# Quantitative Analysis to Verify Fairness of TCP CUBIC in NS-2

Gary Griswold, Marcel Martinez Carrato and Rahul Gomes
Dept of Computer Science
Minot State University
Minot, ND 58704
Gary.Griswold@ndus.edu

## Abstract

Transmission Control Program (TCP) operates on the transport layer and provides host-to-host connectivity. TCP is responsible for packet retransmission. Network Simulator 2 (NS-2) provides access to many TCP variants including TCP-Reno, CUBIC and YeAH.

CUBIC [3] utilizes a cubic function instead of a linear function improving stability, and has two phases, convex and concave. Currently, CUBIC is deployed on Linux systems. This paper investigates how CUBIC interacts in a network with other TCP variants, Reno and YeAH. CUBIC is designed to be optimal for longer distance networks, while, at the same time being TCP-friendly.

Experiments were conducted in NS-2 on Ubuntu 14.07. To test how CUBIC shares network bandwidth, the link delay was altered to simulate a long/short distance network. In all scenarios, CUBIC took most of the bandwidth compared to standard TCP algorithms over the bottleneck link. Further research is required to mitigate this greedy nature of CUBIC.

# 1. Introduction

In 1974, the Transmission Control Program (TCP) was created for the ever-increasing network traffic of the internet. TCP operates on the transport layer and provides host-to-host connectivity. When a packet is sent from one host to another, there is a possibility a packet can be dropped. TCP detects this and requests the sender to retransmit lost data.

TCP controls how data is sent by keeping track of a CWND [1]. The CWND limits how much data can be sent, and continues to increase after each successful ACK. After a packet is determined to be dropped the CWND is decreased. This process prevents more data being sent than the network can handle.

Since 1974, there have been many flavors of TCP created, all with different variants to how they handle congestion. NS-2 provides access to many of these TCP variants, to include TCP-Reno, TCP-Cubic and YeAH-TCP. By utilizing NS-2, a comparison of how these three TCP flavors interact with each other, and the effect of network distance is evaluated, along with fairness between variants.

# 2. TCP Variants Investigated

TCP-Reno utilizes Additive Increase Multiplicative Decrease (AIMD) in its congestion control algorithm. Reno begins with a slow start phase, which exponentially increases CWND, until it reaches a predetermined threshold. Once the threshold is reached, it enters another state called congestion avoidance. In the congestion avoidance stage, CWND is increased by one after each successful ACK, thus creating linear growth. When Reno detects a packet loss, determined by triple duplicate ACKs, the CWND is reduced in half, and a new threshold is set at this point. Then Reno goes into congestion avoidance and does not re-enter slow start until and unless the connection is terminated [1].

YeAH-TCP uses a mixed loss/delay approach to compute the Operates in two modes of Slow and Fast. In Fast mode, increases CWND aggressively, in Slow mode it is like Reno. This design is created to be very efficient and fair to Reno [2].

TCP-CUBIC differs from other TCP connections because it uses a cubic function as opposed to a linear function to improve stability in long distance networks. “CUBIC is Designed with four main principles. Principle 1: For better network utilization and stability, the concave and convex profiles of a cubic function are used to increase CWND. Principle 2: To be TCP-friendly, CUBIC is designed to behave like Standard TCP in networks with

short round trip times (RTT). Principle 3: For RTT-Fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs. Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed[3]." CUBIC has two phases to CWND growth, the convex and concave portions. In the concave portion CWND is increased greatly. Afterwards there is a plateau to allow the network to stabilize, followed by the convex growth phase where CWND begins to search for more bandwidth to increase CWND.

# 3: CUBIC Algorithm

Since TCP-CUBIC uses a cubic function as opposed to a linear function for growth and stability, it does not follow the AIMD algorithm for increasing window size. CUBIC utilizes the window function [3]:

$$W\_cubic(t) = C*(t-K)\ ^3 + W\_max \quad \text{(Eq. 1)}$$

where C is a constant, t is the elapsed time from the start of the congestion avoidance, and K is the time period that the above function takes to increase the current window size to W_max if there are no further congestion events and is calculated using the following equation:

$$K = cubic\_root(W\_max*(1-beta\_cubic)/C) \quad \text{(Eq. 2)}$$

where beta_cubic is the CUBIC multiplication decrease factor, that is, when a congestion event is detected, CUBIC reduces its cwnd to W_cubic(0)=W_max*beta_cubic [3]. When a packet loss, or network congestion is detected, beta_cubic is set to 0.7 and CUBIC updates its W_max, CWND, and ssthresh to the following:

$$W\_max = cwnd$$
$$ssthreth = cwnd * beta\_cubic$$
$$ssthresh = max\ (ssthresh, 2)$$
$$cwnd = cwnd* beta\_cubic$$

To keep CUBIC to Principle 2, CUBIC checks the average window size of standard TCP to verify it is in a TCP-friendly region, afterwards window size is determined by:

$$W\_est(t) = W\_max*beta\_cubic + [3*(1-beta\_cubic)/(1+beta\_cubic)] * (t/RTT) \quad \text{(Eq. 3)}$$

W_est(t) ensures that CUBIC does not grow uninterrupted thereby facilitating bandwidth sharing.

# 4: Experiment

Experiments were conducted on the Linux Ubuntu 14.07 operating system using the NS-2 environment. All settings for each TCP remained the same, except for the link delay, which was altered to simulate a long/short distance network. All TCP variants were tested against themselves, as well with the others, with a final test being conducted with all three variants being tested together.

The topology for each pairing was set up with two nodes transmitting, to a link and being delivered to and end node, with the link acting as a bottleneck. The two transmitting nodes and the end node all had the same transmission delay of 10ms. When all three were tested together, a third node was added to the link, and it was sending data to the same end node.

In order to test the effects which network distance plays on these variants, the link transmission delay was tested at 200ms, 100ms, 50ms and 10ms. This was to ensure varying distances were examined for each variant.

## 4.1: Comparison of matching algorithms with varying delays

### 4.1.1: Reno vs Reno

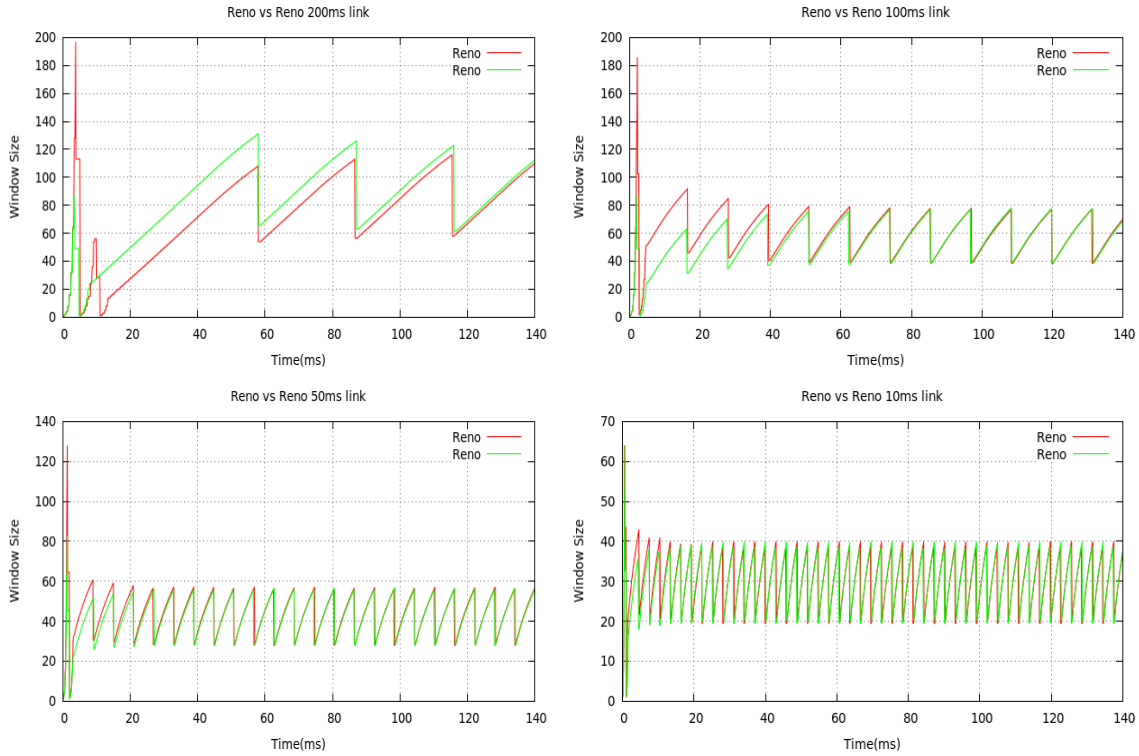


*Figure 1: Reno vs Reno*

Figure 1 (above) shows the output of Reno vs Reno with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. We see a steady rise in window size for the 200ms time interval before a packet is dropped and the window size is halved. It is observed that the system takes more time to achieve equilibrium if the delay is large (top-left). On reducing the delay, the system achieves equilibrium and both the TCP sources can share bandwidth without one overshadowing the other. For a shorter time delay we observe that the system achieves peak window size very fast before dropping a packet and restarting its additive increase phase at half the window size. Because the delay is small the system drops a packet very early and cannot reach a large enough window size like the graphs shown in 200ms delay where the window size peaked at about 120.

### 4.1.2: YeAH vs YeAH

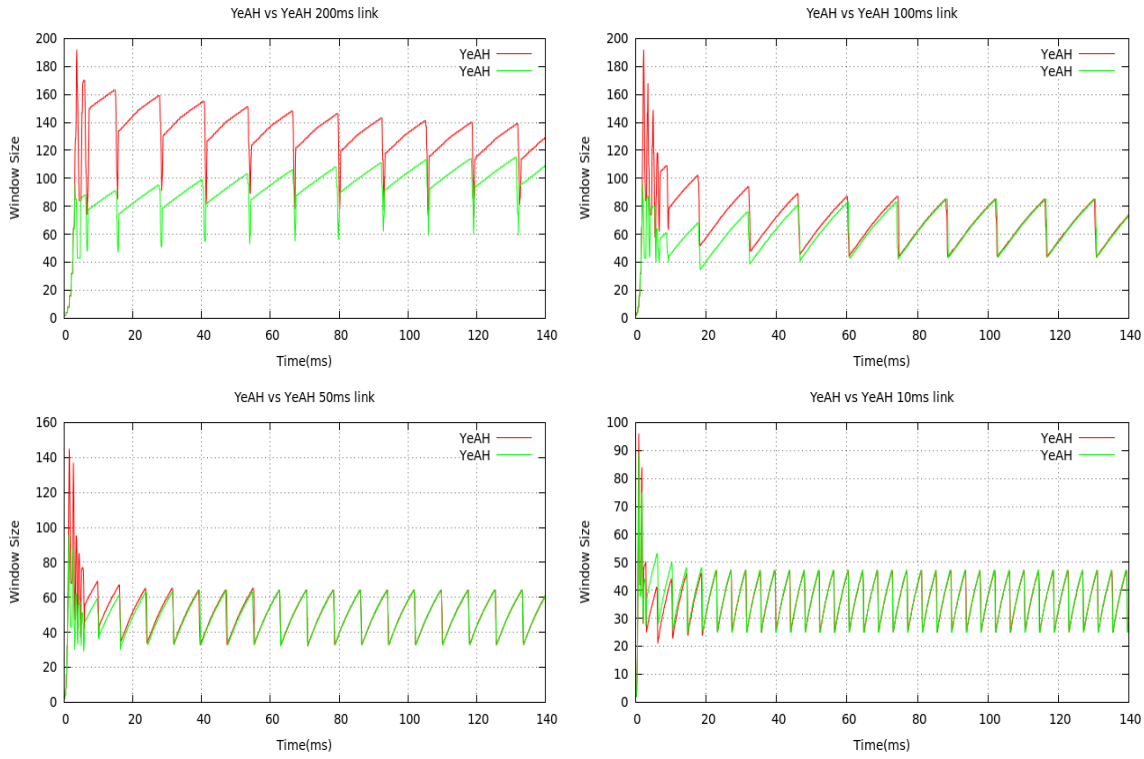


*Figure 2: YeAH vs YeAH*

Figure 2 (above) shows the output of YeAH vs YeAH with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. YeAH, like Reno, shares the bandwidth evenly with another YeAH flow. In all cases, the dominant YeAH flow continues to decrease, and the non-dominant YeAH flow continues to increase, until both converge. As network distance increased, YeAH took longer to converge, but had greater bandwidth shown in top-left. When the link was

shorter, YeAH behaved very similarly to Reno, and had a decrease in bandwidth which appears in top-right, bottom-left and bottom-right.

### 4.1.3: CUBIC vs CUBIC

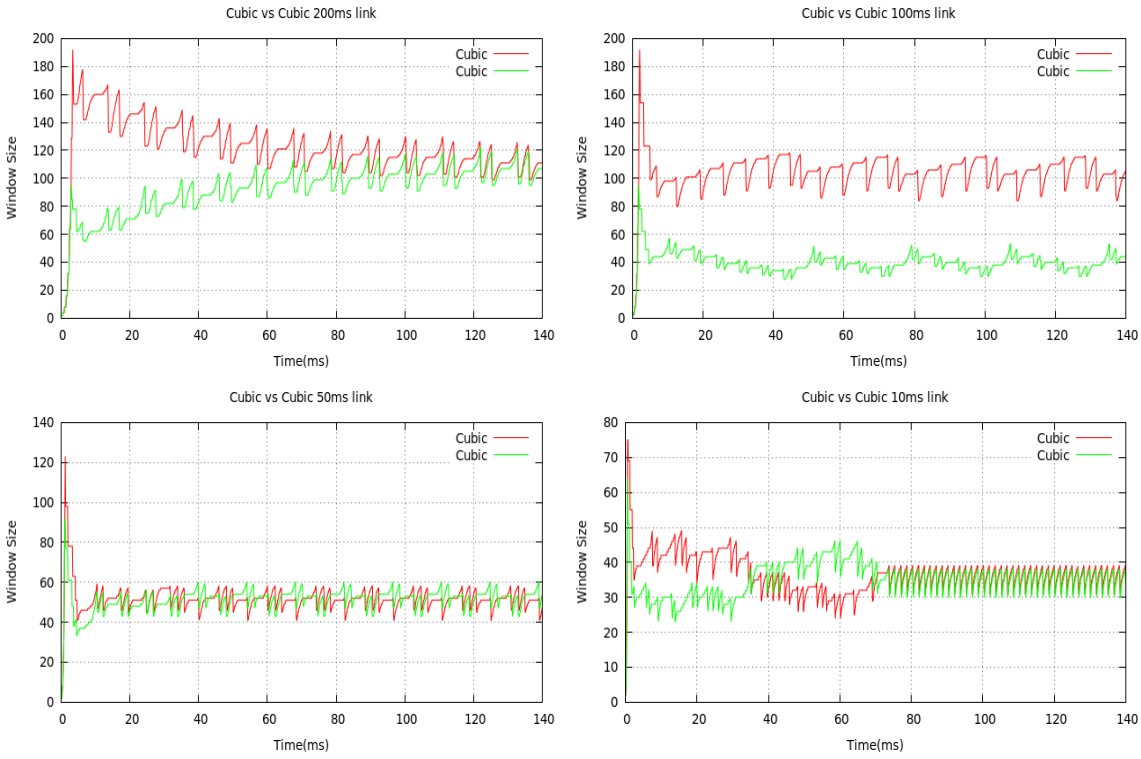


*Figure 3: CUBIC vs CUBIC*

Figure 3 (above) shows the output of CUBIC vs CUBIC with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. As network distance was high (top-left) both flows show different properties of the CUBIC algorithm. The dominant flow shows how CUBIC decreases its window size to better share the bandwidth, and the non-dominant flow show how CUBIC steadily grows, even after a packet-loss. Unlike Reno and YeAH, CUBIC did not converge in every case. When tested with the 100ms bottleneck link, one flow consistently held more bandwidth than the other (top right).

## 4.2: Comparison of differing algorithms with varying delays

### 4.2.1: CUBIC vs YeAH

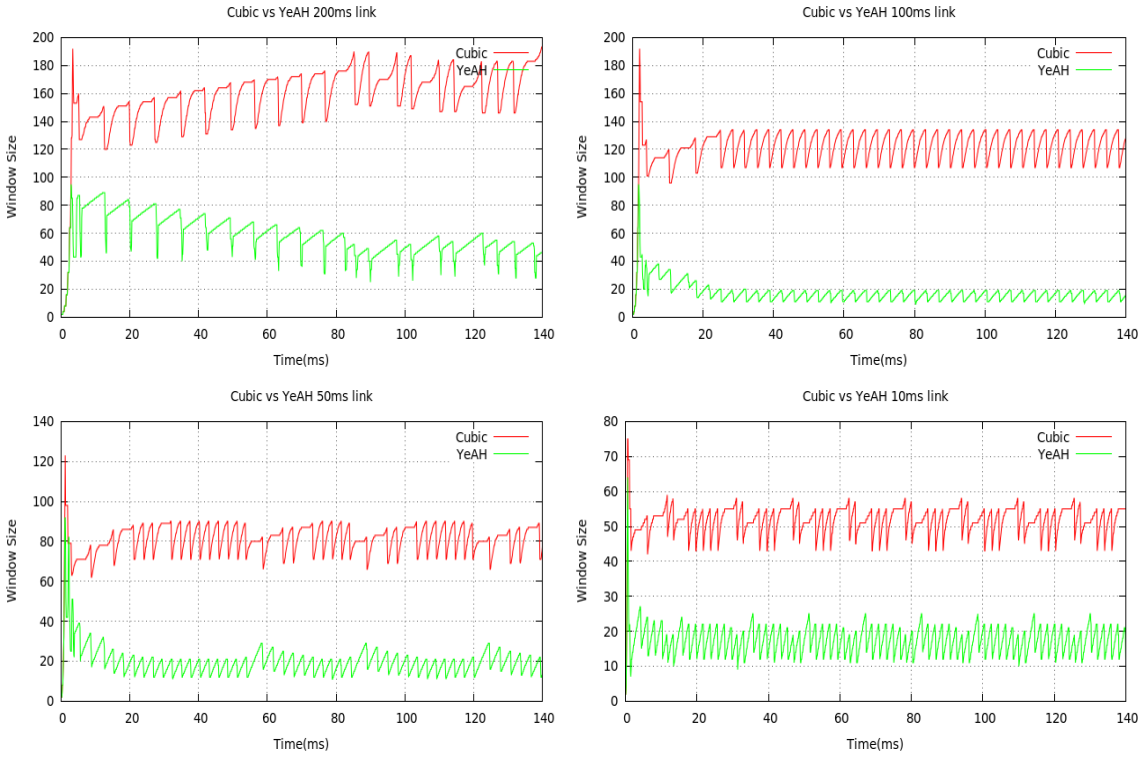


*Figure 4: CUBIC vs YeAH*

Figure 4 (above) shows the output of CUBIC vs YeAH with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. CUBIC held more of the network than YeAH in all cases. Figure 4 top-left shows YeAH having a minimum window size of approximately 40, whereas Figure 2 top-left shows YeAH with a minimum window size of approximately 100. CUBIC continued to grow only in top-left when network distance was greatest. Each test of CUBIC vs YeAH shows a significantly decreased window size compared to that of YeAH vs YeAH.

### 4.2.2: CUBIC vs Reno

Figure 5 (below) shows the output of CUBIC vs Reno with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. CUBIC again dominates the network compared to Reno. In all cases in Figure 5, Reno keeps a steady window size of approximately 20, where CUBIC's window sizes decrease as network distance decreases. However, Figure 1 shows a window size of 20 for Reno being the minimum for all network distances tested. Also, as network distance

decreases, the amount of time CUBIC spends in a TCP-friendly region increases. This is evident in the areas where CUBIC does not show both convex and concave regions. It is most evident in Figure 5 bottom-right, from 50ms onward.

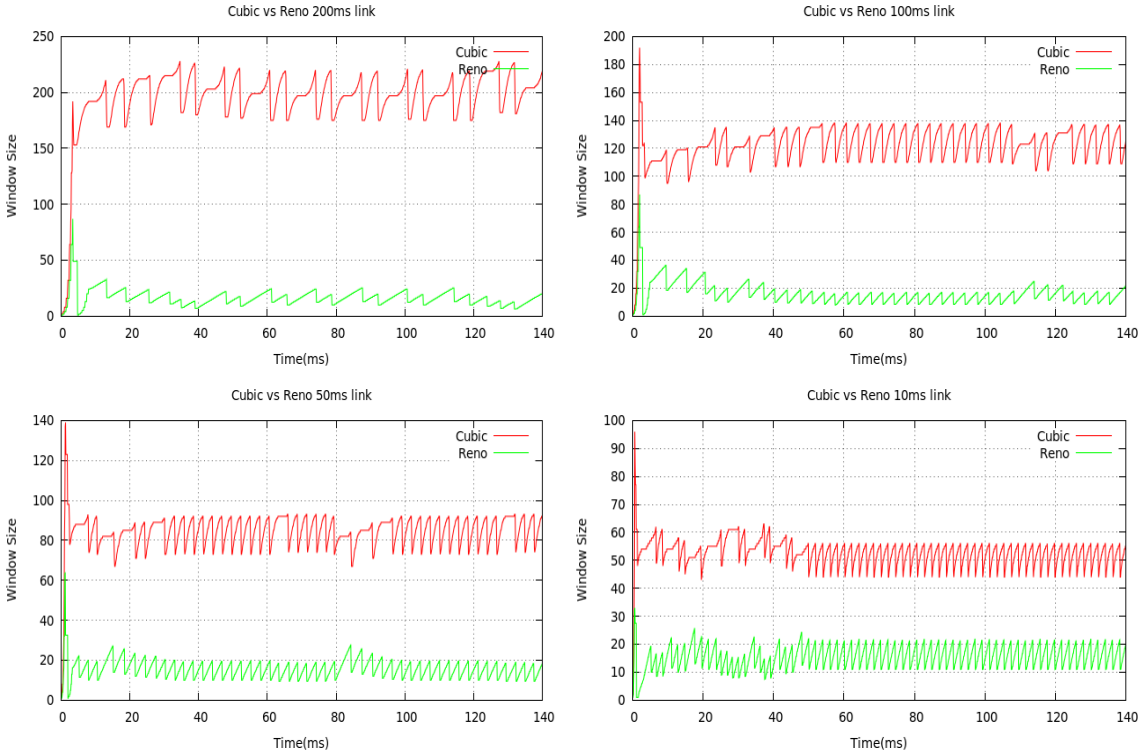


*Figure 5: CUBIC vs Reno*

### 4.2.3: Reno vs YeAH

Figure 6 (below) shows the output of YeAH vs Reno with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. For the long-distance network (Table 6 top-left), YeAH controls the bandwidth. However, when the link delay decreases to 100ms (Table 6 top-right), YeAH and Reno begin to converge. For the shorter distance networks, YeAH and Reno converge very quickly. Figure 6 top-right, bottom-left and bottom-right, all show the characteristic of YeAH which is to be fair to Reno. Figure 1 top-left, bottom-left and bottom-right, Reno has the same window size values as YeAH vs Reno.

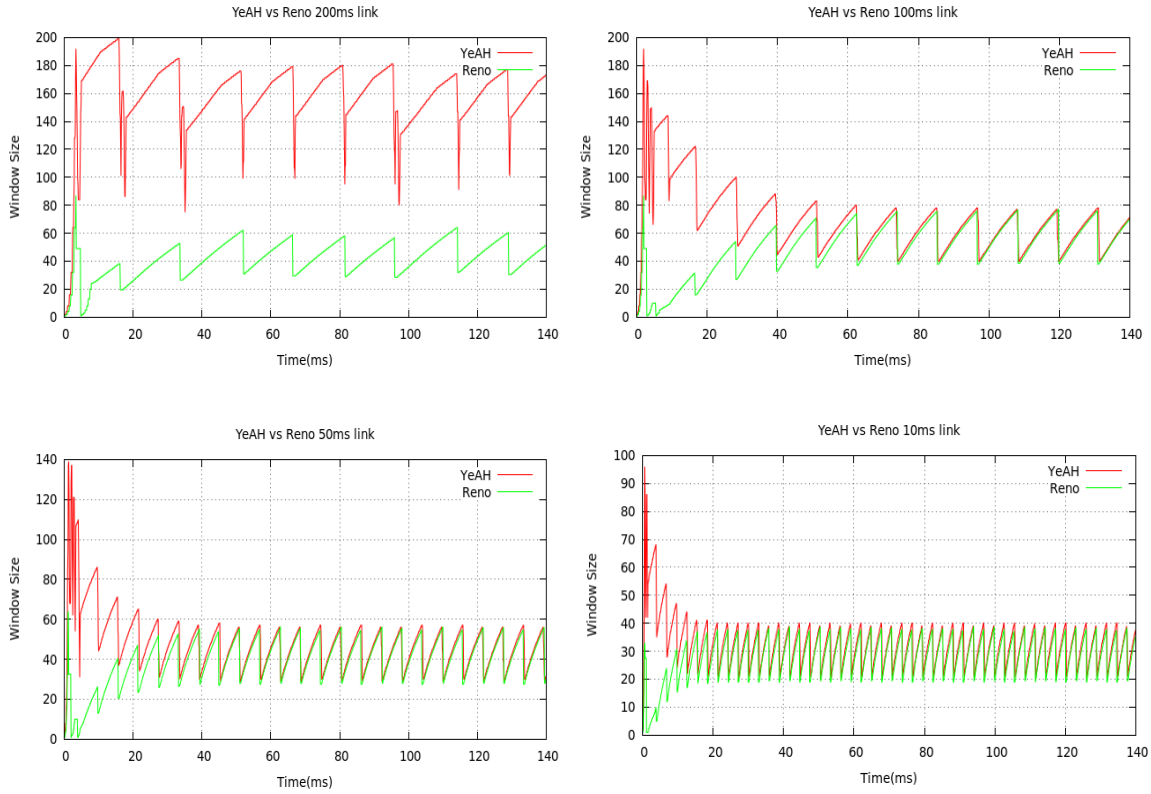


*Figure 6: Reno vs YeAH*

## 4.3: Comparison of all three algorithms with varying delays

### 4.3.1: CUBIC vs YeAH vs Reno

Figure 7 (below) shows the output of CUBIC vs YeAH vs Reno with varying time intervals. The delay for top-left is 200ms, top-right is 100ms, bottom-left is 50ms and bottom-right is 10ms. CUBIC is shown to take a majority of the bandwidth for each of the cases. While CUBIC increases, YeAH decreases and Reno remains relatively steady. When the link delay is decreased to 50ms (Figure 7 bottom-left), YeAH reverts to slow phase and converges with Reno, while CUBIC continues to grow until it hits its TCP friendly region. These TCP friendly regions occur in the 50ms and 10ms links (bottom-left / bottom-right).

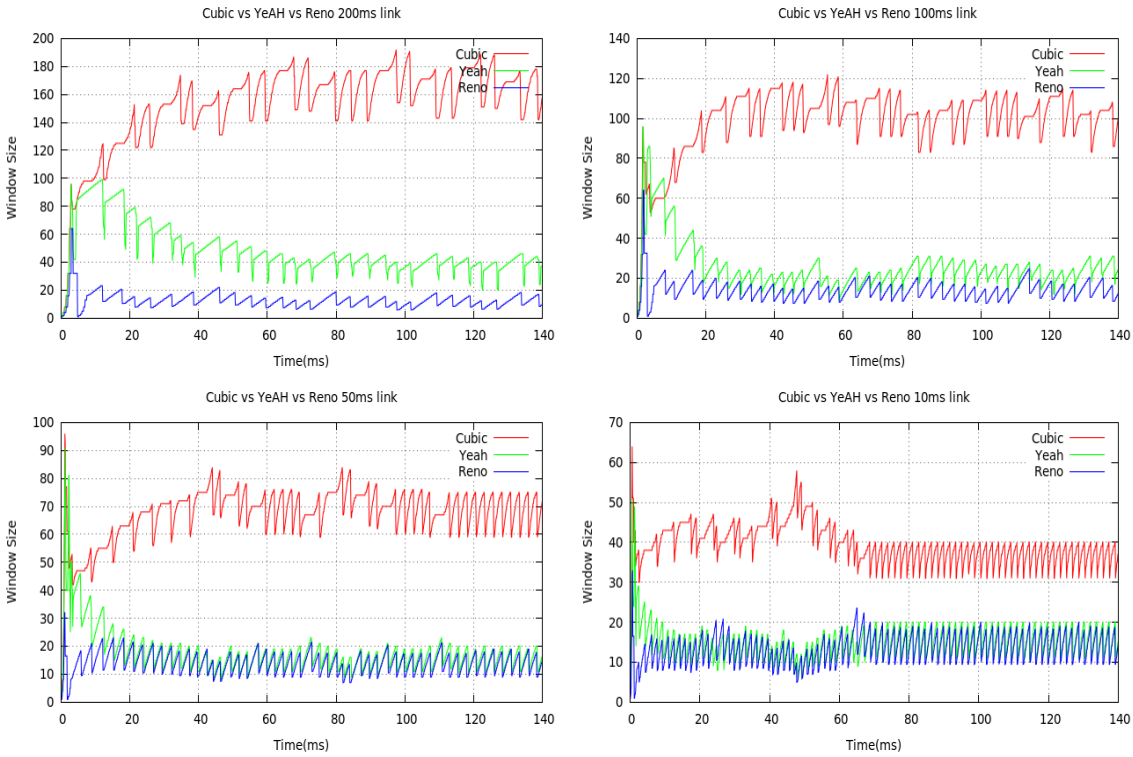


*Figure 7: CUBIC vs YeAH vs Reno*

# 6: Conclusion and Future Work

Throughout all scenarios, Reno remained at a steady bandwidth. Meanwhile, YeAH TCP, consumed more bandwidth than Reno, unless paired with Reno in a shorter distance network. In which case, YeAH converged with Reno. CUBIC in all cases dominated the network. All graphs show how the algorithm probes for more data in longer distance networks by the steady growth rate in each scenario. However, in shorter range networks, CUBIC decreased its bandwidth and remained steady.

These tests show the effect of longer distance networks have on the ability for these TCPs to coexist fairly. As distance increased, CUBIC took over, but the other algorithms were still able to send. Of the three, Reno seems most fair to run with other TCP variants. YeAH appeared to be the most flexible to adapt to the network as it shifted from controlling the network to sharing the network, as network distance was decreased. CUBIC was very aggressive in increasing window size in all tests, except for when paired with itself, where it converged. As network distance decreased, CUBIC became less aggressive to prevent from starving other variants from bandwidth. Further testing of different topologies, and network distances may show how CUBIC shares the network more fully. Also, adjusting the algorithm to detect when CUBIC should enter the TCP-Friendly region, to slow down its aggressive growth in various scenarios.

# 7. References


[1] A. Esterhuizen and A.E. Krzesinski. *TCP Congestion Control Comparison.*

[2] Baiocchi A., Angelo P. Castellani and Franseco Vacirca. *YeAH-TCP: Yet Another Highpeed TCP*. http://infocom.uniroma1.it/~vacirca/yeah/yeah.pdf

[3] Arianfar, Somaya. *TCP's Congestion Control Implementation in Linux Kernel*

[4] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert and R. Scheffenegger. *CUBIC for Fast Long-Distance Networks*. https://tools.ietf.org/html/rfc8312

[5] Floyd S., Handley M., and J. Padhye. *A Comparison of Equation-Based and AIMD Congestion Control*, May 2000.

[6] Fall K., Floyd S. *Simulation based comparison of Tahoe, Reno & SACK TC.*

[7] Tetcos Engineering. *Transmission Control Protocol.*

[8] L. Xu, Harfoush K., and I. Rhee. *Binary increase congestion control (bic) for fast long-distance networks.*