TCP-Tahoe Network Simulator

Mark Mchedlishvilli, Kirk Cameron, Darwin Huang, Eric Mendoza-Conner, Sehr Ahmed
The Cooper Union for the Advancement of Science and Art

Abstract— TCP, or the transmission control protocol, is a protocol that provides host-to-host connectivity at the Transport Layer of the Internet. This paper outlines and analyzes a system designed to simulate a communication network at the TCP layer. The system takes in a description of an abstract network and simulates its operation using TCP Tahoe and Distance Vector Routing as realized via the Bellman-Ford Algorithm.

I. Introduction

Real-world networks experience various forms of loss and noise that introduce error into packets on various layers of the internet architecture. TCP is a protocol that was designed to provide guarantees against error while maximizing throughput on the transport layer, through use of acknowledgements (ACKs) and congestion control. This protocol was chosen due to its simplicity and efficacy, as well as its universal use in existing systems.

TCP Tahoe is a variant of traditional TCP that functions by sending packets from hosts to hosts through "flows" across a network and waiting for acknowledgements that indicate the next expected packet. It exponentially increases the number of packets it sends until a certain threshold is reached, at which point it linearly increases the number of packets it sends until a packet is lost and detected via a timeout. At this point, the threshold is halved and the congestion window size, or the amount of packets it can send without receiving an ACK, is set to one and exponentially increases again. It was decided that TCP Tahoe would be used as the basis for how the flow class works in our TCP simulator and ultimately as a metric to determine how well the simulation worked, due to its simplicity relative to several other variants of TCP, such as TCP CUBIC and Vegas.

II. Network Topology

The network is built to send flows from hosts to hosts along the networks. It is built upon hosts, routers, and the links that connect them in the network. Hosts and routers function as 'vertices', or 'nodes', on a graph while links function as 'edges'. The hosts send and receive data, in the form of packets, that are created in flows. They also send and receive acknowledgements when they receive data. Routers route the data that's sent to them via links and use a routing table to determine which link to forward the data onto. Links hold buffers which correspond to data that has yet to be transmitted or that is being transmitted across the link. Once data has been transferred, routers route them to another link buffer. Flows are responsible for the overhead of having the packets pass through to and from their destination until all packets in said flow have completed sending. As stated previously, TCP Tahoe was implemented in the flow class, so flows adjust how many packets are sent from one host to another before waiting for an acknowledgement by that protocol definition.

Network topology, in the form of distances, is estimated and kept track of in routers' routing tables. After routers pass through the handshaking, discovery, and clear-to-send phases, they contain distance vectors to each other router based on the links' throughput, propagation delay, and available router space. Routing decisions are made through use of these distance vectors via the Bellman-Ford algorithm.

III. Design Decisions

The system was designed to run a simulation of a given network and record data derived from simulation variables at regular intervals. The system then required some sort of output functionality that would allow us to graphically show the progression of the specified variables over time.

In regards to the actual backend used to create the simulation, the programming language used was C++. C++, an object-oriented programming language that builds upon the C programming language, was chosen for its relative efficiency and object-oriented behavior, as well as the level of familiarity of the language among our team members.

When designing the simulator, a decision was made such that the event queue could operate in discrete time, as such an approach would be more computationally efficient without a loss of definition as compared to a real-time event manager. As a result, implementation would be more resilient against faults, and simulations could be completed in a much shorter time-frame.

All components of the network (links, flows, hosts, routers) would be implemented by creating classes that would be called as objects in the user interface, taking advantage of the use of an object oriented language. With regards to the user interface, a specific syntax for text files was chosen such that the text file could be parsed after the user entered a filename once prompted. Since we decided that a console window would be the most efficient interface to run the simulation, and opted not to use GUI or graphing frameworks available to C++, it was found that reading in a network description file would be easier to test on than manually inputting the data. Due to the lack of graphing and data visualization suites available in the C++ language, as well as the increased difficulty of designing graphical interfaces in C++, the loaded network would then be simulated and graphs could be output into .csv type files, which could then be exported to Microsoft Excel for post-processing and graphing. For the purpose of simulating packet flows, the flow class would be called upon to create data and send it over the network from one host to another. The data that was measured included flow rate, congestion window (cwnd) size, and various other parameters that help manage TCP communication and output for graphing functionality.

IV. Validation Methodology

Three test cases were simulated using our program, beginning with the network discovery phase performed by routers, and proceeding with sending data packets until the flow is completed. The transmission of each packet, along with relevant data such as start/end times of transmission, were stored in a text file. Processing this file allowed for the production of the link and flow rates. In addition, flow information was stored to produce data for congestion window, packet loss graphs, etc.

The following graphs show flow rate and congestion window size given the following parameters for the network:

Link Specifications

LinkID	Link Rate (Mbps)	Link Delay (ms)	Link Buffer (KB)
L1	10	10	64

Flow Specifications

Flow ID	Flow Source	Flow Dest	Data Amt (MB)	Flow Start (s)
F1	H1	H2	20	1.0

The test cases' data (seen in figures below) indicate trends for both congestion window size and flow rates over time, showing packet growth according to TCP Tahoe, along with the expected times of packet loss. Other test cases show behavior similar to that exhibited in the given simulation results, though not identical. These results correspond to our expectations of TCP Tahoe.

V. Charts- Test Case 0

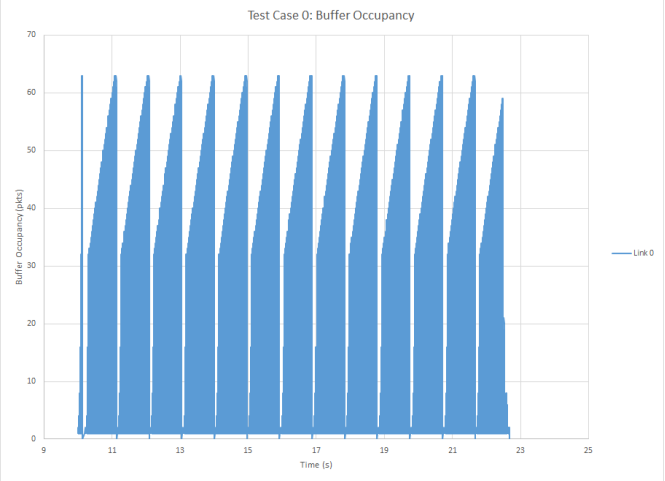

Figure 1: Time vs. buffer occupancy graph for Test Case 0. Buffer occupancy increases linearly as packets are sent through the link.

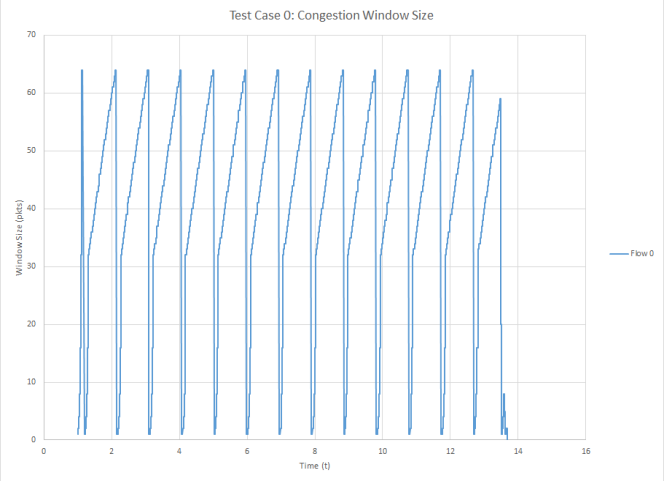

Figure 2: Time vs. congestion window size for test case 0. CWND grows exponentially, then linearly.

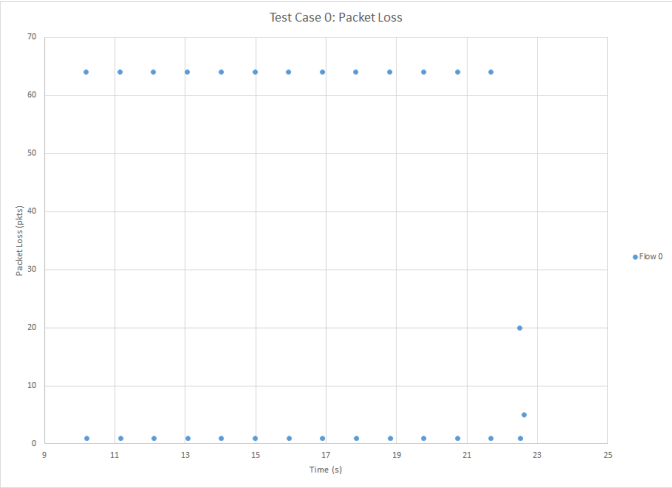

Figure 3: Time vs. packet loss for test case 0. Packet loss corresponds to link buffer overflow points, and congestion window resets.

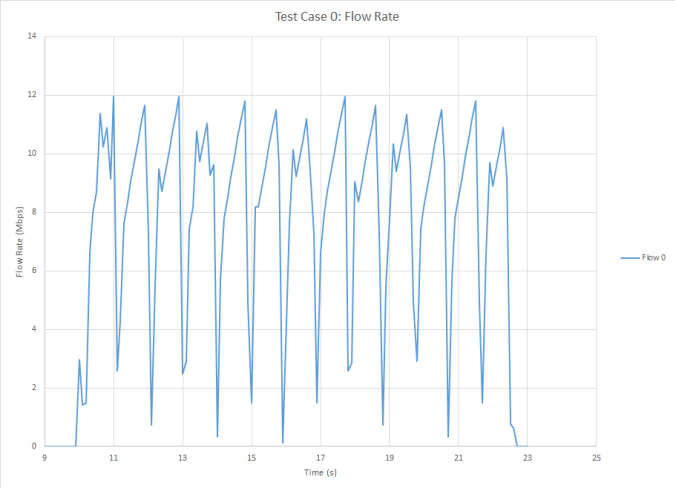

Figure 4: Time vs. flow rate for test case 0. Flow rate corresponds to congestion window changes.

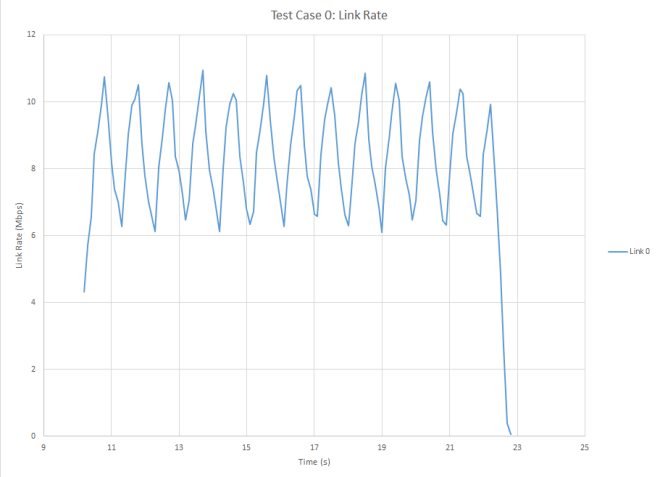

Figure 5: Time vs. packet delay for test case 0. Packet delay spikes when packets are lost and congestion window resets

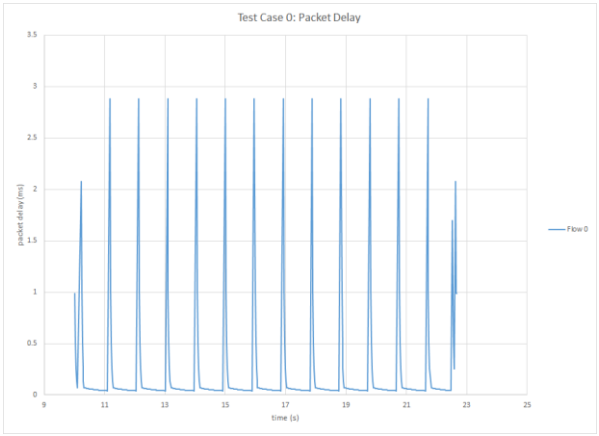

Figure 6: Time vs. packet delay for test case 0. Packet delay spikes when packets are lost and congestion window resets.

VI. Charts- Test Case 1

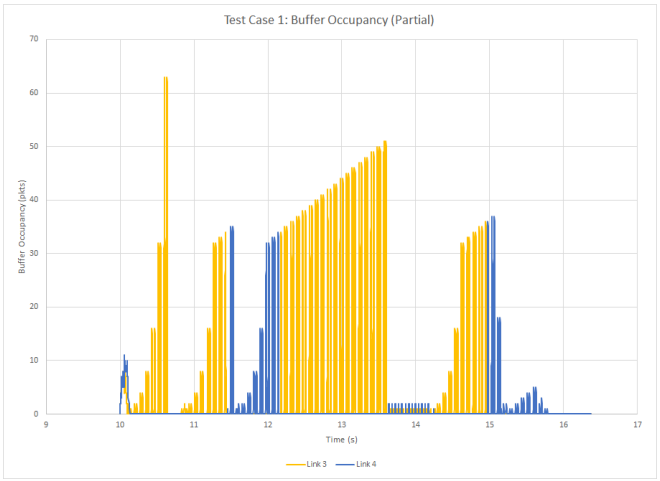

Figure 7: Time vs. buffer occupancy for test case 1. Shows the changes made in the routing table corresponding to an improved path, which alternates between Link 3 and Link 4.

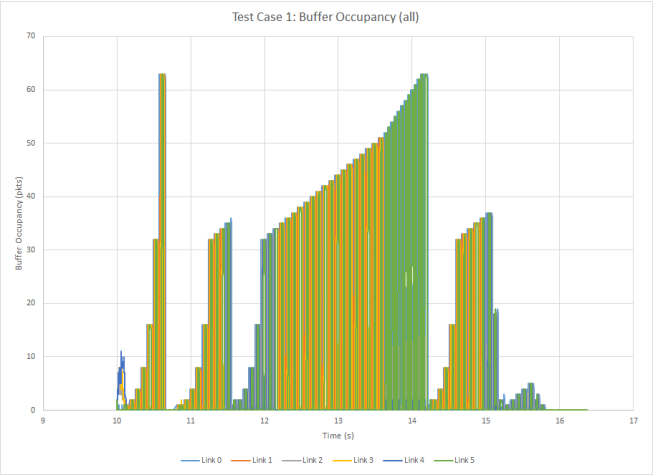

Figure 8: Time vs. buffer occupancy for test case 1. Complete graph of buffer occupancy of all link rates, matching the flow rate.

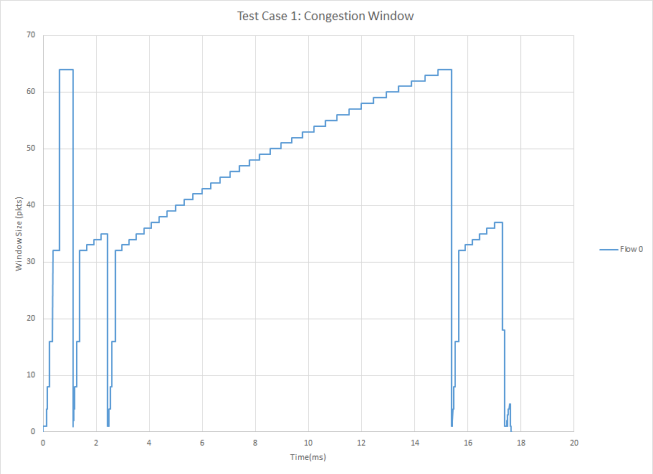

Figure 9: Time vs. window size for test case 1.

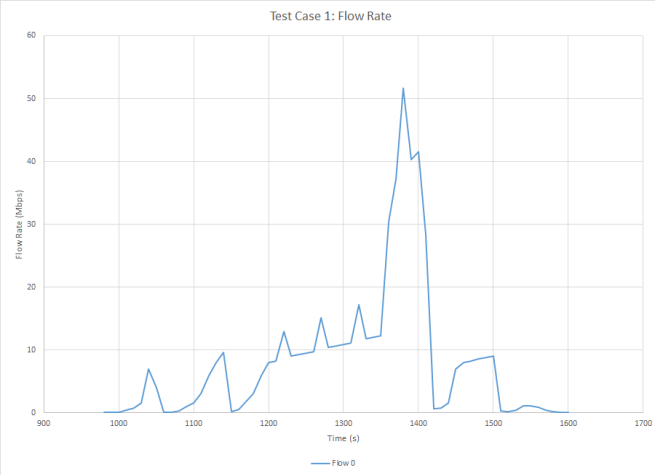

Figure 10: Time vs. flow rate for test case 1. A spike in flow rate is shown at the point where both link paths are utilized before falling back down to a single path.

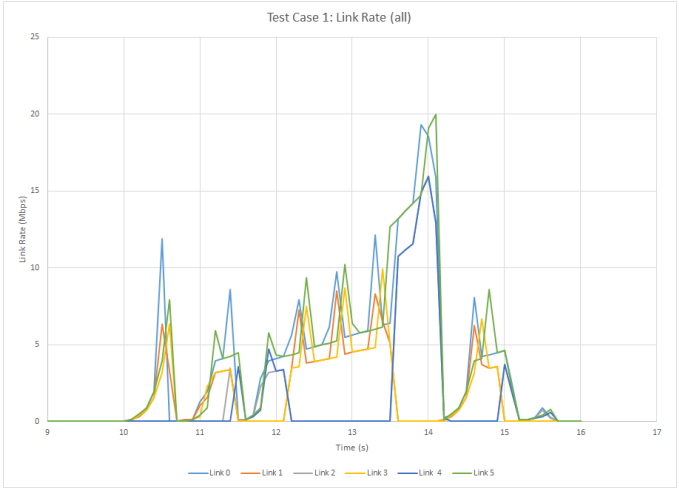

Figure 11: Time vs. all link rates for test case 1.

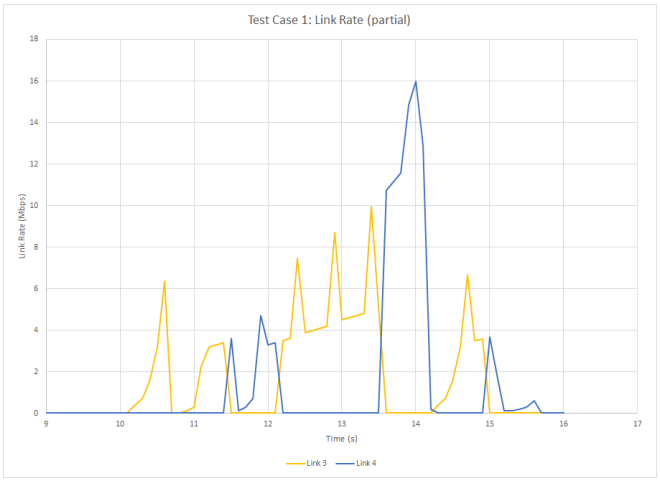

Figure 12: Time vs. link rates for Link 3 and Link 4 for test case 1. Shows how the routing changes dynamically between the two link paths.

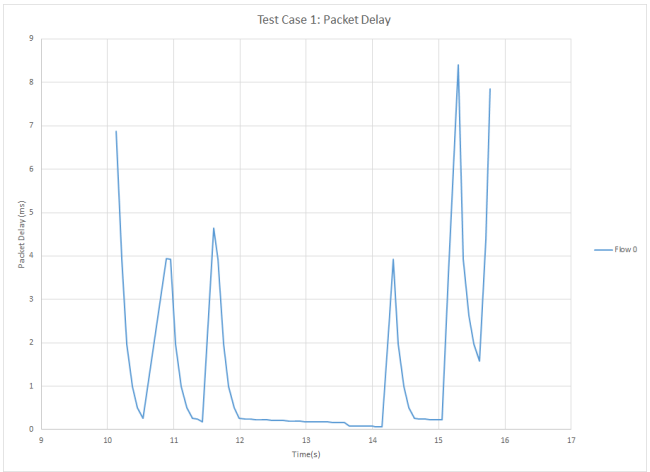

Figure 13: Time vs. Packet delay for test case 1.

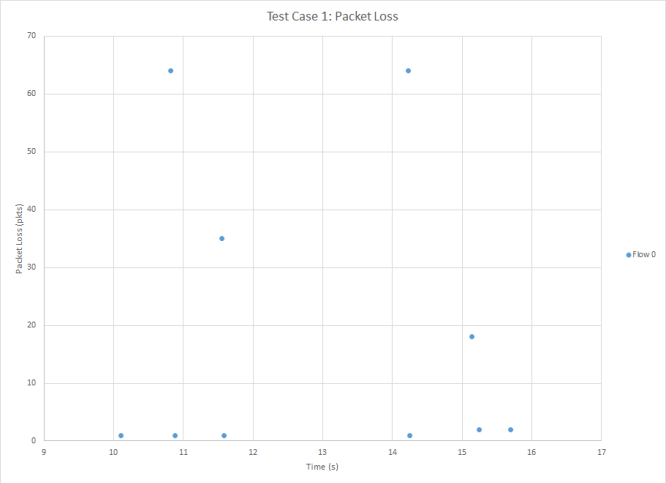

Figure 14: Time vs. packet loss for test case 1.

I. Charts- Test Case 2

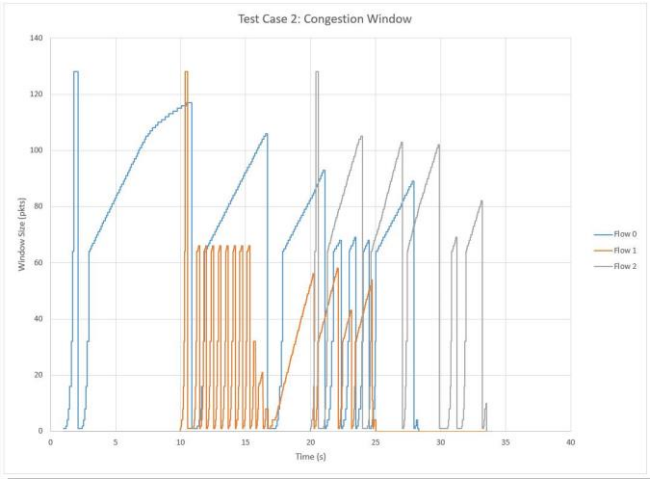

Figure 15: Time vs. congestion window for test case 2. Shows how the congestion windows of each flow affect each other.

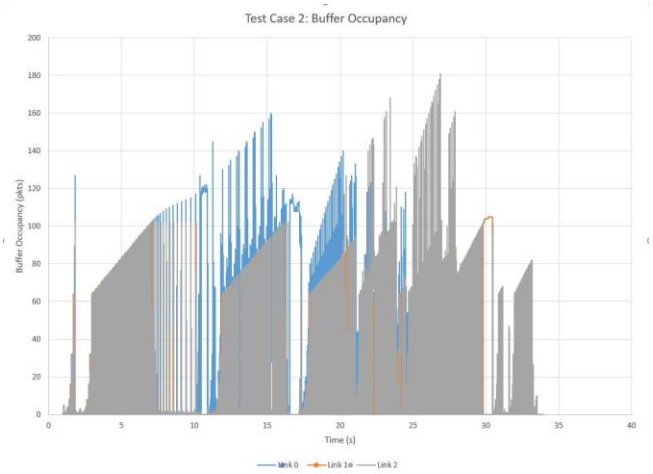

Figure 16: Time vs. buffer occupancy for test case 2.

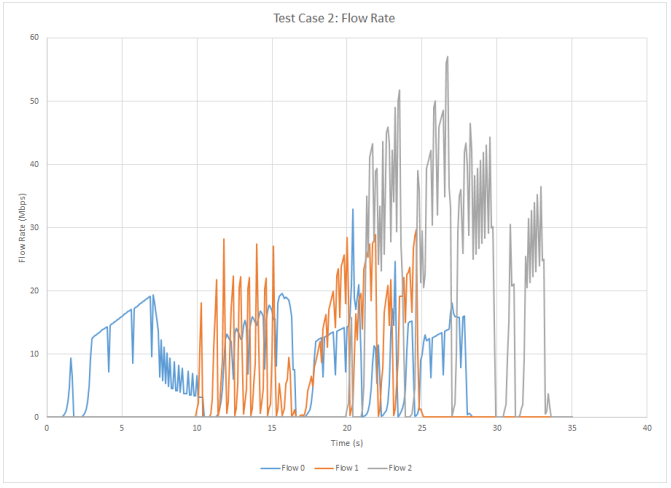

Figure 17: Time vs. flow rate for test case 2. Shows how flows affect each other when transmitting simultaneously.

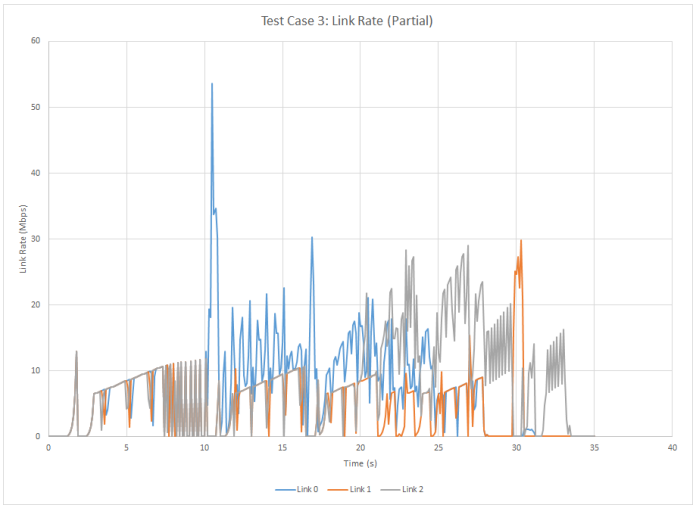

Figure 18: Time vs. link rate for links 0,1,2 for test case 2.

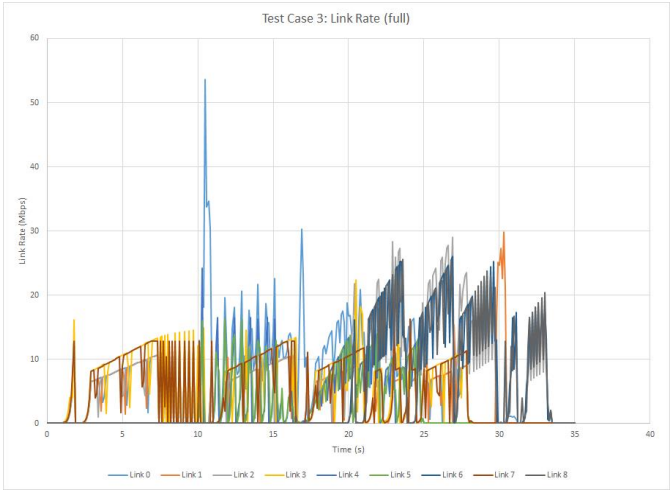

Figure 19: Time vs. link rate for all links for test case 2.

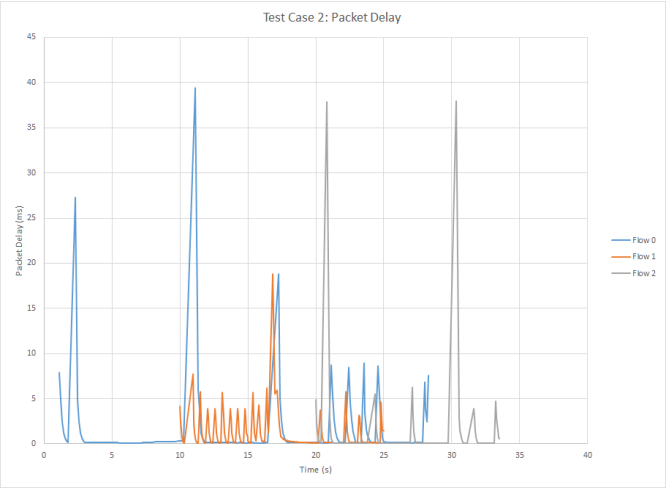

Figure 20: Time vs. packet delay for test case 2.

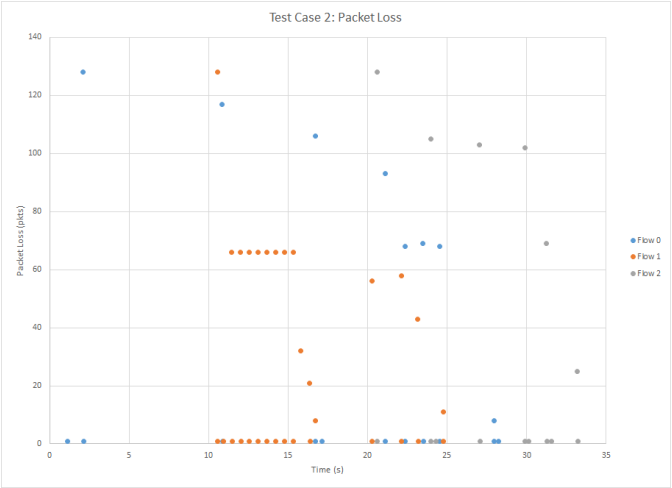

Figure 21: Time vs. packet loss for test case 2.

II. CONCLUSION

This simulator confirmed everything understood about how TCP works, and effectively demonstrated the benefits and consequences associated with this version of TCP. As can be seen from the graphs, TCP regulated data flow using congestion window sizes, and packet timeout events. Through the development of this simulator, we were able to take an abstract network and simulate its operation. Ultimately, this allowed us to enhance our understanding of networks and their components and provided us with a functional tool used to validate theoretical analyses of networks.