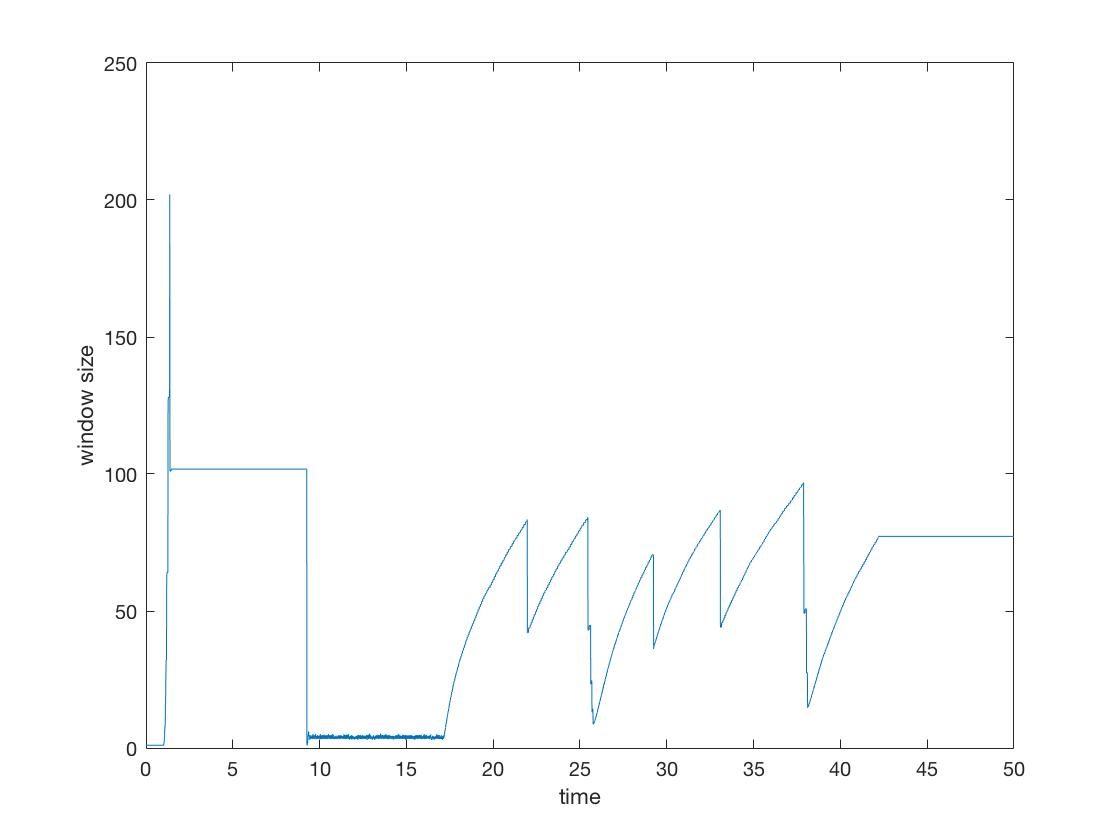
* At time t after 20s, F3 starts. All three flows are active. F1 and F3 share the link capacity of L3. F1 and F2 share the link capacity of L1. The queueing delay of L2 is zero. The steady state throughput is 10240 packet/s.

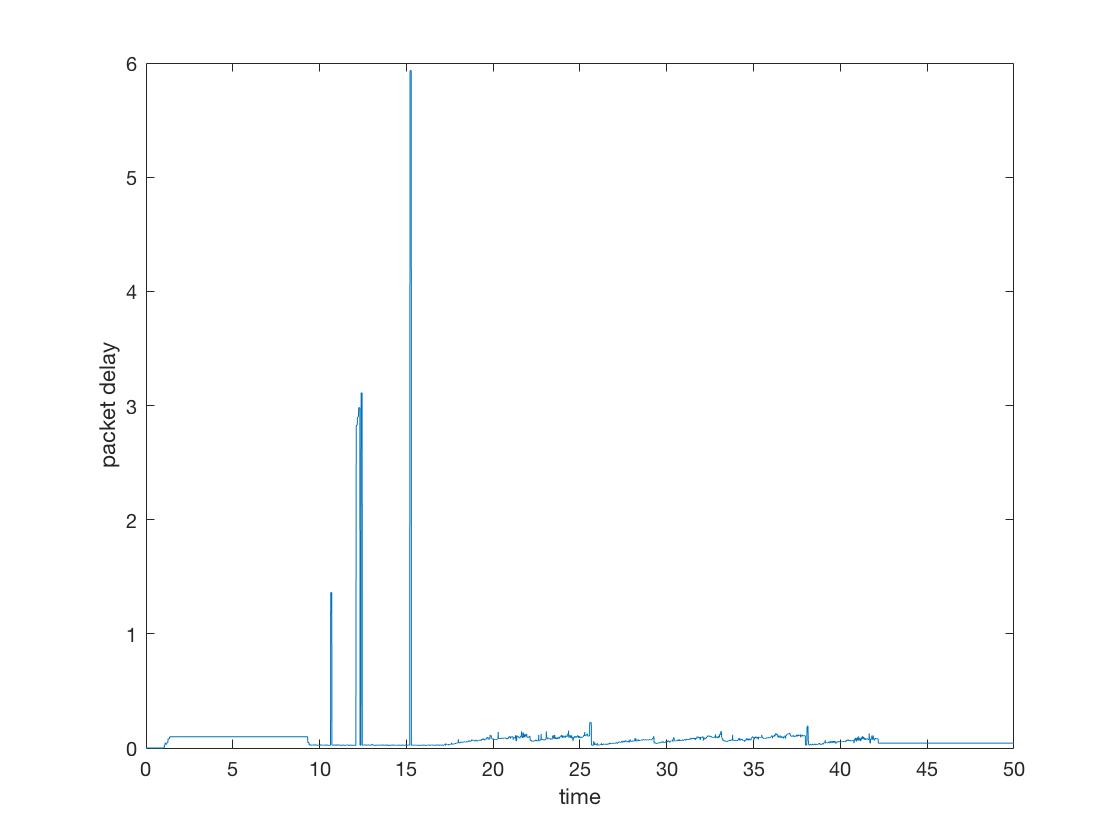
The queue of L1 is 348 packets. The queue of L3 is 297 packets There is no queue for L2 (L2 queue = 0 packet).

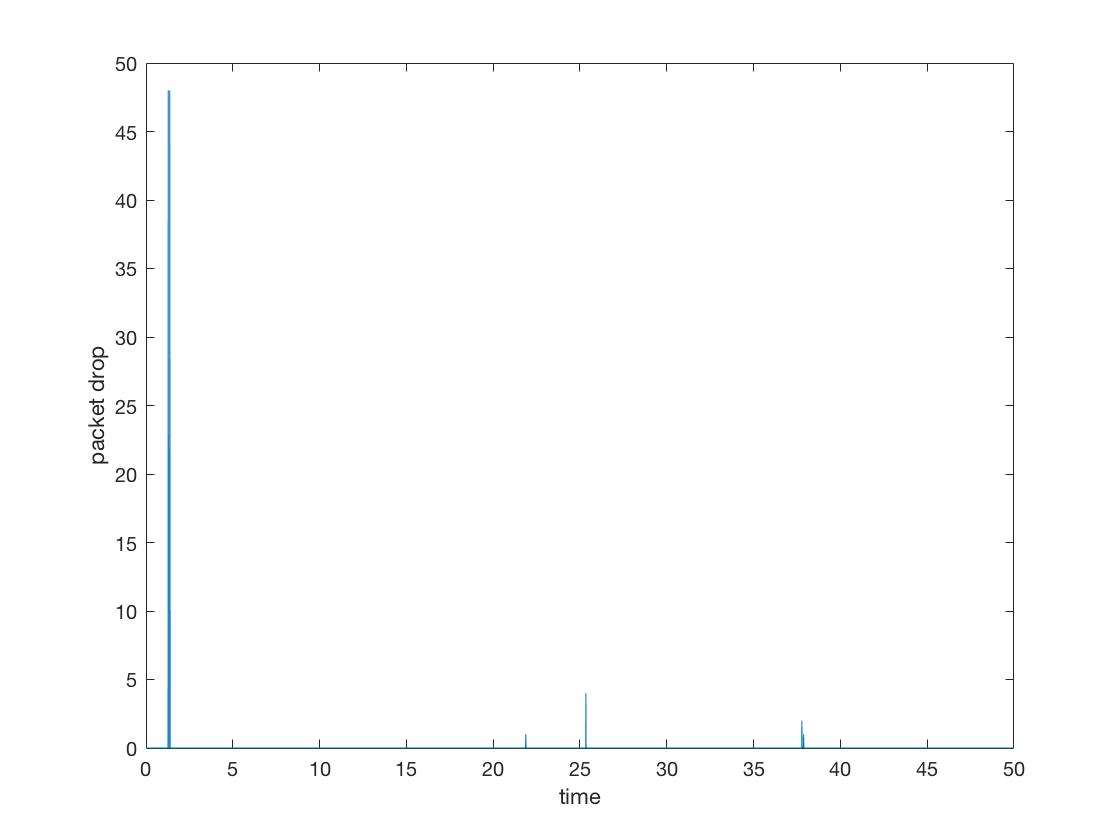
**5 Simulation Result**

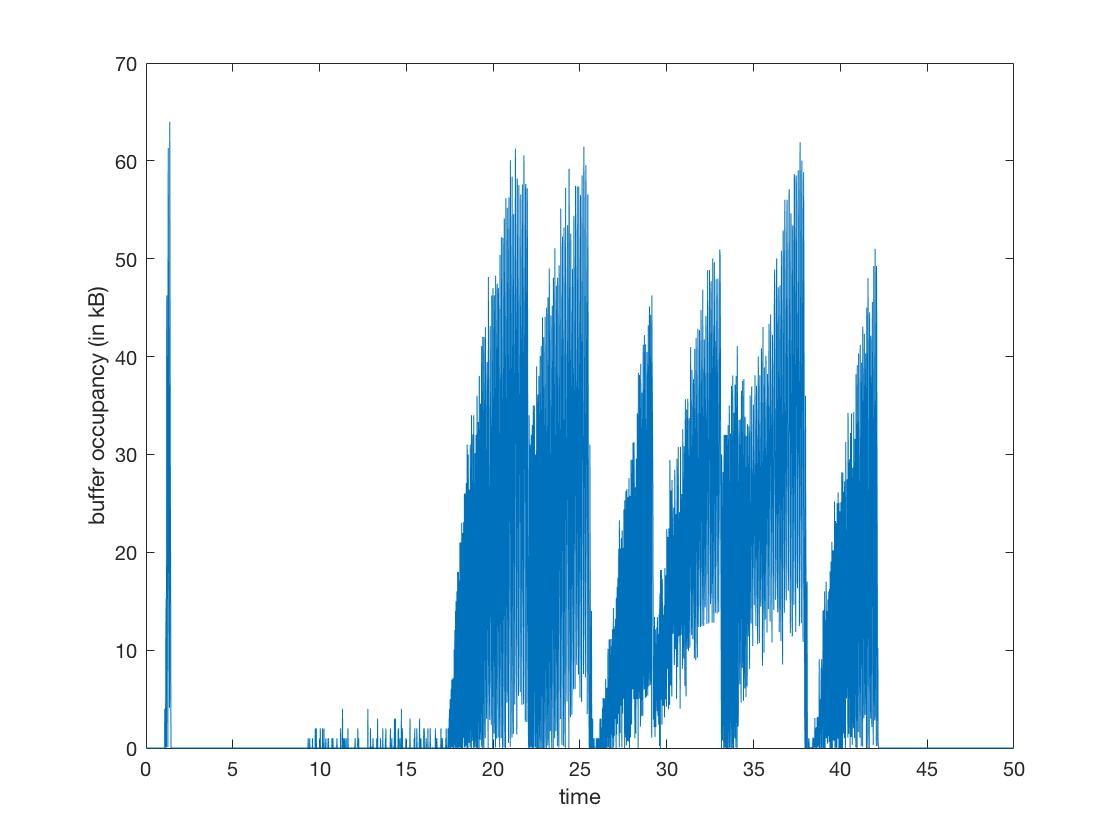
**Test Case 0**

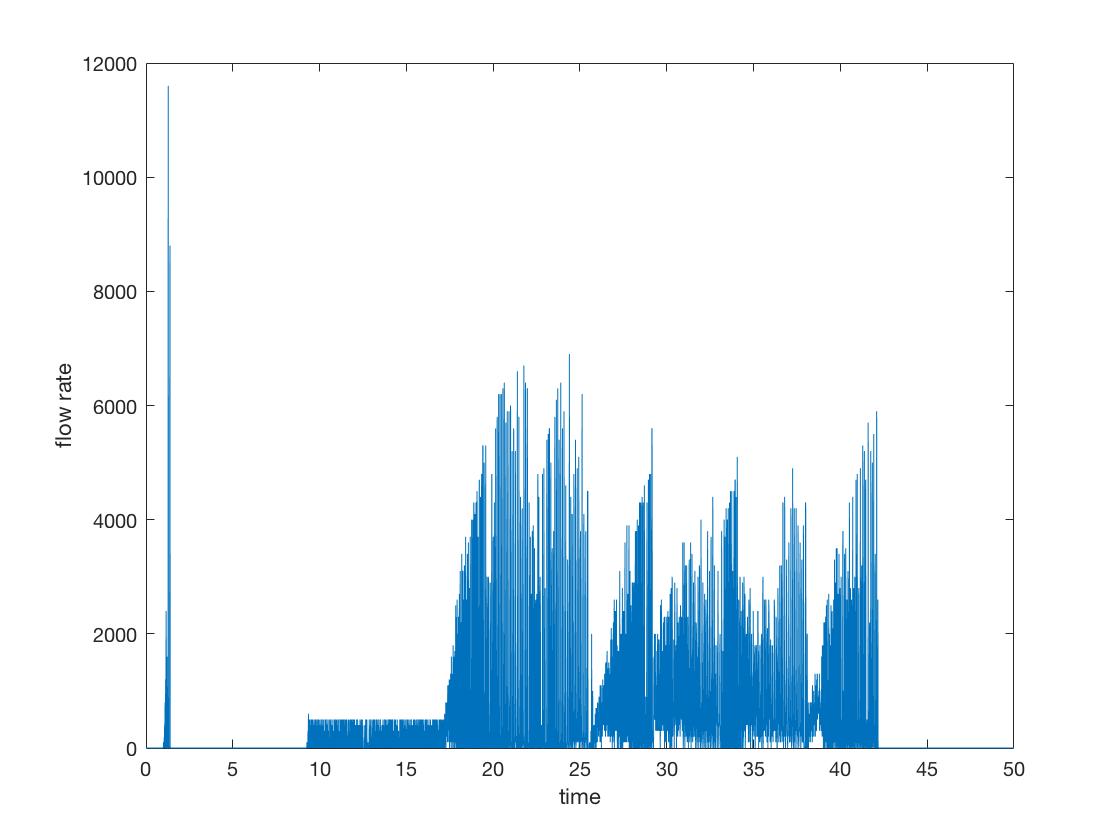
* **TCP Reno**

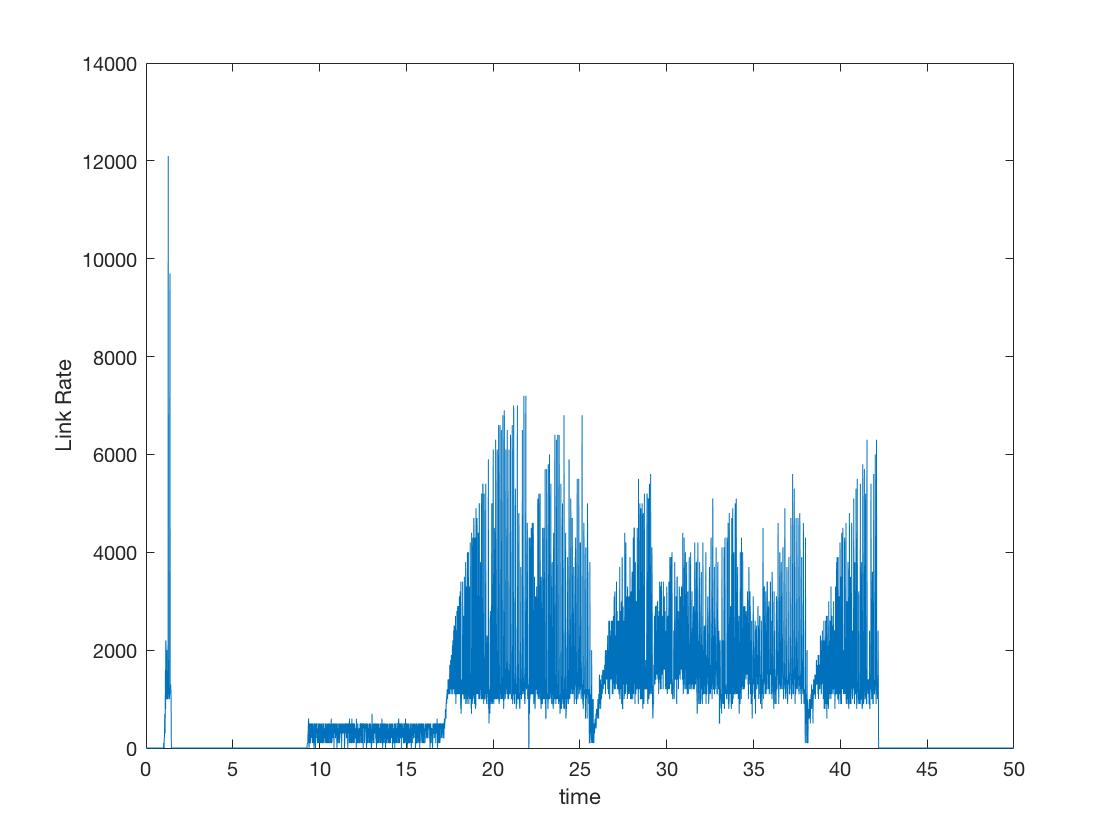




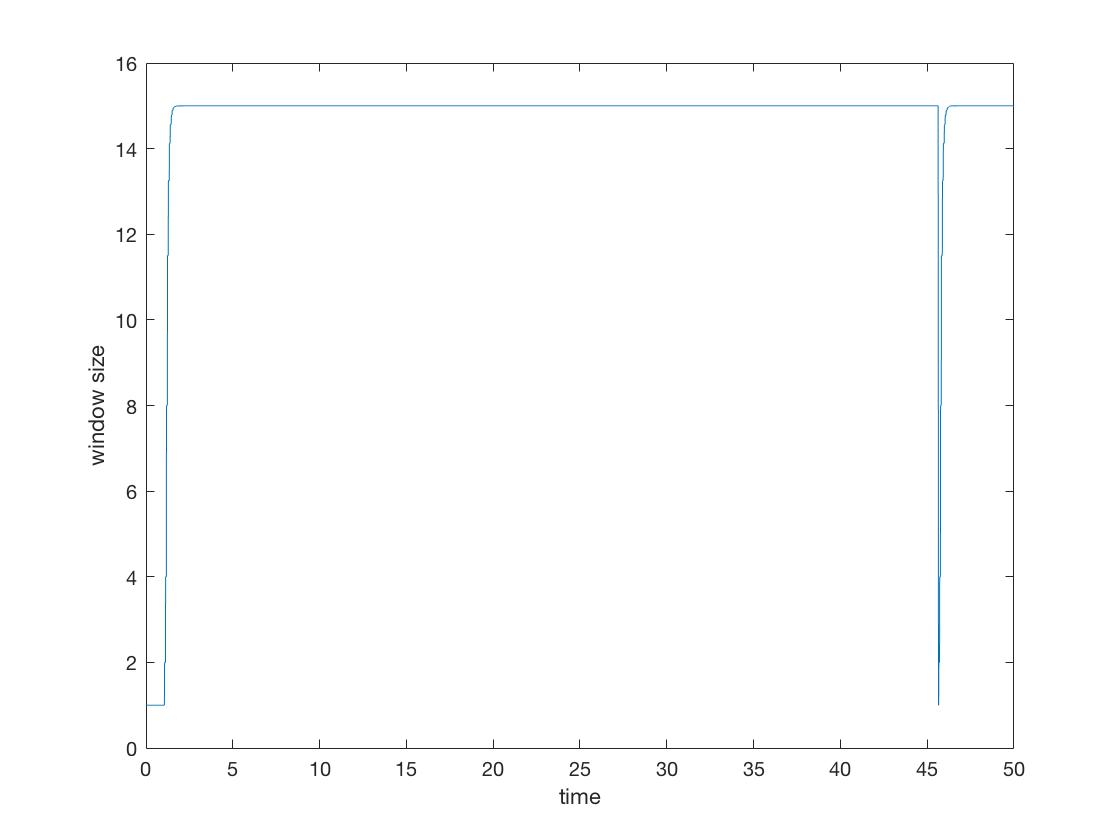


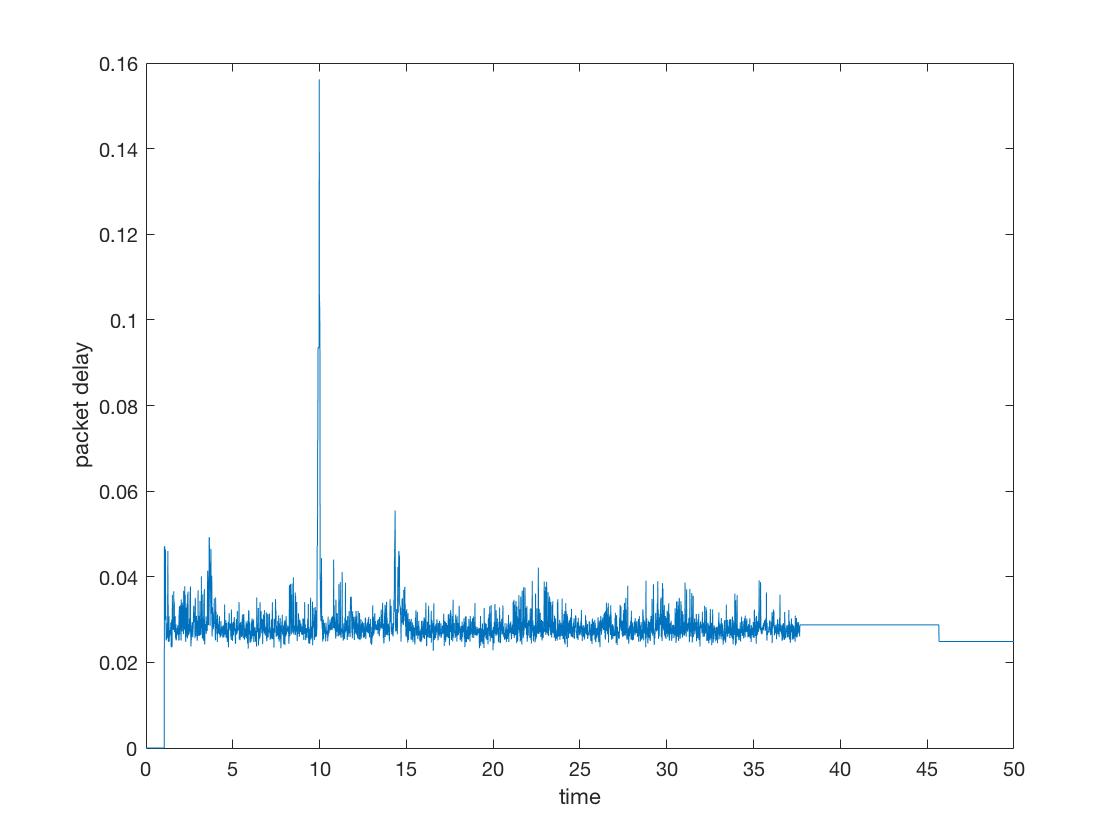


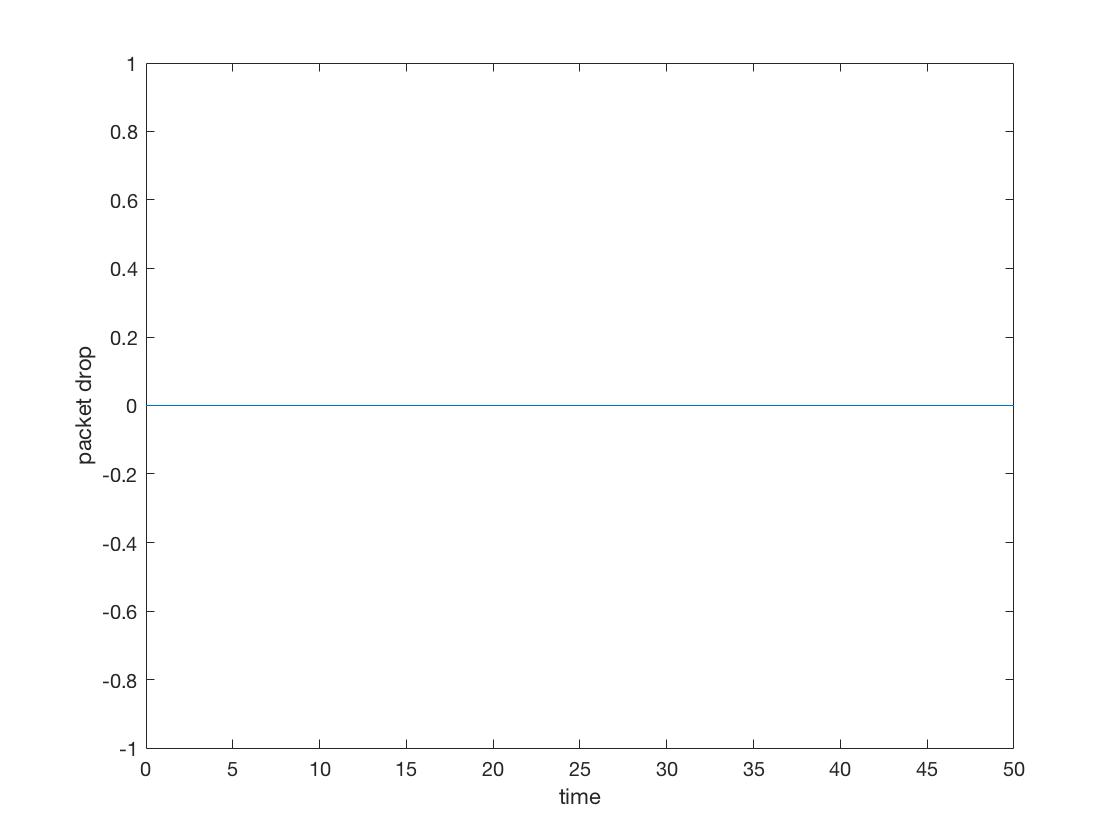


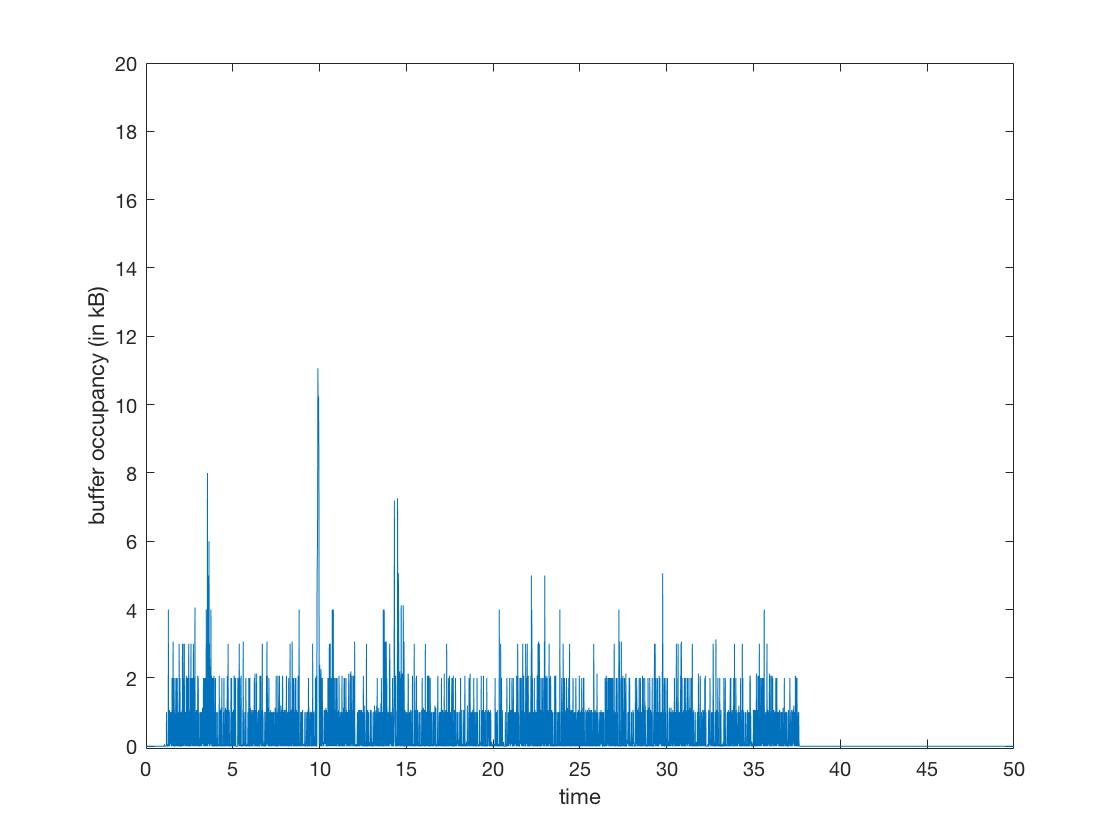


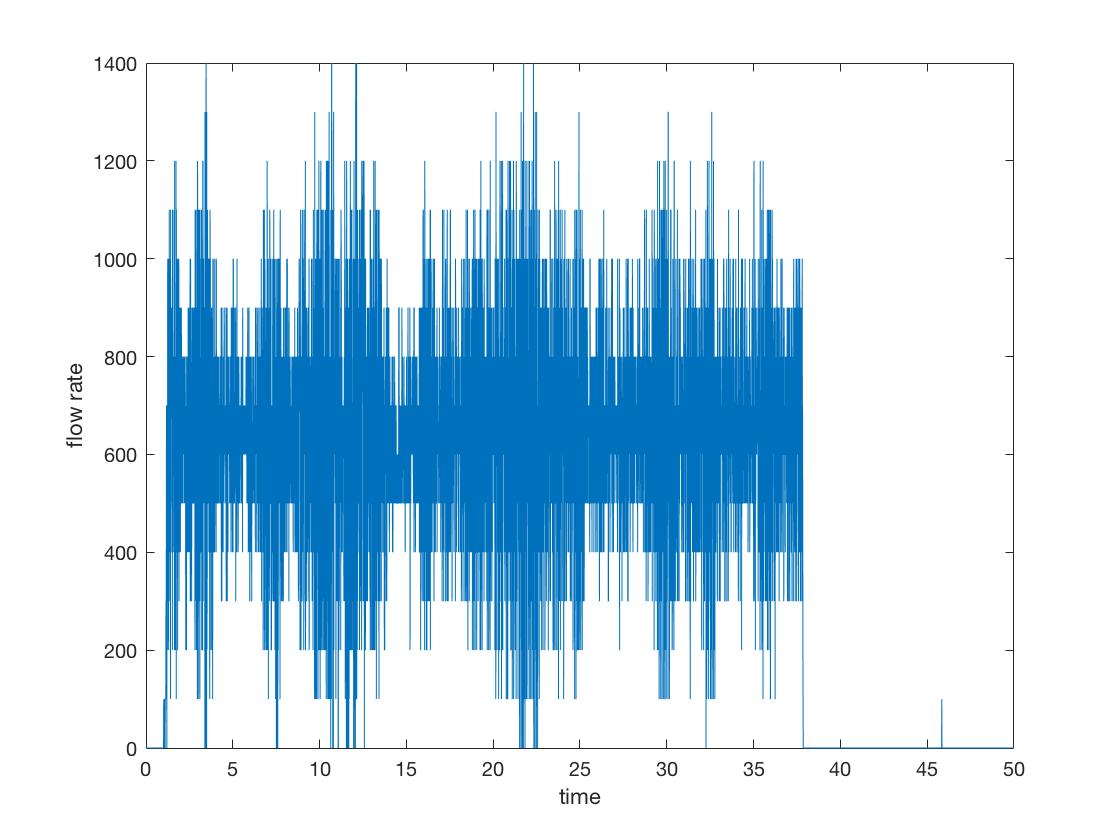
* **FAST TCP**

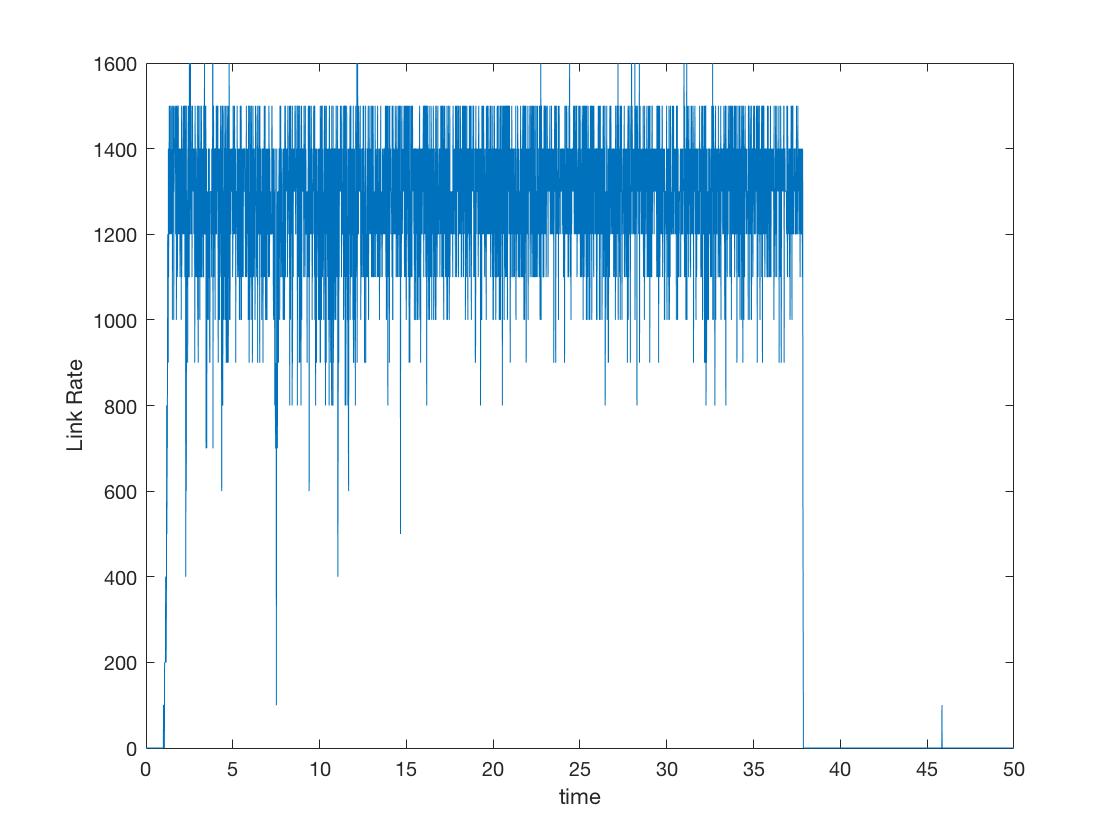






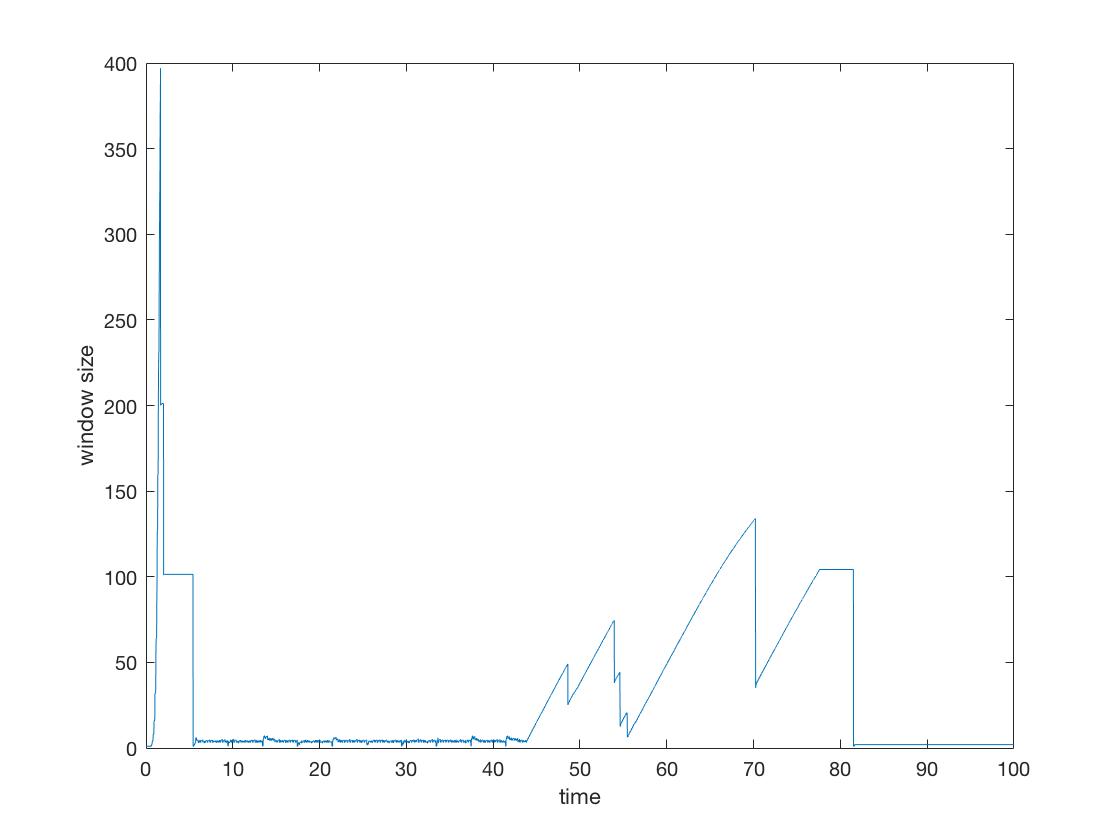


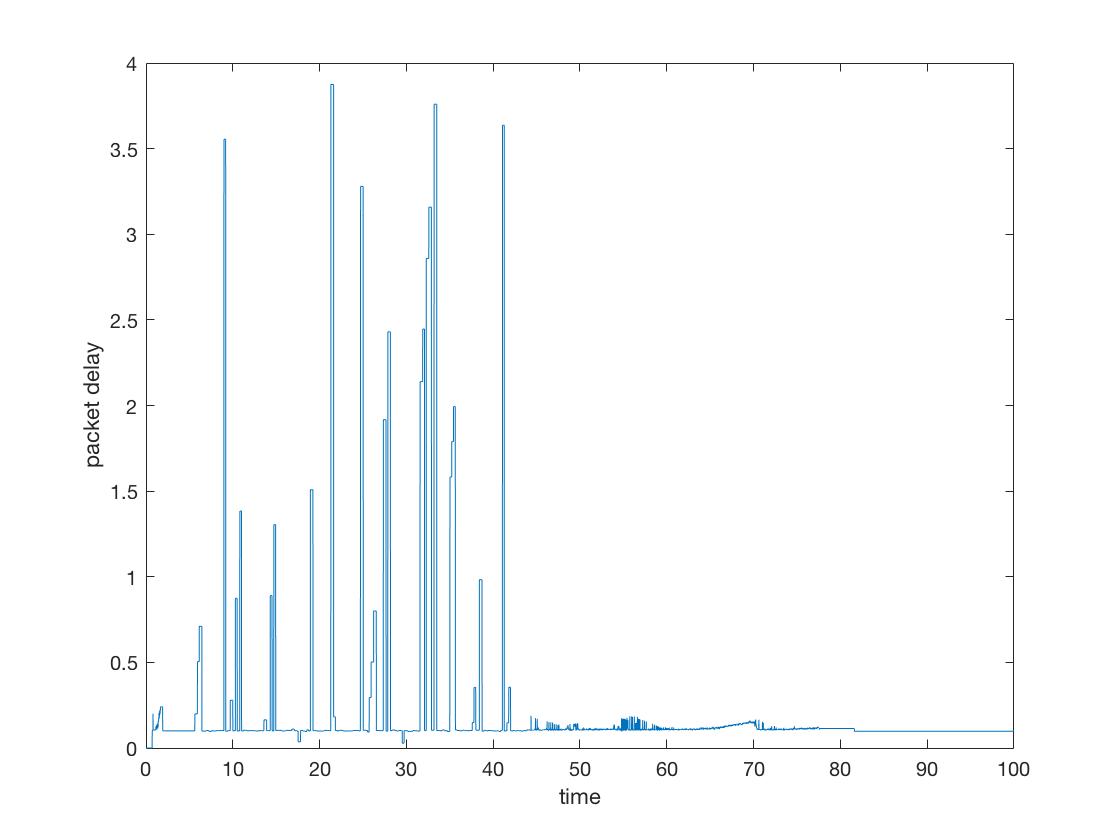


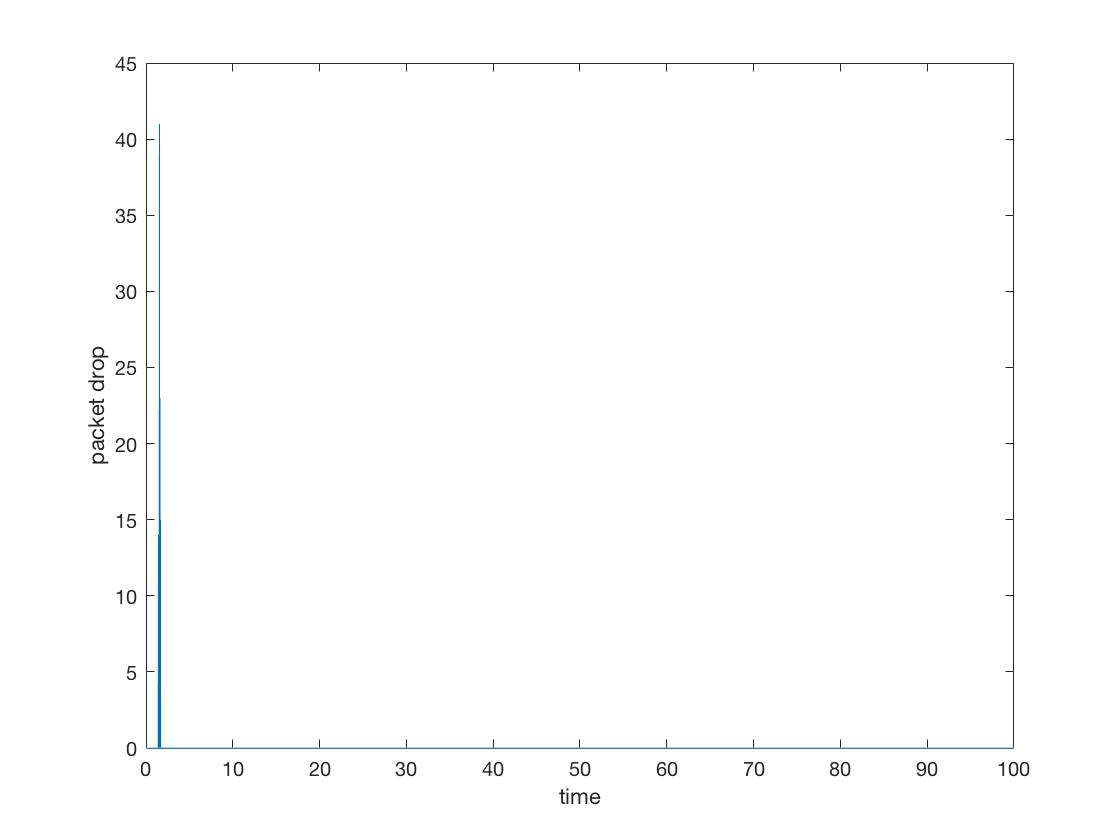


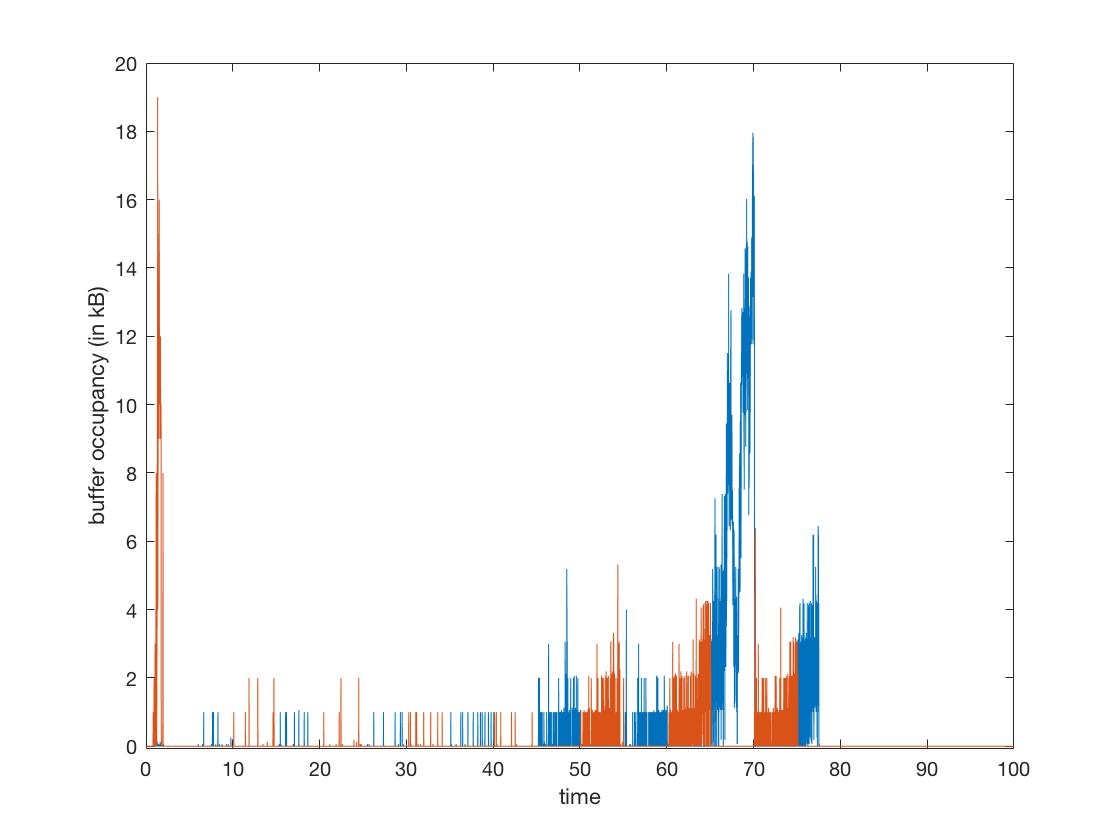
**Test Case 1**

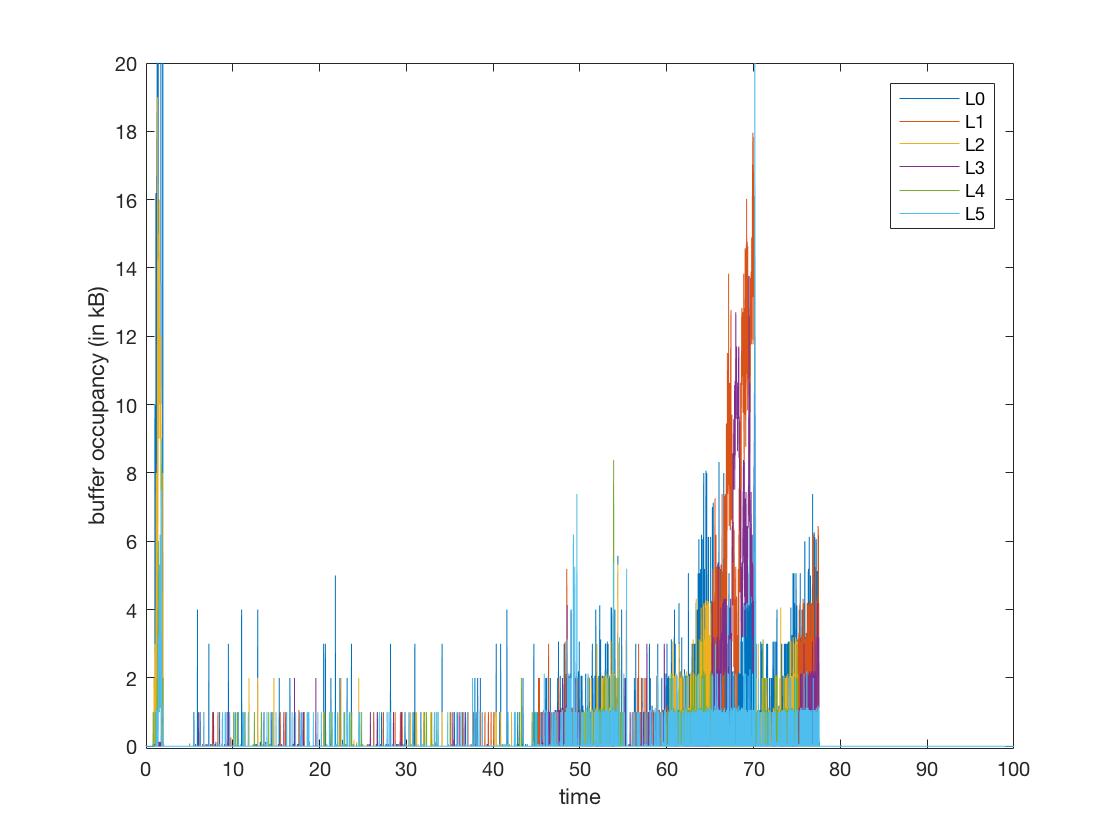
* **TCP Reno**

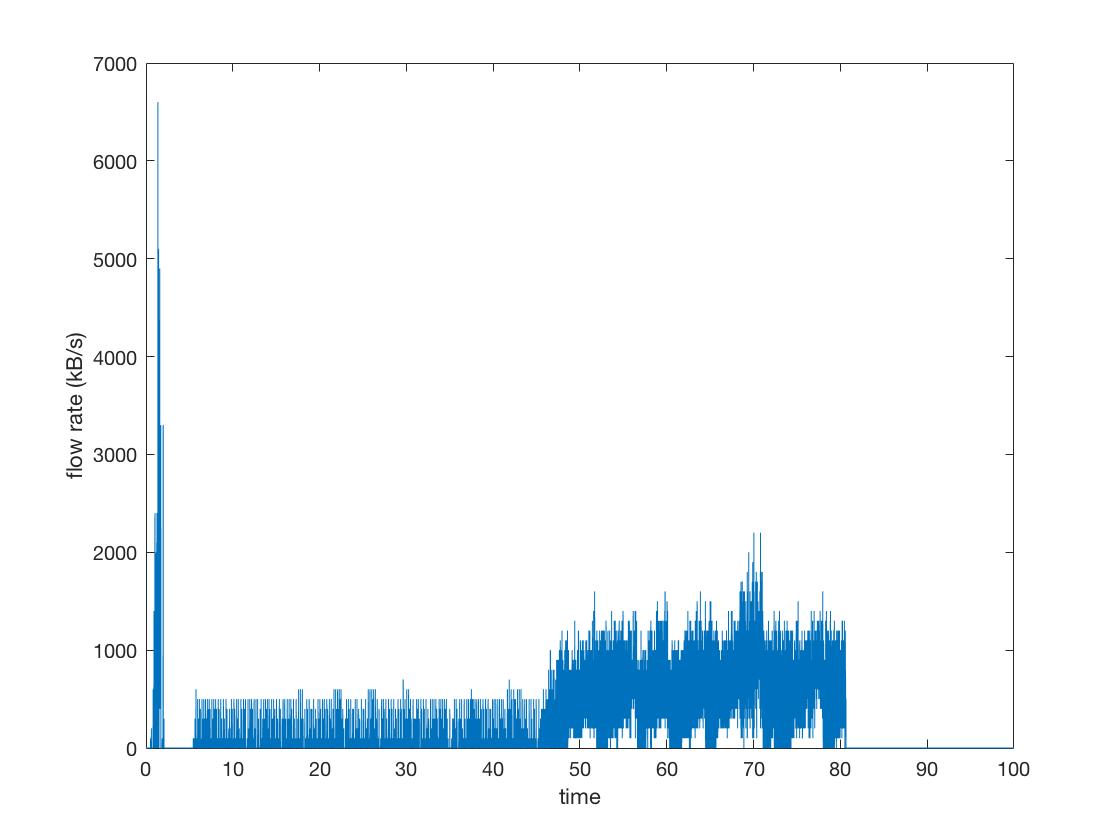
****

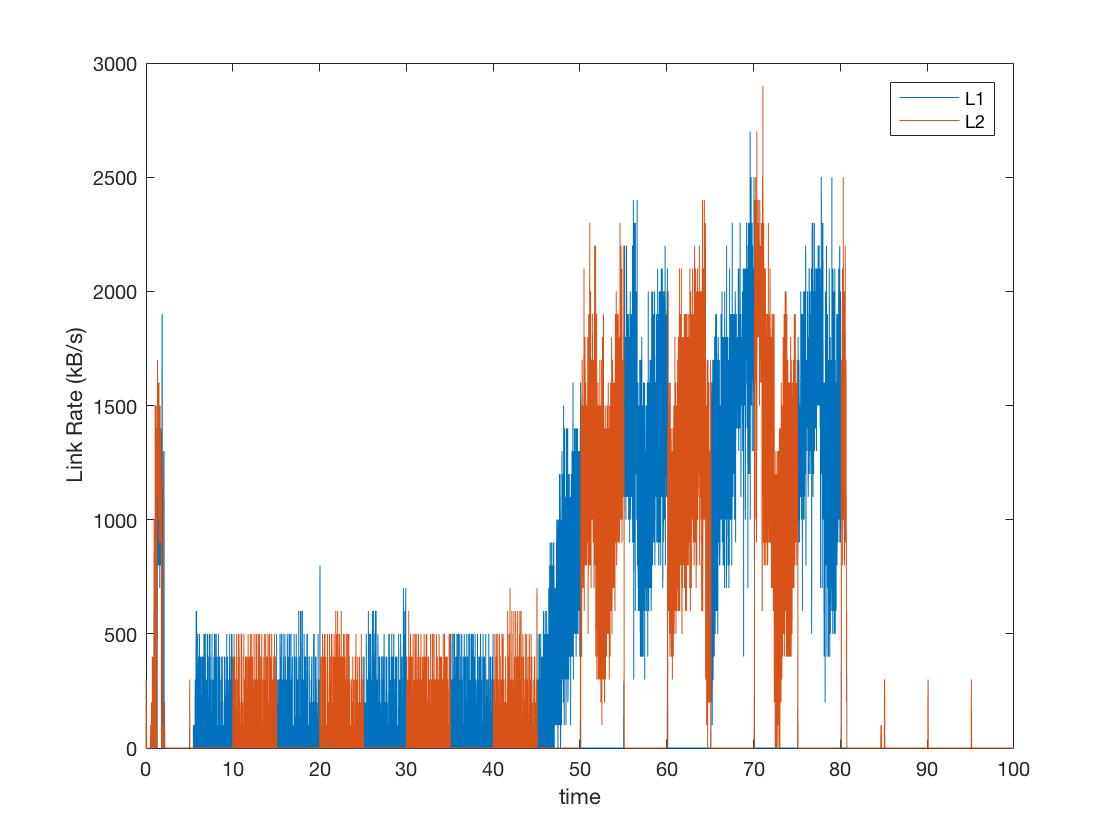
****

****

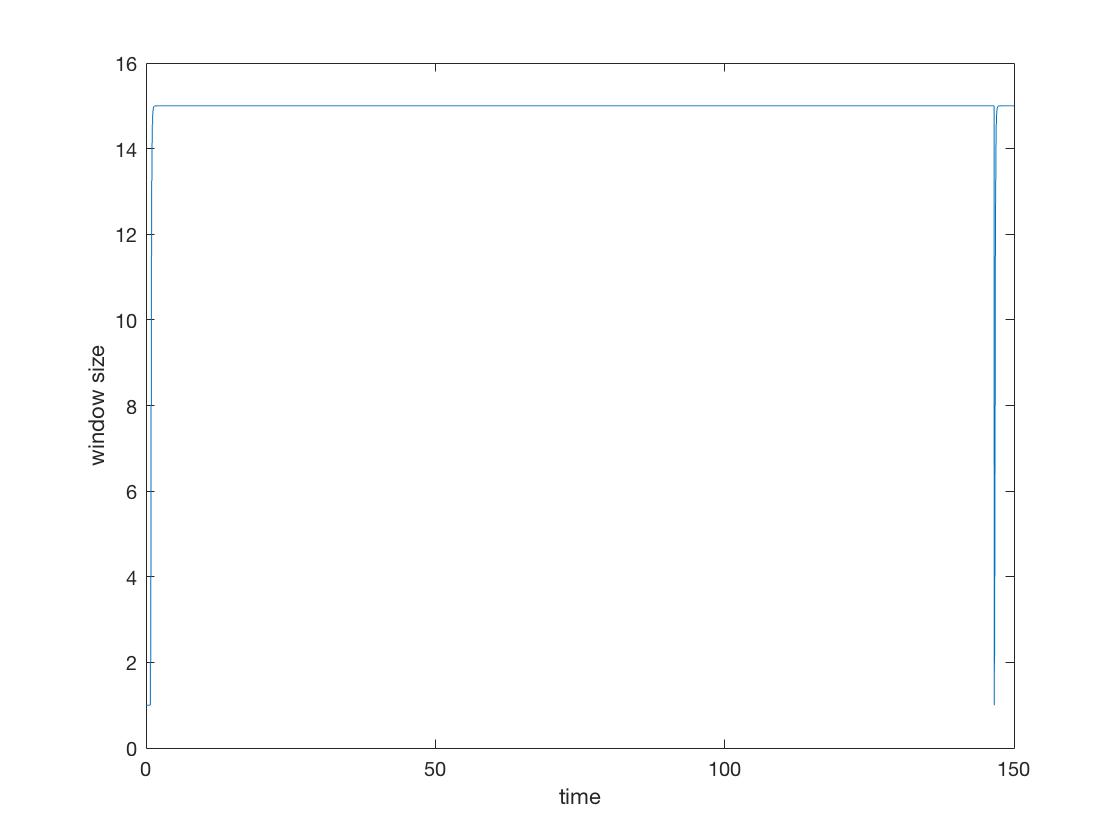
****

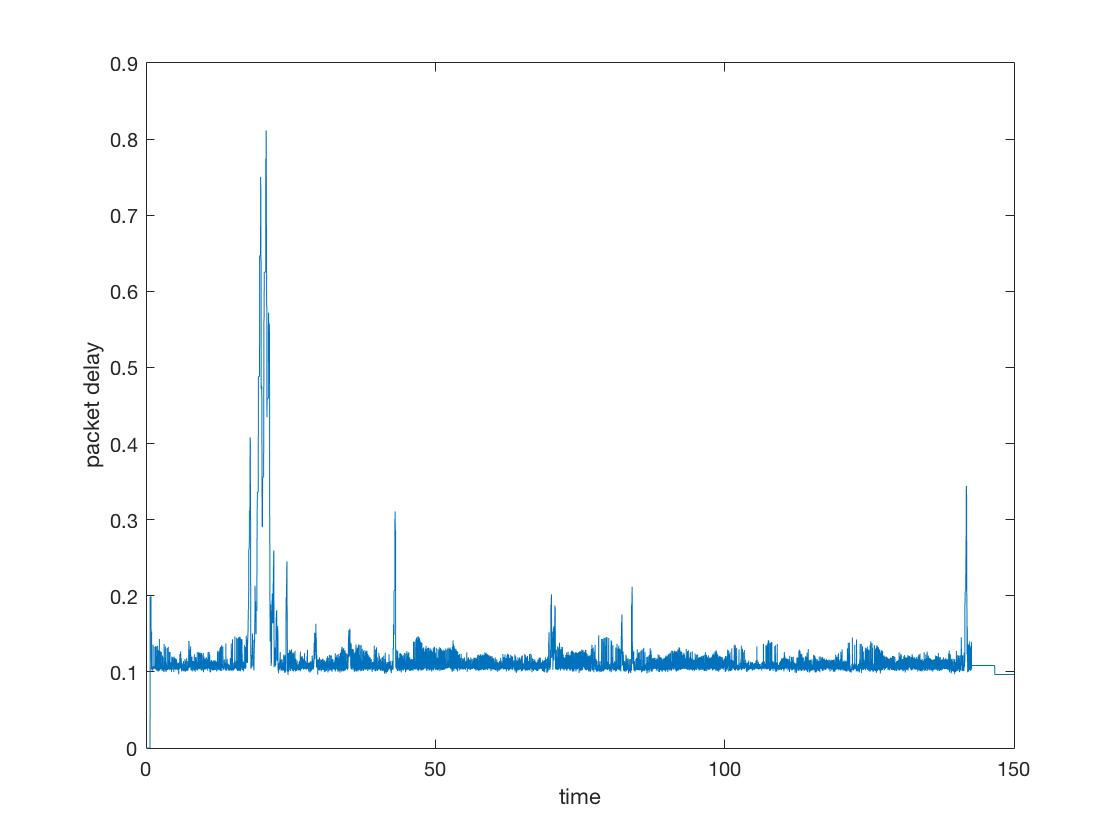
****

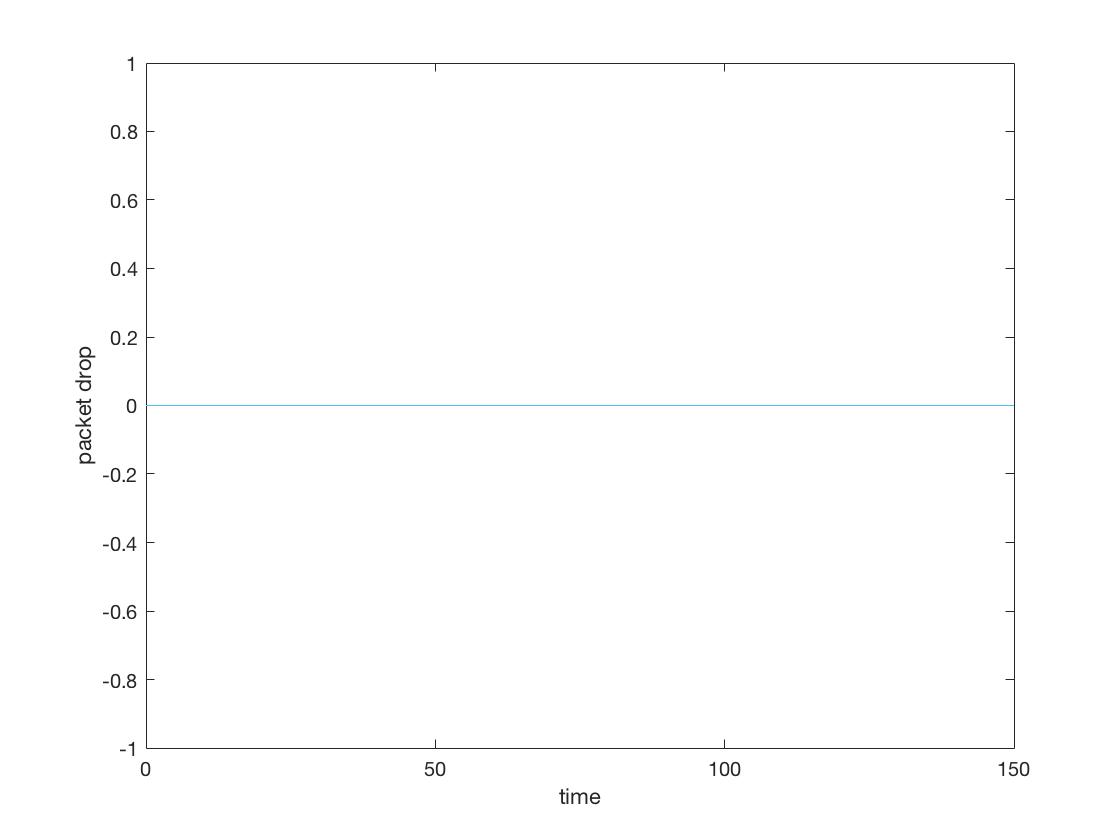
****

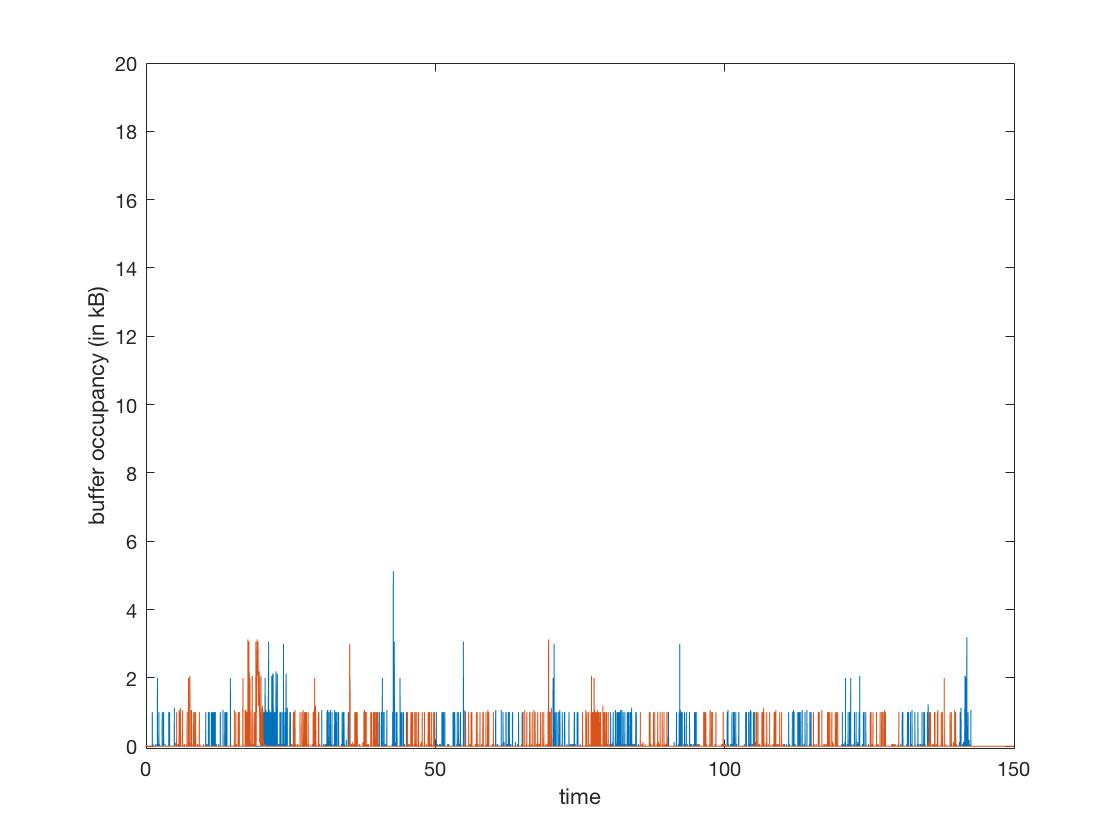
****

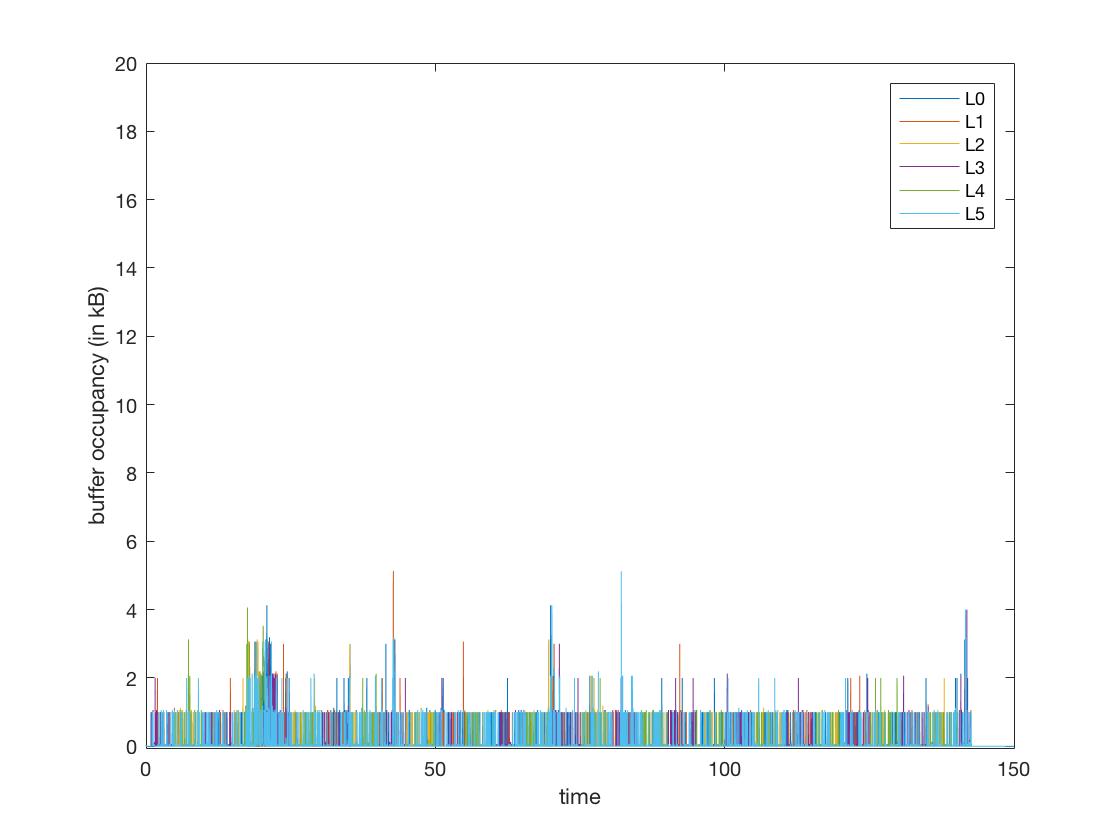
* **FAST TCP**

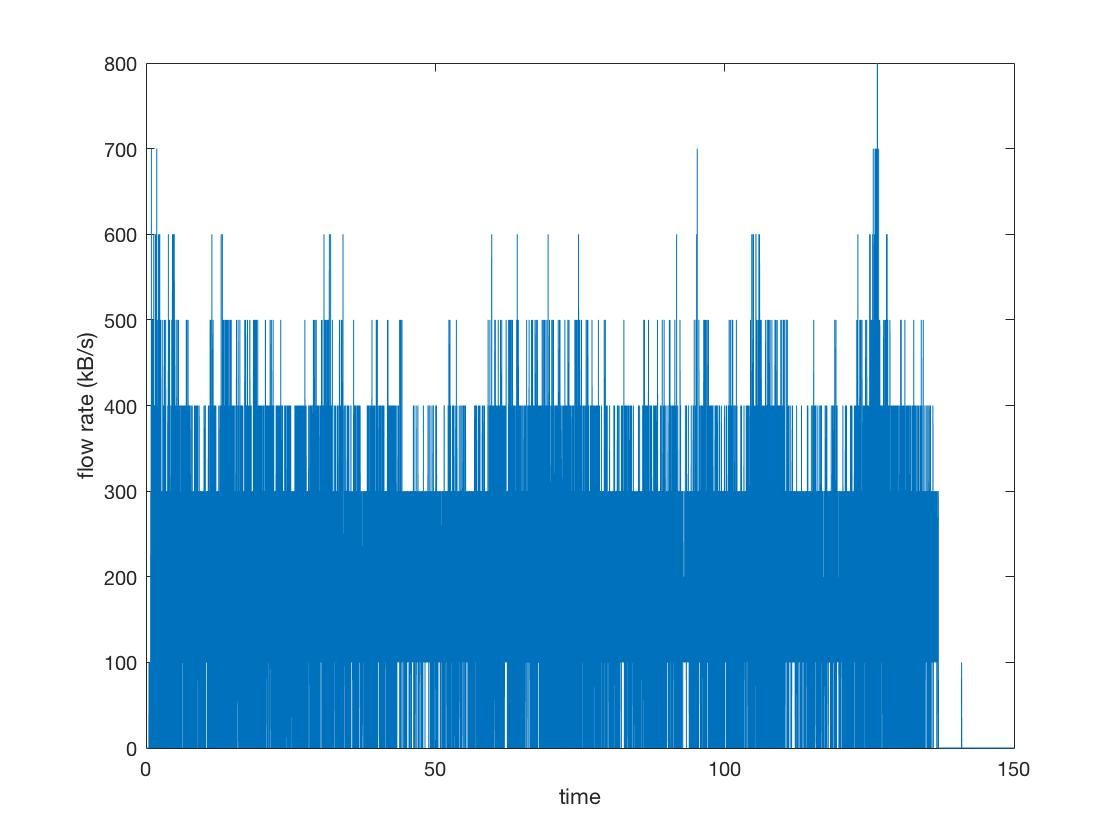
****

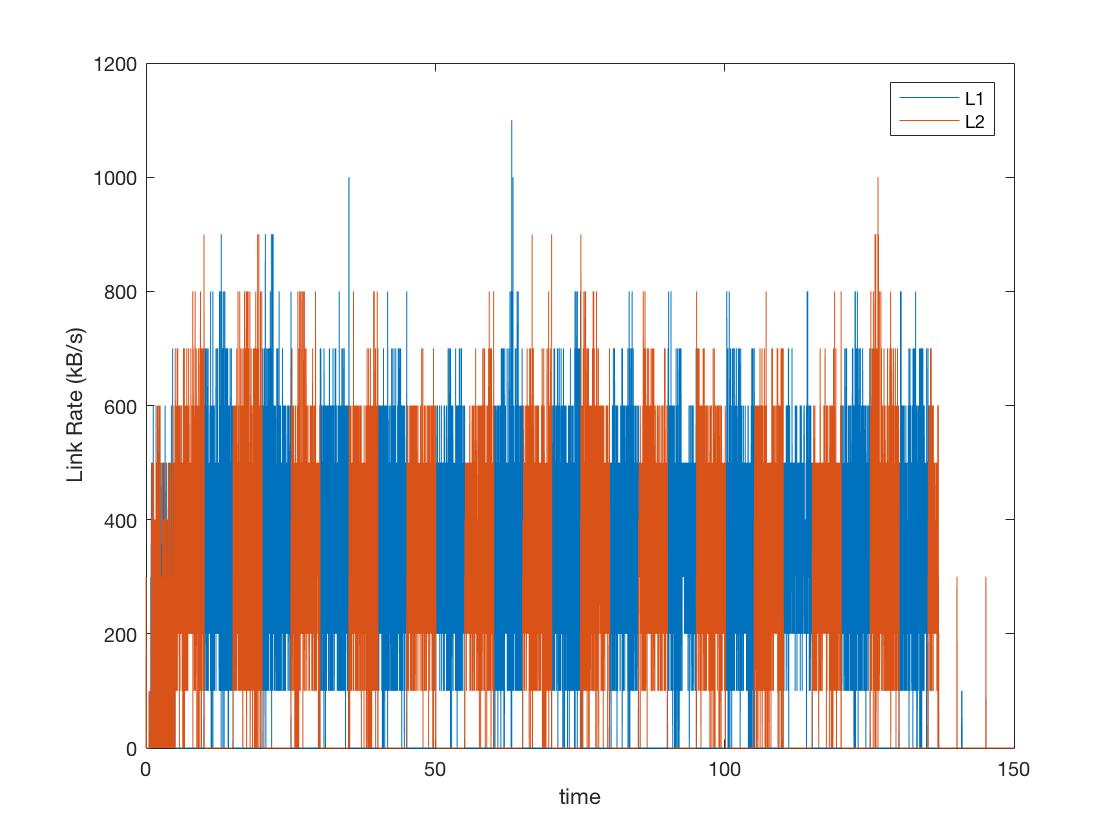
****

****

****

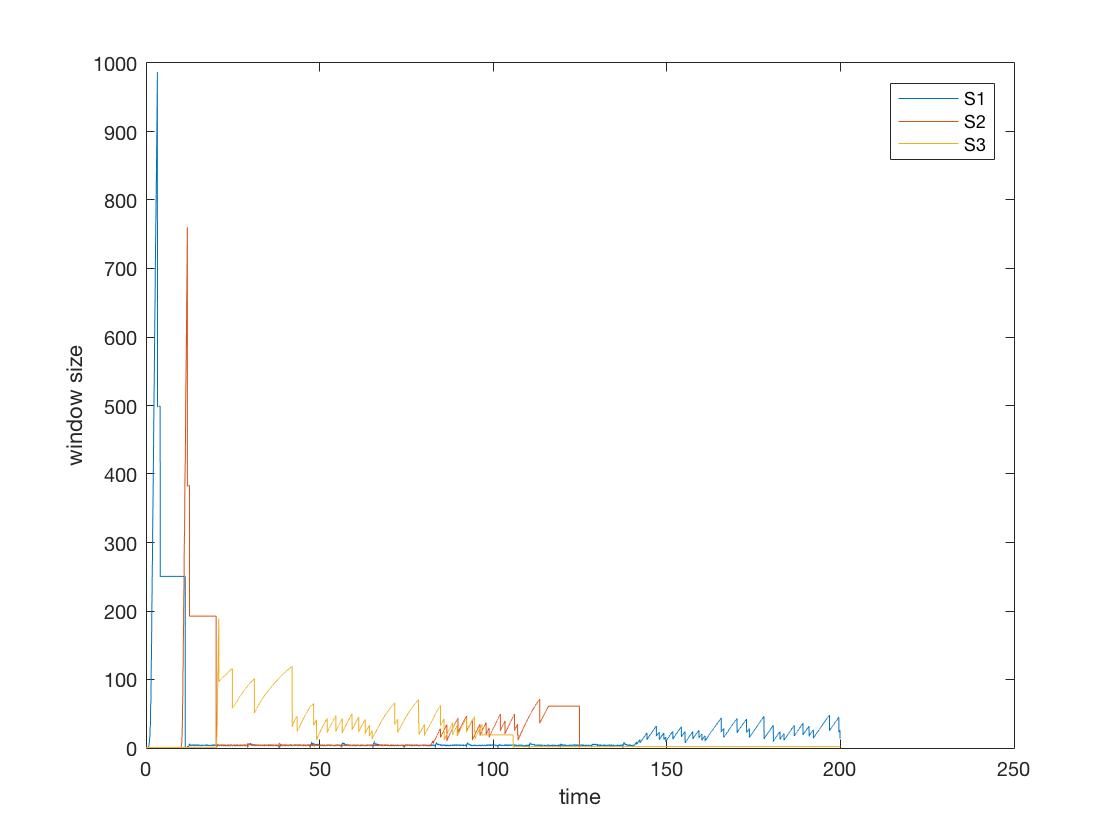
****

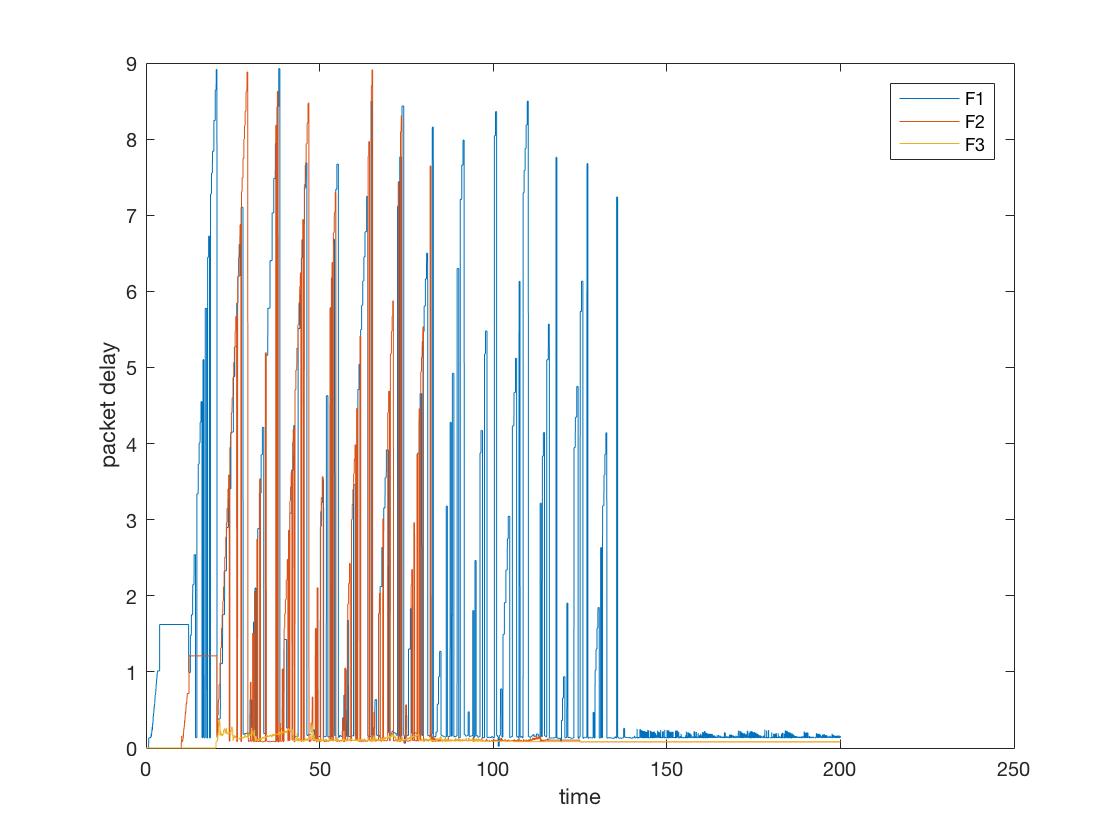
****

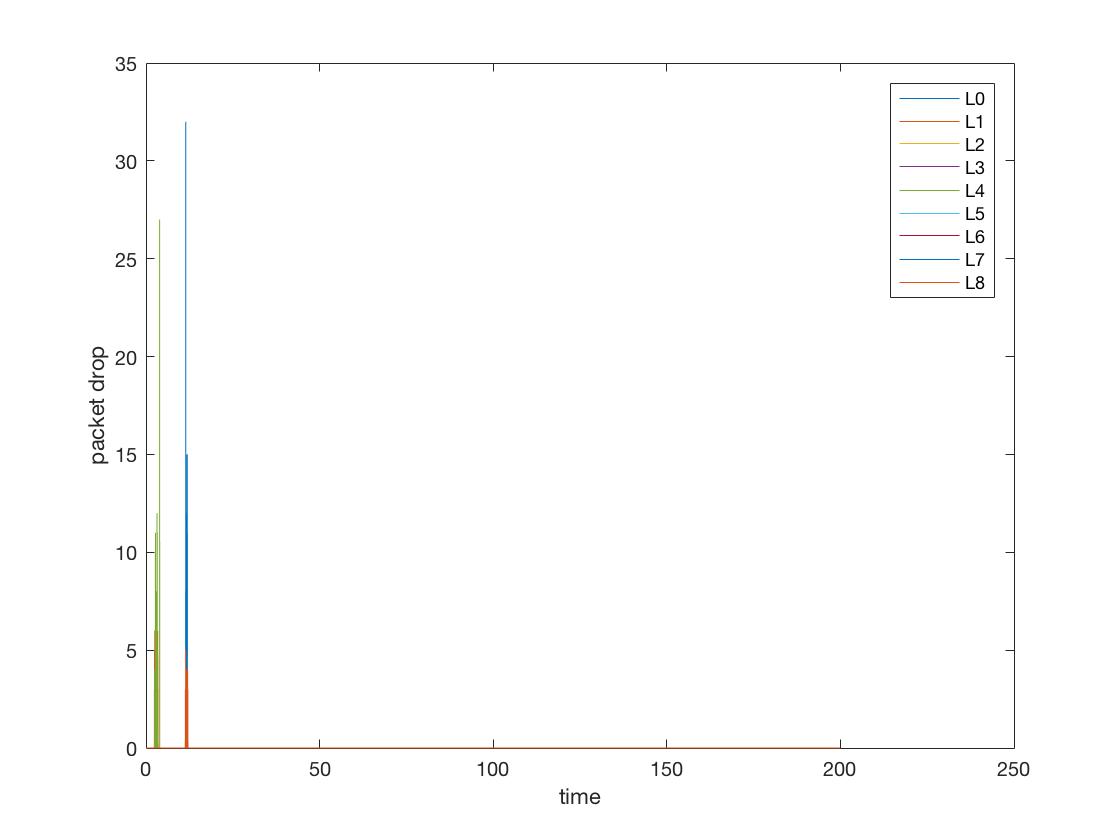
****

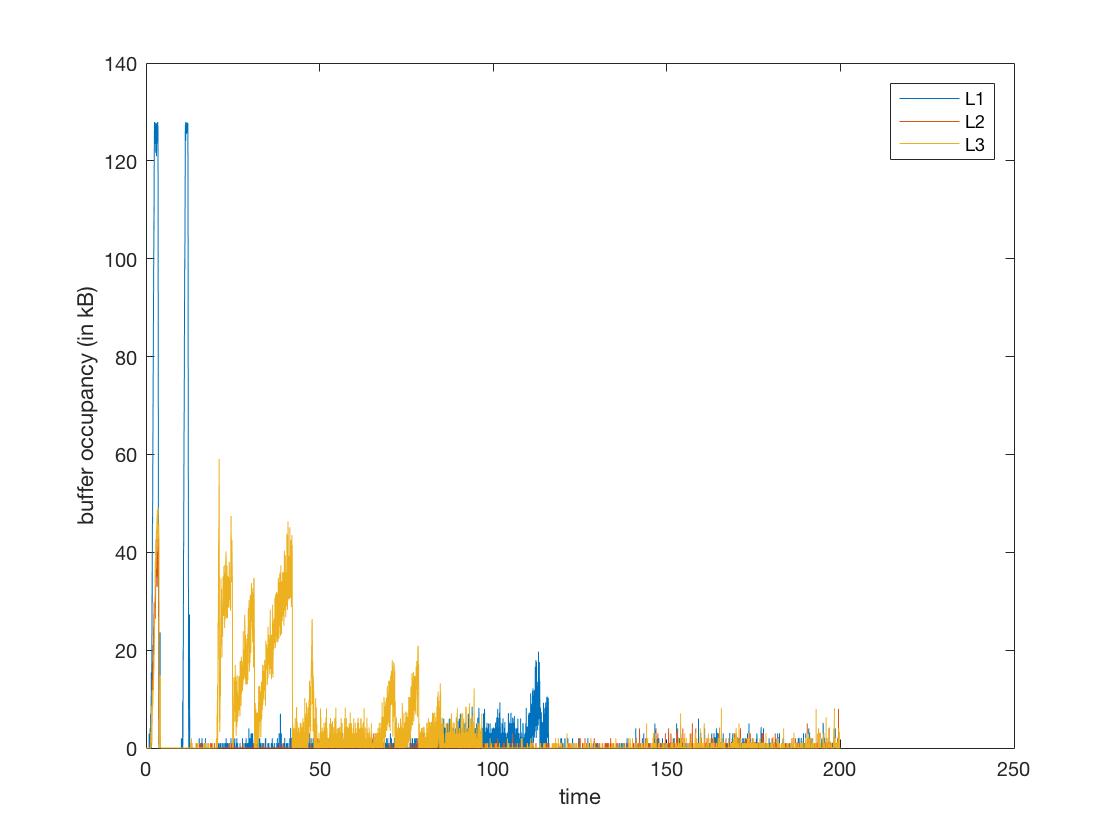
**Test Case 2**

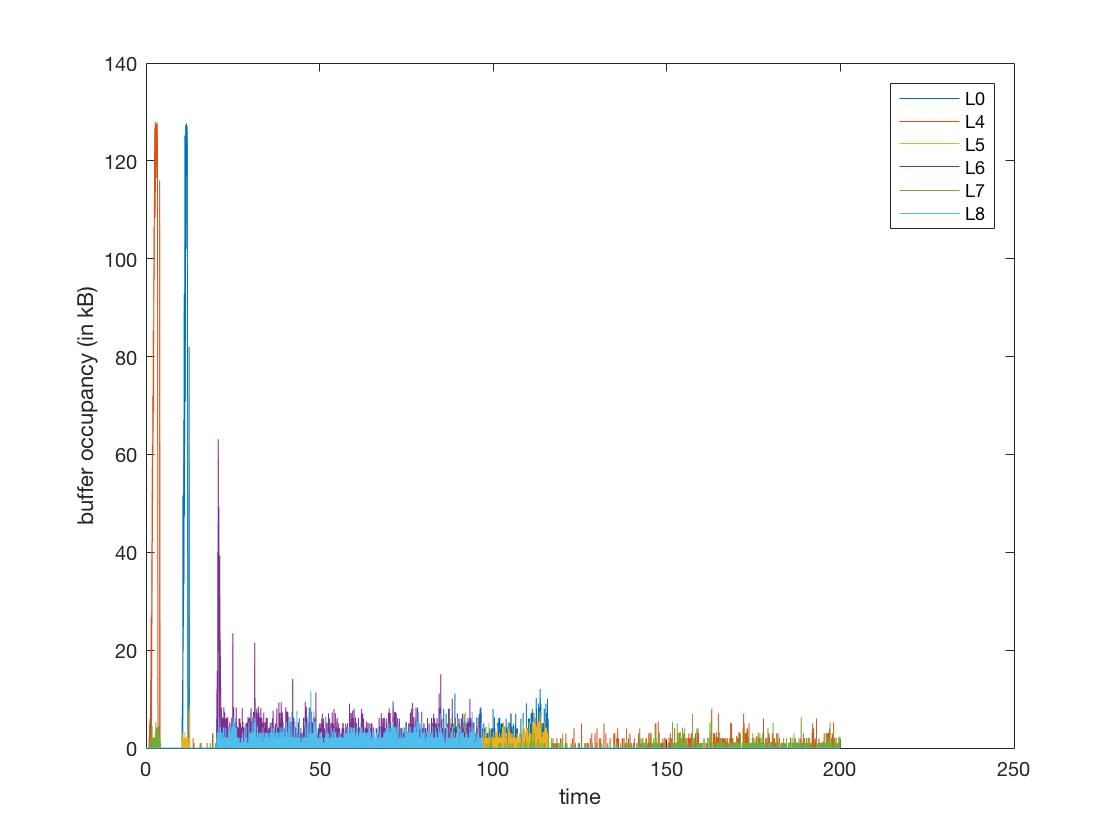
* **TCP Reno**

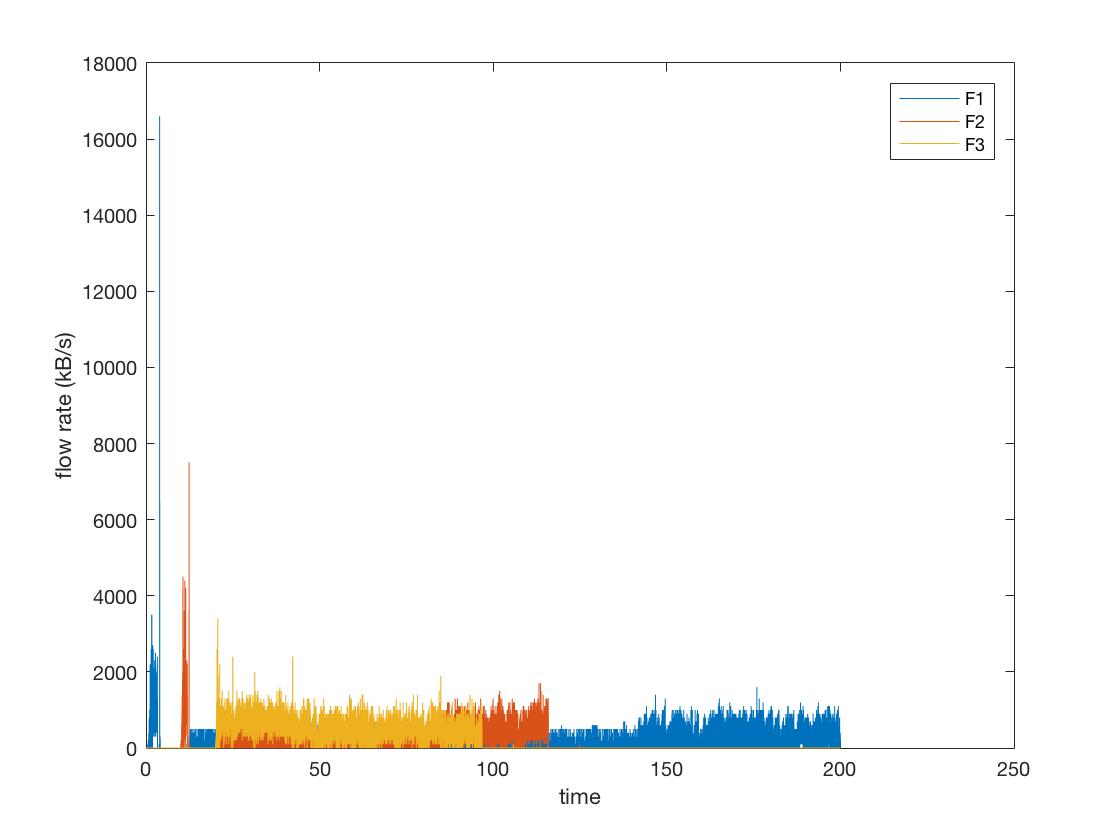
****

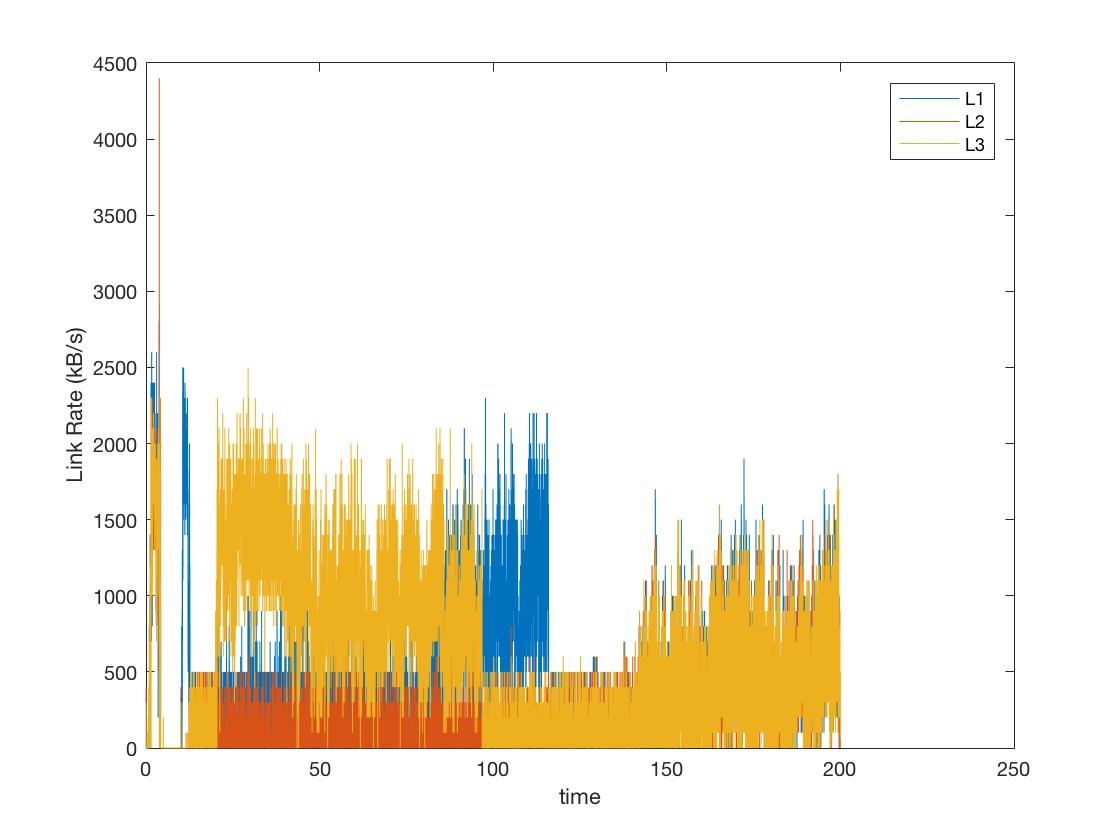
****

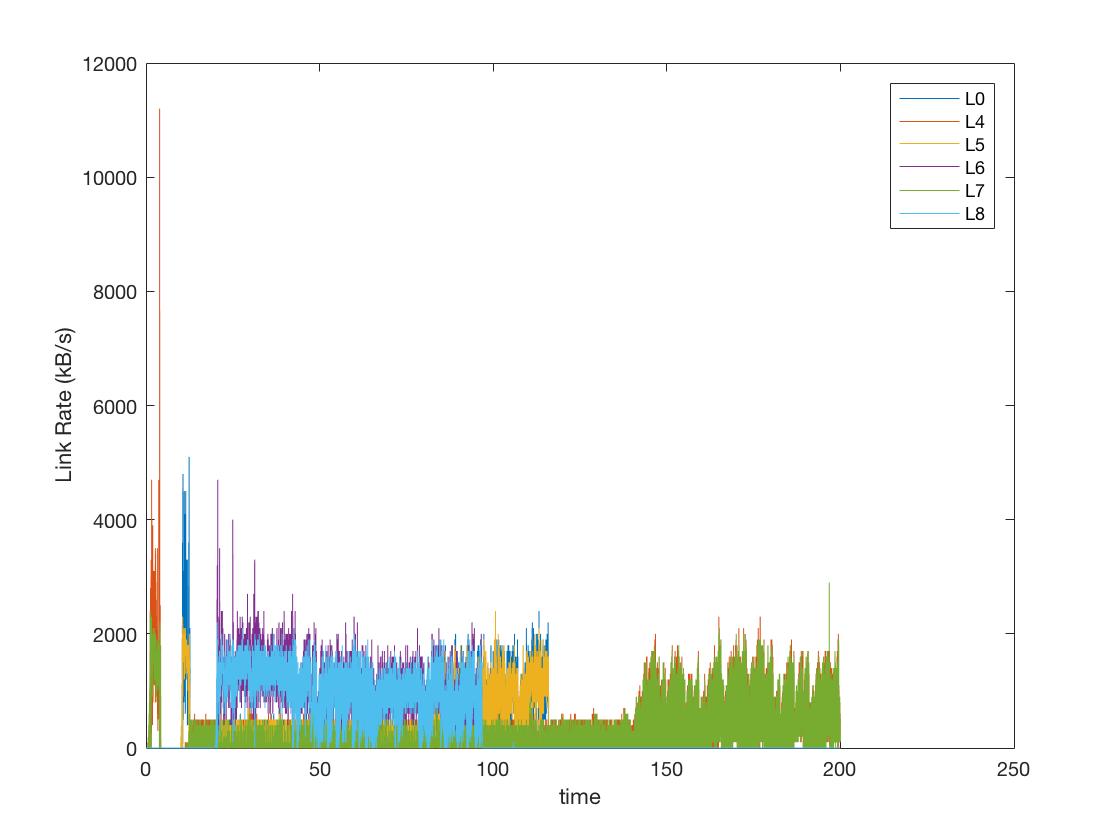
****

****

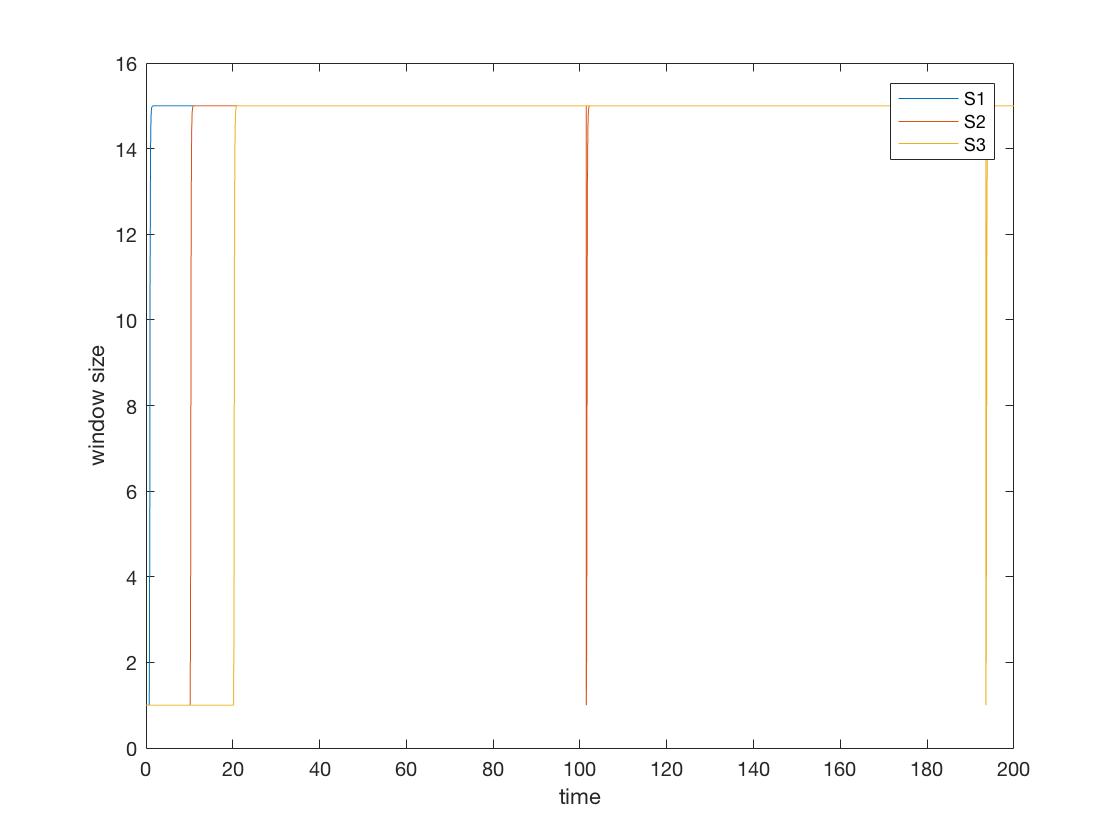
****

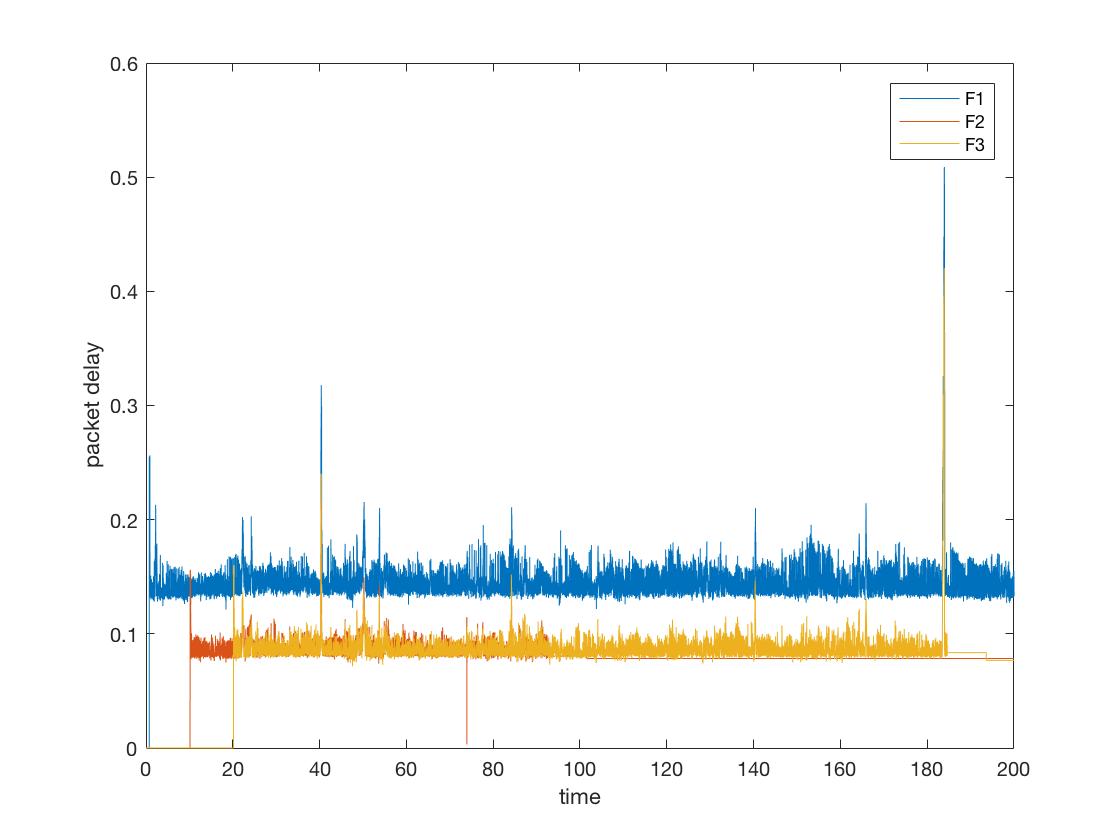
****

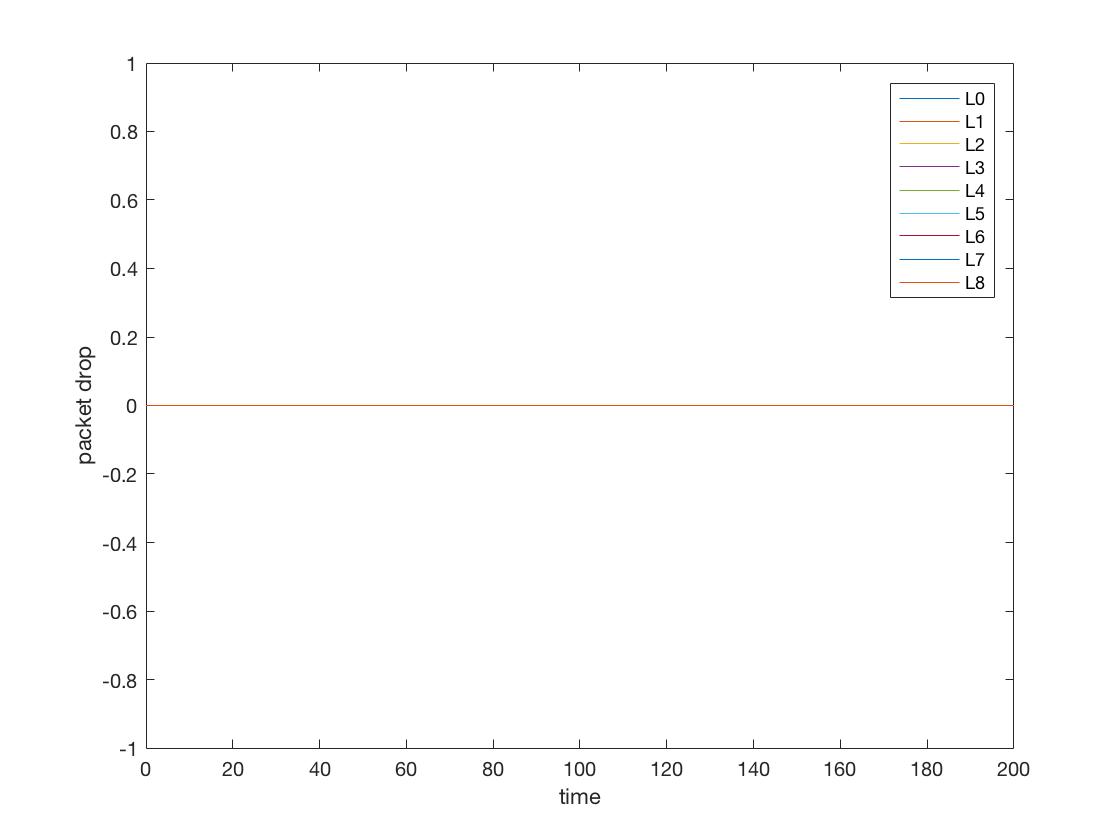
****

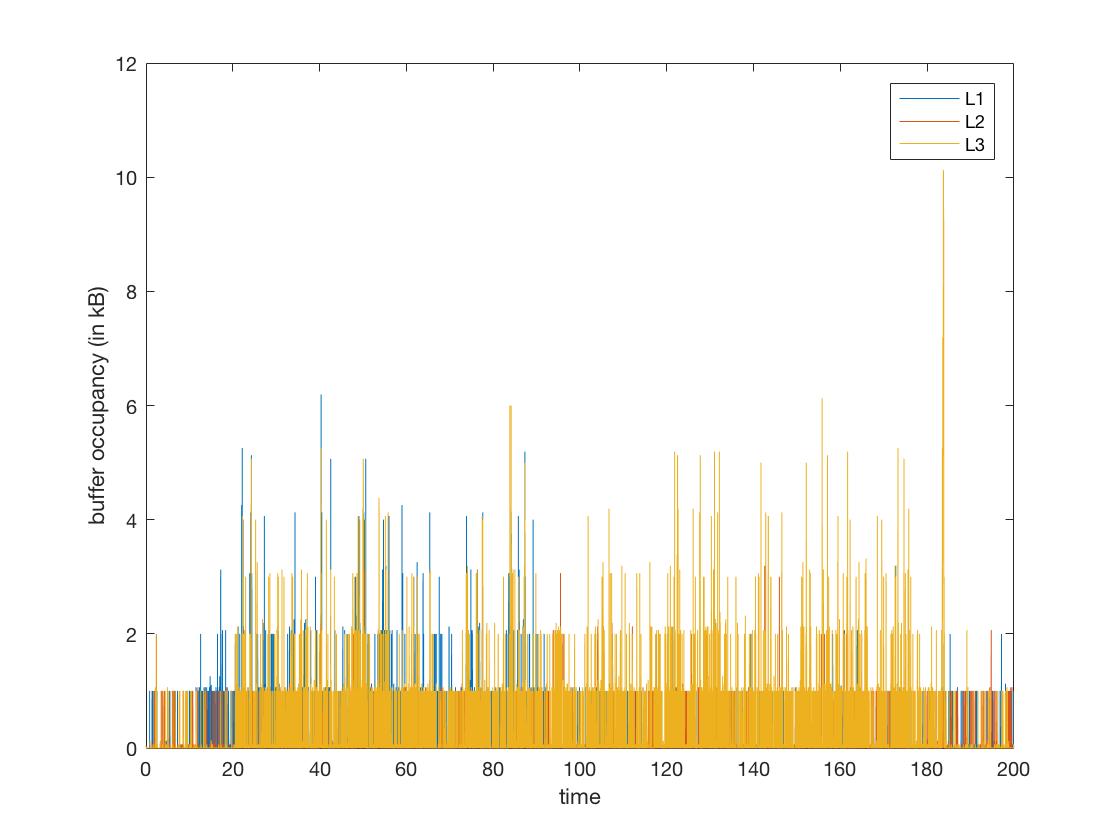
****

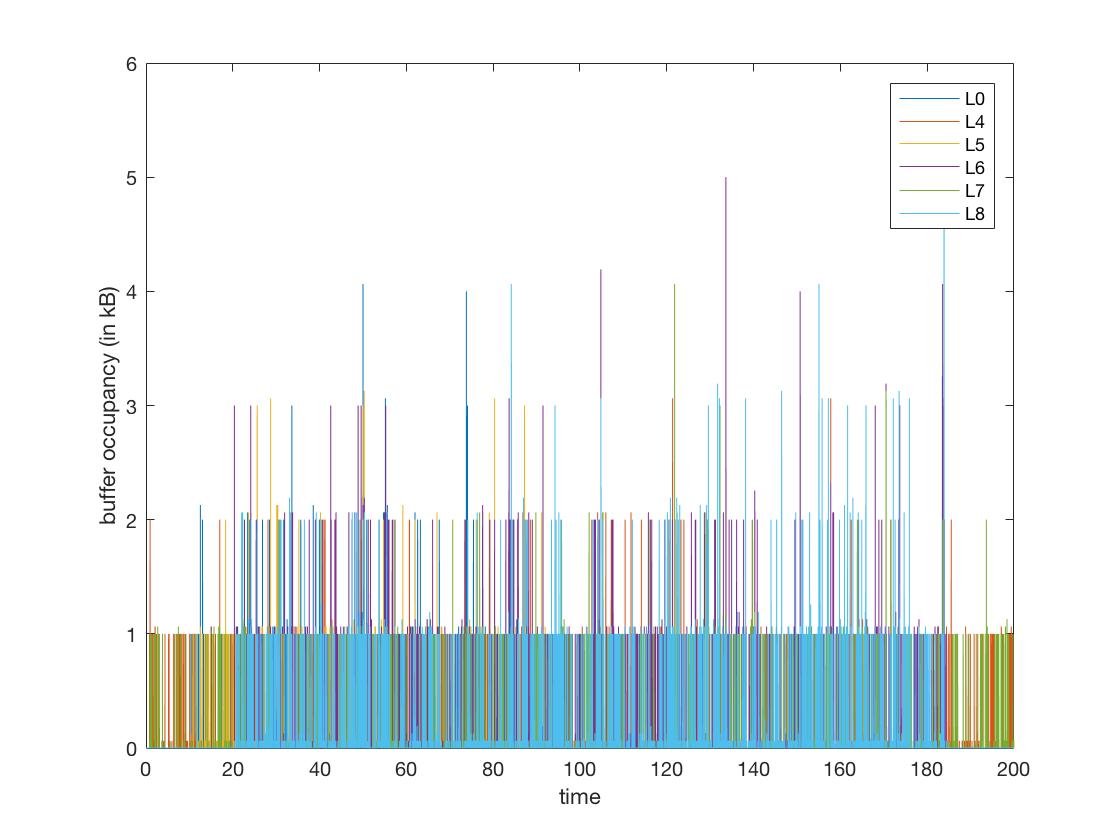
* **FAST TCP**

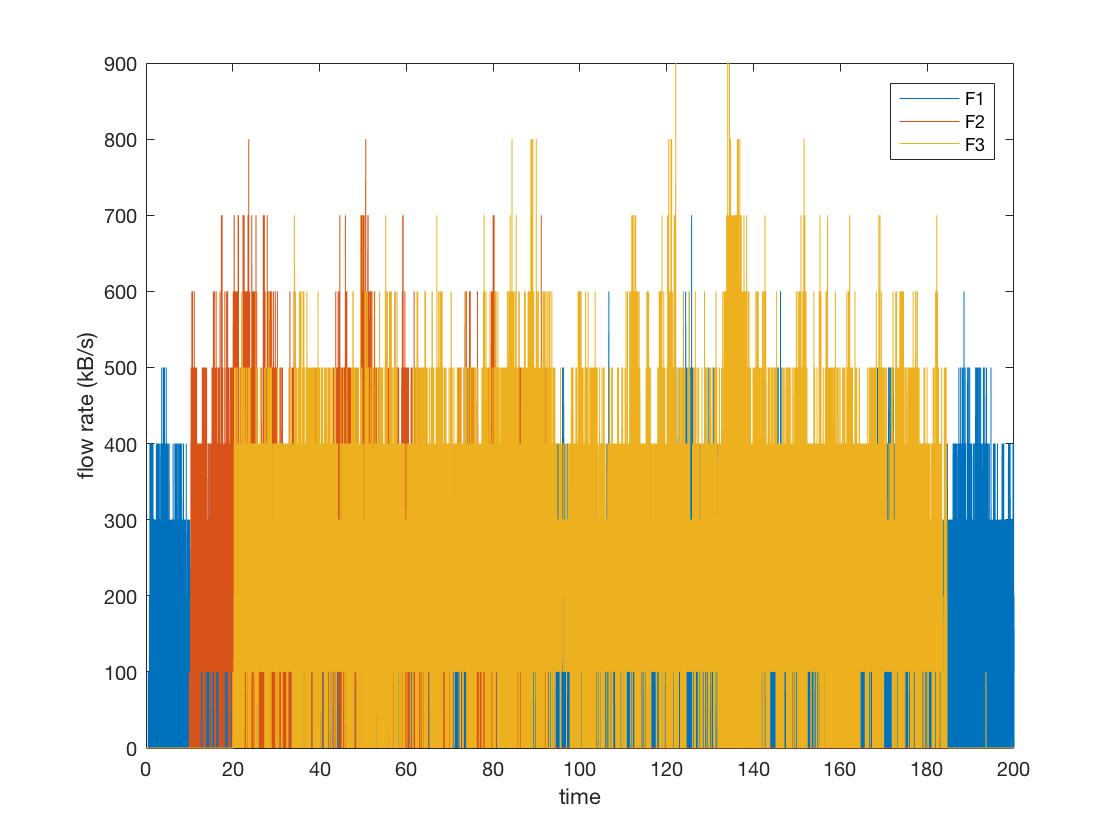
****

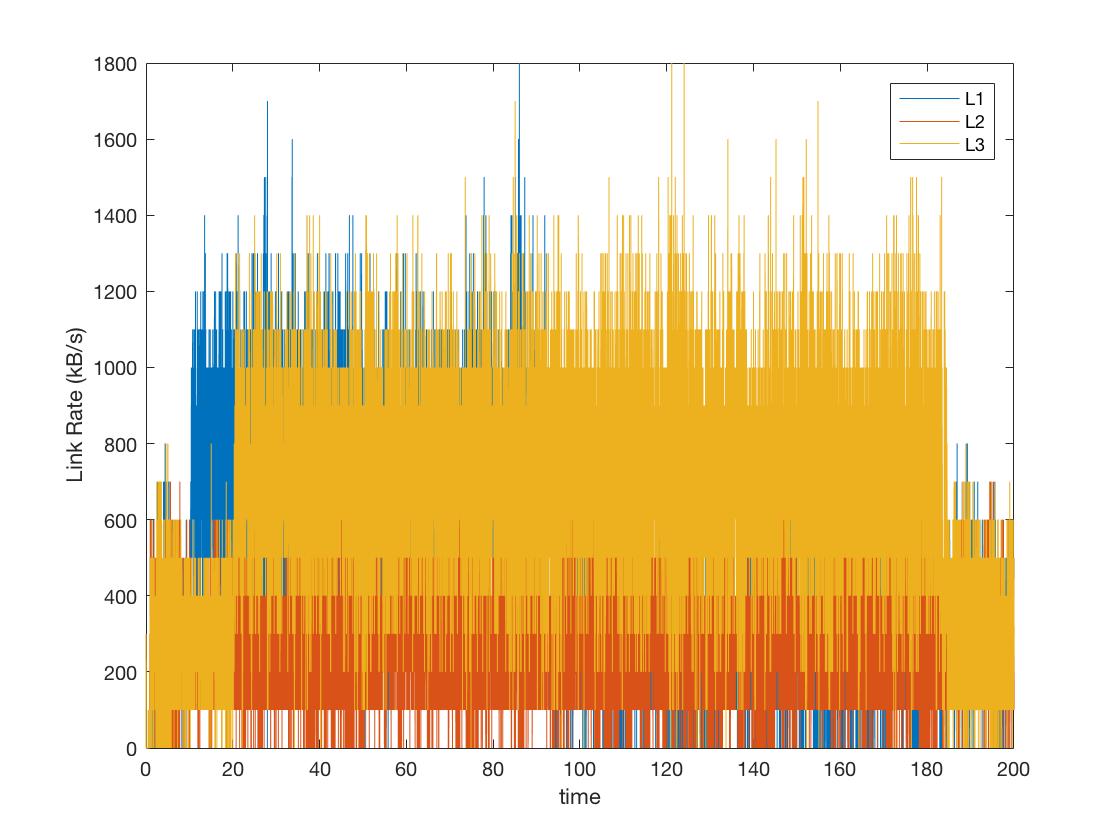
****

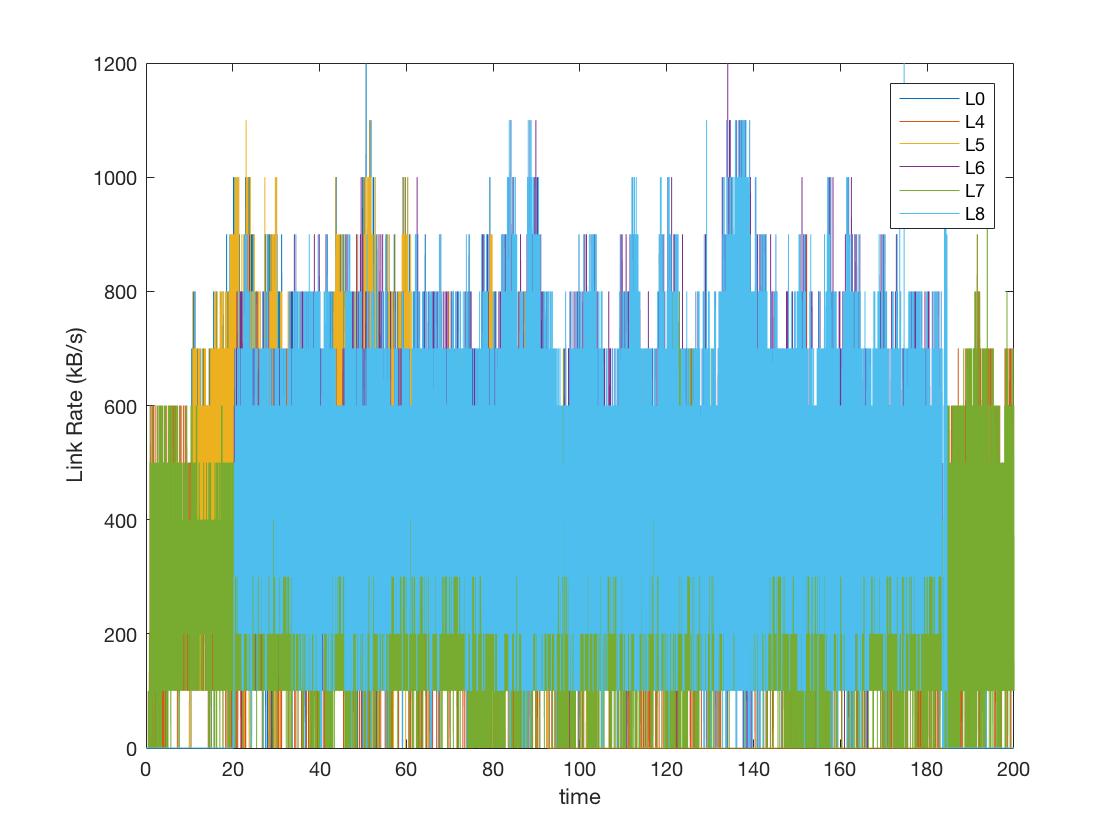
****

****

****

****

****

****

**6 Comment**

**Test Case 0**

**Test Case 1**

**Test Case 2**

**7 Conclusion**

Since the simualtion and the theoretical result gave us similar or at least the meaning of how the network is working in real world, there are still a lot of way to improve and create a better network, both in performance and reliability aspects. In this simulation, FAST TCP has proved that in a certain cases, it might work much better than the most popular and well-known TCP, TCP Reno. There are still many congestion control and routing update algorithm need to be tested and improve. This is the very first step to understand the functionarity of congestion control and routing update algorithm. With the help of simulation basis, we can obtain the result without trying in the actual network with host, and routers, and links. This project enable us to work as a team and test out many cases, using different method in each parts. Using the same simulation for host, link and router, we can plug in different type of TCP, or even UDP.

**8 Reference**

[1] CS/EE 143: Communication Networks Website, <http://courses.cms.caltech.edu/cs143/>

[2] Jean Walrand and Shyam Parekh, [Communication Networks: A Concise Introduction](http://www.morganclaypool.com/doi/abs/10.2200/S00254ED1V01Y201002CNT004?journalCode=cnt), Claypool, 2010

[3] Kurose and Ross, Computer Networking, 5th Edition, 2009

[4] Bersekas and Gallager, Data Networks, 2nd Edition, 1997

[5] Steven Low, Analytical Methods for Network Congestion Control, 2016