Study of Different TCP Protocols in Wireless Network

Name: Sayem Kabir
ID: 2017-1-55-023

Name: Sadia Rahman
ID: 2018-2-55-006

Name: Akash Chandra Biswas
ID: 2018-1-55-019

Supervised By

Dr. Anup Kumar Paul
Associate Professor, Department of Electronics & Communications Engineering

This Thesis Paper is Submitted in Partial Fulfillment of the Requirements of the Degree of Bachelor of Science in
“Electronic & Telecommunication Engineering”,
Department of Electronics & Communications Engineering

EAST WEST UNIVERSITY

Approval

The thesis titled “Study of different TCP protocols in wireless network” submitted by Sayem Kabir (ID: 2017-1-55-023), Sadia Rahman (ID: 2018-2-55- 006) and Akash Chandra Biswas (ID: 2018-1-55-019) to the Department of Electronics and Communications Engineering, East West University, Dhaka, Bangladesh has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Electronic and Telecommunication Engineering and approved as to its style and contents.

Approved By

____

Supervisor

Dr. Anup Kumar Paul
Associate Professor
ECE Department
East West University
Dhaka, Bangladesh

DECLARATION

We declare that our work has not been previously submitted and approved for the award of a degree by this or any other University. As per our knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made in the thesis itself. We hereby declare that the work presented in this thesis is the outcome of the investigation performed by us under the supervision of Dr.Anup Kumar Paul, Associate Professor, Department of Electronics & Communications Engineering, East West University, Dhaka, Bangladesh.

Countersigned

____

Supervisor

Dr. Anup Kumar Paul

Signature

____

Sayem Kabir

ID:2017-1-55-023

Signature

____

Sadia Rahman

ID:2018-2-55-006

Signature

____

Akash Chandra Biswas

ID:2018-1-55-019

ACKNOWLEDGEMENT

We offer our genuine thanks to our supervisor, Dr. Anup Kumar Paul for his significant direction and guidance from the proposed idea to the implementation of that in our research. His tendency to offer his works and time is extraordinarily refreshing. We might likewise want to express gratitude toward him for his supportive interactive behavior and extraordinary inclination. Throughout our research, we gained numerous important experiences and ideas about TCP protocols and their implementations from our supervisor. His persevering consolation gave us the certainty to manage our task. Ultimately, we thank the Almighty and our supervisor for the overwhelming support. This thesis would not have been possible without our supervisor's judicious support throughout the research work.

ABSTRACT

Today's world is extremely dependent on the internet, and so as a result, the usage of the internet in all aspects of our life is rapidly rising. The congestion control algorithm is a crucial component of TCP, and it was built on the idea that packet loss is generally relatively minimal, and that packet loss is consequently an indicator of network congestion. The congestion control algorithm used by a specific version of TCP determines its performance characteristics. This thesis will give an overview of different TCP variants, and their characteristics in the wireless network. In this paper, we have experimented with four TCP variants (TCP CUBIC, TCP Vegas, TCP Yeah, TCP Westwood Plus) and noticed their performance for increasing nodes. For the increasing nodes, the throughput decreases. The performance of TCP protocols are experimentally evaluated with an implementation in Linux using Network Simulator Version-3 (NS3). We have chosen the desired file of wifi-tcp.cc from NS3 and edited it according to our experimental need. The performance of throughput in CUBIC gives the best result according to our simulation.

Contents

Chapter 1

Introduction

1.1 Introduction

Van Jacobson's TCP congestion control system is based on a sliding window mechanism and uses an Additive Increase Multiplicative Decrease (AIMD) algorithm to match transmission rate to available network resources [12]. The sender identifies packet losses and changes the transmission rate depending on the acknowledgment (ACK) feedback provided by the receiver. In a wired network, the TCP congestion control algorithm is meant to reduce congestion losses. The adoption of the TCP congestion control algorithm improves TCP's performance as a reliable end-to-end data transport in wired networks. When the TCP congestion control algorithm is used in a wireless network, however, TCP performance suffers [11]. The TCP protocol, which supports the bulk of Internet services (Web, FTP, Telnet), is one of those mechanisms that is intrinsically inefficient on wireless networks due to its architecture. This thesis paper focuses on evaluating and comparing the throughput performance of four TCP variants: TCP Cubic, TCP Vegas, TCP Yeah, TCP Westwood Plus. The ultimate objective is to figure out which TCP protocol in wireless works best. The capability of the protocol to differentiate between various forms of packet loss and respond accordingly is the core emphasis of all TCP solutions for wireless and heterogeneous networks. TCP-based Internet applications are expected to continue to do so in the future. With the widespread adoption of wireless networks, it's critical to support these applications in both wired and wireless contexts. So the desired throughput and performance are hence an important part.

CUBIC's Linux implementation has gone through multiple revisions. The most noteworthy improvement is the improved efficiency with which cubic root calculations are performed. Implementing it in the kernel necessitates some integer approximation because it involves a floating-point operation. It started with the bisection approach and then switched to the Newton-Raphson method, which cuts the computing cost by approximately tenfold. The removal of window clamping was another adjustment made to CUBIC after its creation. BIC-TCP introduced window clamping, in which window increments are limited to a maximum increment, which was inherited by CUBIC for the first version. When the goal mid-point is substantially greater than the current window size, this compels the window expansion to be linear.

Every new TCP protocol invents due to the drawback of the previous protocol. So, for some issues of TCP Reno, TCP Vegas was introduced. The loss of segments is used by TCP Reno's congestion detection and control methods as a warning that the network is congested. As a result, TCP Reno lacks the means to identify the early stages of congestion before losses occur and hence is unable to avoid such losses [12]. As a result, TCP Reno is reactive, as it must induce losses in order to determine the connection's available bandwidth. TCP Vegas congestion detection system, on the other hand, is proactive, in that it seeks to identify impending congestion by analyzing changes in the throughput rate. TCP Vegas may be able to cut the transmitting rate before the connection suffers losses since it infers the congestion window adjustment strategy from such throughput metrics. TCP Westwood's issues in the presence of compressed/delayed ACKs prompted a change in the ACK filter used in bandwidth calculation, resulting in the formation of the Westwood plus protocol [12]. In our simulation with the increasing number of nodes, we find TCP CUBIC to work more efficiently. Though the differences in throughputs among TCP CUBIC, Westwood plus, and Yeah are almost similar while simulating in Linux, TCP CUBIC performs a bit better. That's the desired result as in Linux, TCP CUBIC has been implemented.

1.2 Problem Statement

Everyone uses a network enable devices in today's era. The network follows a model called the OSI model. In the OSI model Transmission control protocol performs a vital rule[3]. To establish TCP on a network, we choose different variants of TCP like TCP Vegas, TCP Cubic, TCP YEAH, and TCP WestwoodPlus. These variants perform their best with fewer nodes or devices. When the number of the device increases throughput of the variants decreases. Some variants' throughput decreases rapidly, some show constant throughput after two or three nodes. This is the problem that we are worrying about. TCP Cubic and TCP Yeah show higher throughput compare to TCP Vegas. But, TCP Vegas shows a constant throughput as the station node or device increases in the network. On the other hand, TCP Westwood plus shows high throughput but decreases sharply as the node increases. The goal of this thesis can be started as

- Increasing Station Node or devices.
- Finding out the best variants.
- Calculating the throughput of the variants.
- Comparing the throughput.
- Finding out the best variants on purpose

1.3 Motivation

There are enormous possibilities for Transmission control protocol. Transmission Control Protocol/Internet Protocol model is to permit communication over enormous distances[25]. We can not replace TCP, and its variant's from a network system. The variants of TCP can be used for reliable and fast communication. The throughput of different variants like TCP Vegas, TCP Cubic, TCP YEAH, and TCP WestwoodPlus shows that communicating can be faster by twerking or altering some parameters. The future of communication can be more flexible and faster. We can ensure the security as well as reliability of a network by using the different variants of TCP. As we know, TCP Cubic used in the Linux Operating system. It shows us how reliable a TCP variant can be. If we do more research on different variants of TCP, we can get higher throughput which will ensure faster communication with a large amount of devices connected[26].

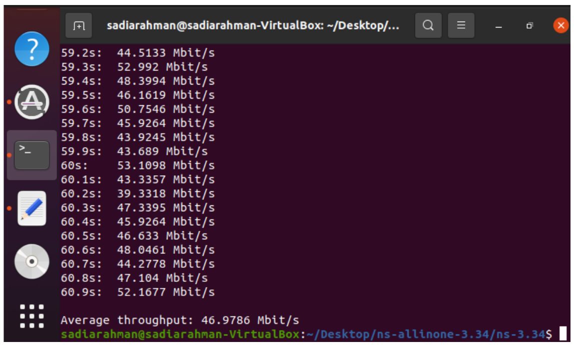
Chapter 2

Methodology

This section of the thesis refers to the chosen methods for justification or analysis of a given data or required scenario. This thesis paper is simulation-based. At first, we installed Linux Operating System. Then we installed version-3 of Network Simulator (NS3). We chose the file wifi-tcp.cc, for our desired simulation. In this file we had to edit some important parameters. The default file is given for one Access point (Ap) one Stationary point (STA) for a 10-second simulation time.As it is not possible to understand the throughput behavior of TCP protocols from only 1 node, so

- We increased the number of Stationary points. We have taken up to 7 stationary points to understand the average throughput of our four TCP variants.
- We increased the simulation time to 60 seconds to observe the throughput for a longer time.
- As we decided to increase the stationary points, so we had to add a different IPv4 address for each stationary points.
- We set the data rate of 100Mbps to observe the changes and compare effectively

From the Terminal of Linux, we ran the simulation for 60 seconds and achieved each variant of TCP with different throughput result. The output looks like below figure:



```
59.2s:  44.5133 Mbit/s
59.3s:  52.992 Mbit/s
59.4s:  48.3994 Mbit/s
59.5s:  46.1619 Mbit/s
59.6s:  50.7546 Mbit/s
59.7s:  45.9264 Mbit/s
59.8s:  43.9245 Mbit/s
59.9s:  43.689 Mbit/s
60s:    53.1098 Mbit/s
60.1s:  43.3357 Mbit/s
60.2s:  39.3318 Mbit/s
60.3s:  47.3395 Mbit/s
60.4s:  45.9264 Mbit/s
60.5s:  46.633 Mbit/s
60.6s:  48.0461 Mbit/s
60.7s:  44.2778 Mbit/s
60.8s:  47.104 Mbit/s
60.9s:  52.1677 Mbit/s

Average throughput: 46.9786 Mbit/s
sadiarahman@sadiarahman-VirtualBox:~/Desktop/ns-allinone-3.34/ns-3.34$
```

Figure 2.1: Throughput

Chapter 3

TCP Fundamentals

3.1 TCP Fundamentals

Transmission Control Protocol(TCP) enables application programs and computing devices to exchange messages over a network [15]. This operation can execute in two way

1. **Full Duplex**
2. **Reliable Delivery**

The reliability is ensured by using Connection-oriented service. Then, we can use error detection using checksum. Then, we can use error control using go-back N ARQ(Automatic Repeat Request) technique. Then, we can use flow control using sliding window protocol. After that, we can use congestion avoidance protocol (Multiplicative Decrease and Slow Start)

3.1.1 TCP Datagram

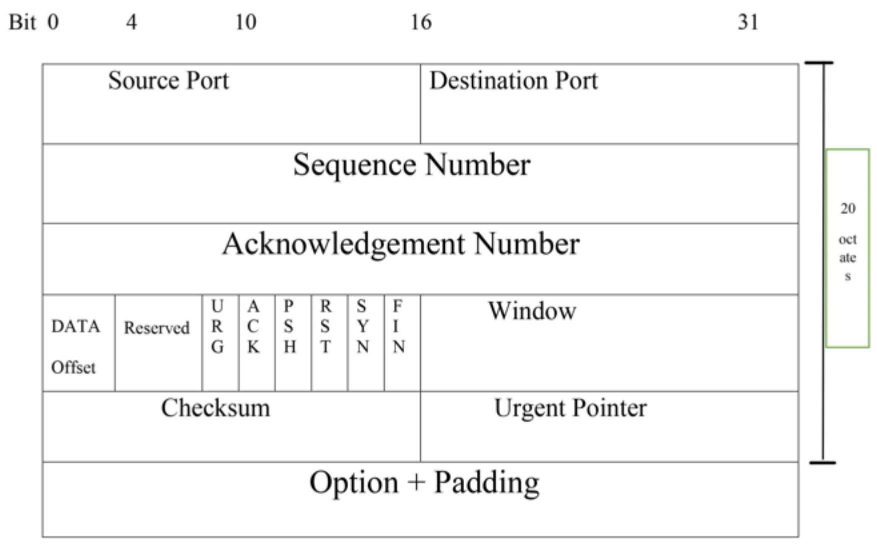


Figure 3.1: TCP Datagram Format

1. Source Port (16 bits): It defines the port number of the application program in the host of the sender.
2. Destination Port(16 bits): It defines the port number of the application program in the host of the receiver.
3. Sequence Number(32 bits): It conveys the receiving host which octet in this sequence comprises the first byte in the segment.
4. Acknowledgement Number(32 bits): This specifies the sequence number of the next octet that receiver expects to receive.
5. HLEN(4 bits): Header Length mainly specify the number of 32 bit words present in the TCP header.
6. URG: Urgent Pointer
7. ACK: Indicates whether acknowledge field is valid.
8. PSH: Push the data without buffering.
9. RST: Resent the connection.
10. SYN:Synchronize sequence numbers during connection establishment.
11. FIN: Terminate the connection.
12. Window(16 bits): Specifies the size of window.
13. Checksum(16 bits): Checksum used for error detection.
14. Option: Optional 40 bytes of information

3.1.2 Working Principle of TCP

We know that TCP operates in two ways from our earlier knowledge.

- Full Duplex
- Reliable Communication For connection establishment in full-duplex mode, a four-way protocol can be used. However, the second and third steps can be combined to form a three-way handshaking protocol with the following three steps: [7]

1. **Step-1:**The client sends SYN segment, which includes, source and destination port numbers, and an Initialization Sequence Number (ISN), which is essentially the byte number to be sent from the client to the server.
2. **Step-2:**The server sends a segment, which is a two-in-one segment. It acknowledges the receipt of the previous segment and it also acts as initialization segment for the server.
3. **Step-3:**The sends an ACK segment, which acknowledges the receipt of the second segment.

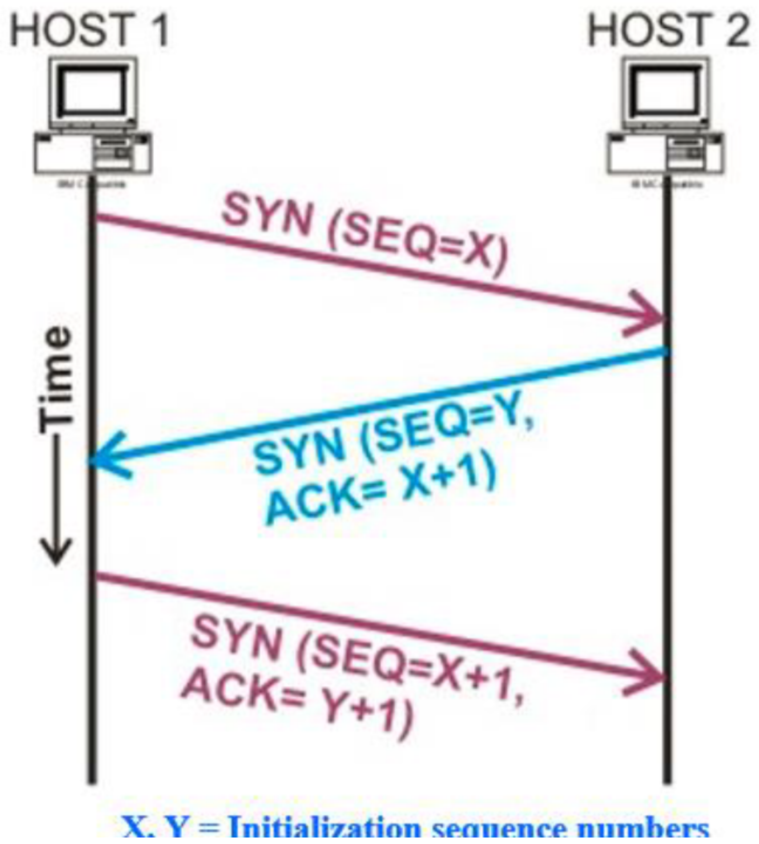


Figure 3.2: Connection Establishment Protocol in TCP

To terminate the connection in both direction a four way hand shaking protocol is necessary [18]. The four steps are as follows:

1. **Step-1:**The client sends a FIN segment to the server.
2. **Step-2:**The server sends an ACK segment indicating the receipt of the FIN segment and the segment also acts as initialization segment for the server.
3. **Step-3:**The server can still continue to send data and when the data transfer is complete it sends a FIN segment to the client.
4. **Step-4:**The client sends an ACK segment, which acknowledges the receipt of the FIN segment sent by the server.

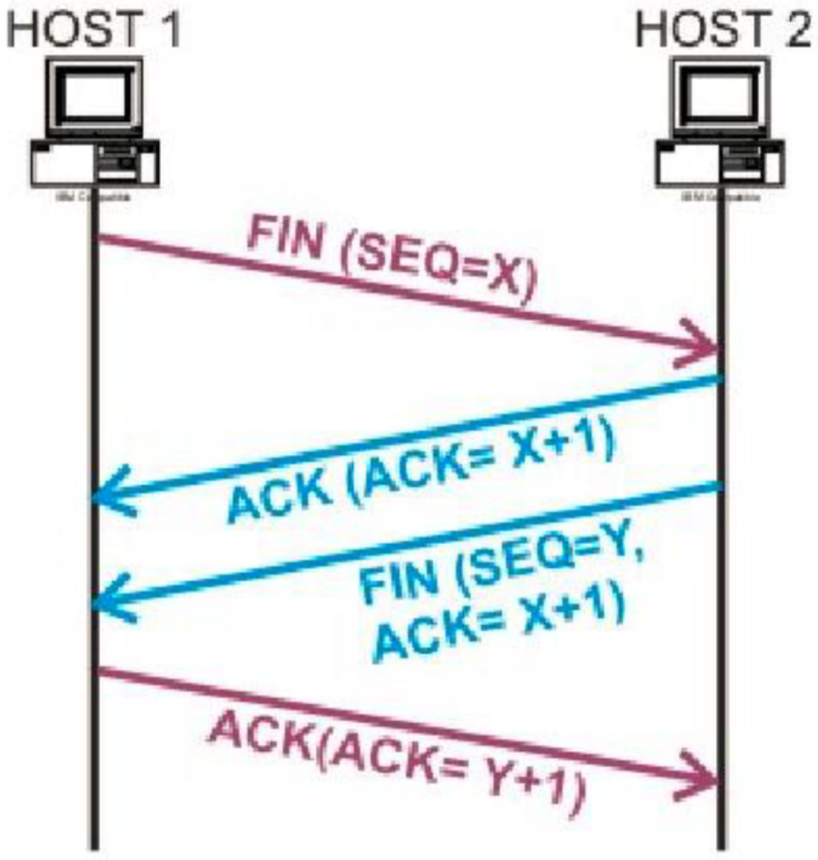


Figure 3.3: Connection Termination Protocol in TCP

To ensure reliable communication,[20] TCP performs the following methods:

1)**Flow Control**

TCP uses byte-oriented sliding window protocol, which allows efficient transmission of data and at the same time the destination host is not overwhelmed with data. The receiver has a buffer size of 8 K bytes. After receiving 4 K bytes, the window size is reduced to 4 K bytes. After receiving another 3 K bytes, the window size reduces to 1 K bytes. After the buffer gets empty by 4 K bytes, the widow size increases to 7 K bytes. So, it may be noted that the window size is totally controlled by the receiver window size, which can be increased or decreased dynamically by the destination. The destination host can send acknowledgement any time.

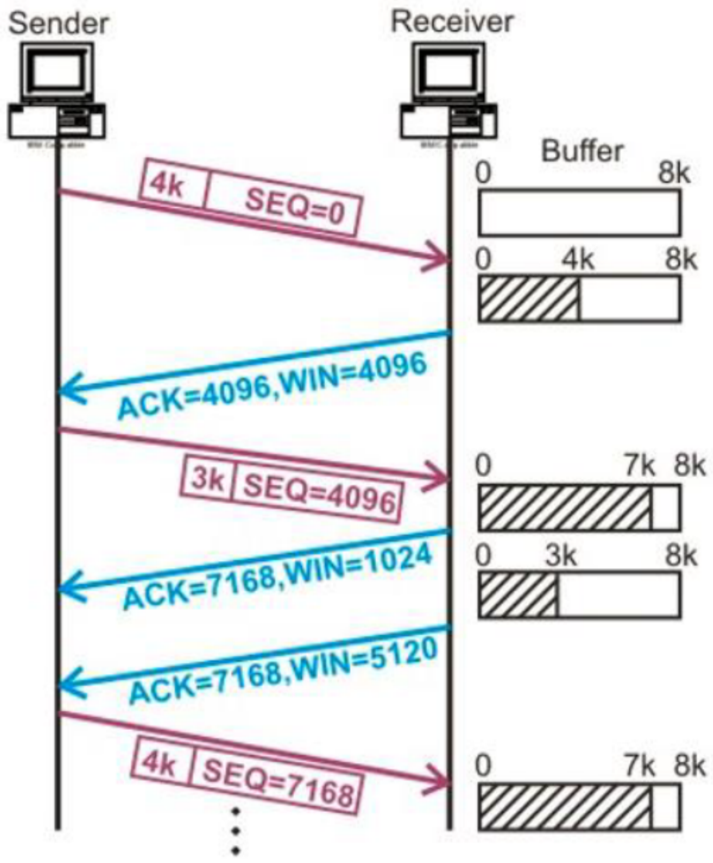


Figure 3.4: Flow Control

2) **Error Control**

TCP includes mechanism for detecting corrupted segment with the help of checksum field. Acknowledgement method is used to confirm the receipt of un-corrupted data. There is no negative acknowledgement in TCP. If the acknowledge is not received before the timeout, it is assume that the data has been corrupted of lost.

To keep track of lost or discarded segments and to perform the operations smoothly, the following four timers are used by TCP:

- **Re-transmission:**It is dynamically decided by Round Trip Time (RTT) .
- **Persistence:**This is used to deal with window size advertisement.
- **Keep-alive:**Used in situations where there is long idle connection between two processes.
- **Time-waited:**It is used during the connection termination.

3.1.3 Congestion Control

Congestion occurs when bandwidth is insufficient and network data traffic exceeds capacity. Congestion window determines the number of the bytes that can be sent out at any time [24]. To avoid congestion, the sender process uses two strategies known as

1. Slow start and Additive Increase
2. Multiplicative Decrease

Slow Start and Additive Increase At the beginning, the congestion is set to the maximum segment size. For each segment that is acknowledged, the size of the congestion window size is increased by maximum segment size until it reaches one half of the allowable window size. Ironically, this is known as slow start [21].

In the additive increase, the rate of the increase is exponential. After reaching the threshold, the window size is increased by one segment for each acknowledgement. This continues till there is no time out.

Multiplicative Decrease Multiplicative decrease happens when a time out occurs, the threshold is set to one half of the last congestion window size [9].

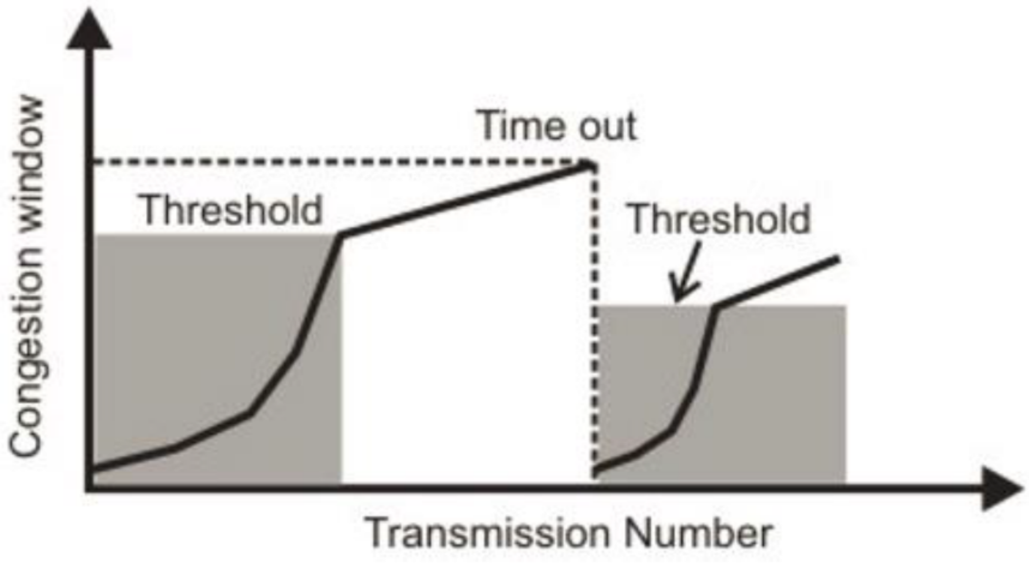


Figure 3.5: Congestion Control in TCP

Chapter 4

TCP Variants

4.1 TCP Cubic

In order to increase TCP scalability across rapid and long distance networks, the protocol alters the linear window growth function of existing TCP standards to be a cubic function.In linux, by default Cubic is used as TCP algorithm. Cubic is a more advanced form of TCP BIC.Cubic converted the normal TCP's linear window growth function to a cubic function. Cubic reduces the size of congestion windows (cwnd) during communication in Saturation-States and in Stable-states it raises it's size instantaneously.This characteristic enables Cubic to be very scalable when the network's capacity and delay product is enormous, while simultaneously being very reliable and fair to regular TCP flows.The window growth function in Cubic is a cubic function with a form that is remarkably similar to the growth function in BIC.Cubic is intended to simplify and improve BIC's window control.

4.1.1 Window Growth Function of CUBIC

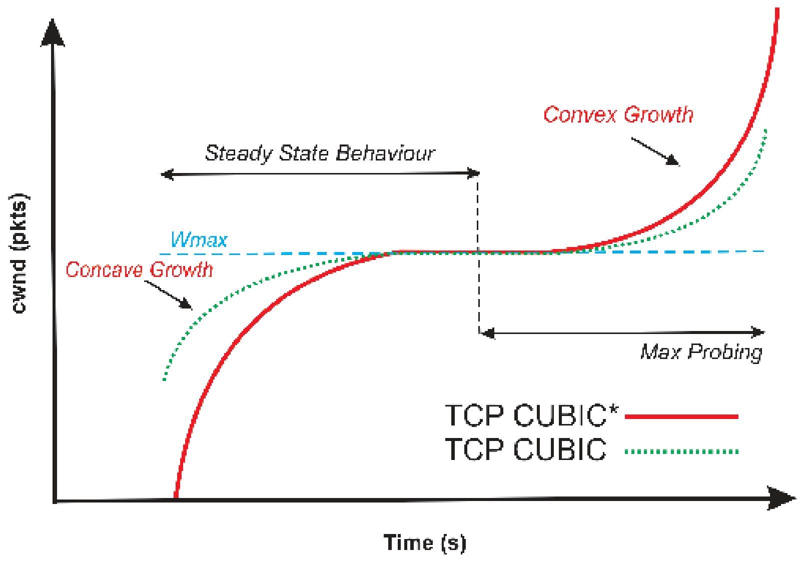


Figure 4.1: Window Growth Function of Cubic

The window growth function of CUBIC, as the name implies, is a cubic function with a form quite similar to that of BIC-TCP. CUBIC employs a cubic function of the time elapsed since the last congestion incident. CUBIC employs both the concave and convex profiles of a cubic function for window increase, whereas most alternative algorithms to Standard TCP use a convex increase function where the window increment is constantly growing following a loss event [6].The following function determines CUBIC's congestion window:

$$W(t) = C(t-K)^3 + W_{max} \tag{4.1}$$

where C is a scaling factor, t is the elapsed time from the last window reduction, W_{max} is the window size just before the last window reduction and k is the time period that the above function takes to increase W to W_{max}. We can find k by following equation:

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{4.2}$$

CUBIC uses the first equation to calculate the window growth rate for the following RTT interval after getting an ACK during congestion avoidance.It chooses W(t + RTT) as the congestion window's candidate target value. Consider the current window size of cwnd. CUBIC operates in three modes depending on the value of cwnd.CUBIC is in TCP mode if cwnd is less than the window size that (standard) TCP would reach at time t following the previous loss event. Otherwise, CUBIC is in the concave region if cwnd is less than W_{max}, and in the convex region if cwnd is more than W_{max}.

4.1.2 TCP-friendly Region

When we get an ACK in congestion avoidance, we first determine whether or not the protocol is in the TCP area. This is accomplished as follows. We can calculate the TCP window size in terms of the elapsed time t. We can discover the average window size of additive increase and multiplicative decrease (AIMD) with an additive factor, α and a multiplicative factor,β using a basic analysis in to be the following function:

$$cwnd(t) = \frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{2-\beta}{\beta}\frac{1}{p}} \tag{4.3}$$

By the same analysis, the average window size of TCP with α=1 and β=0.5 is (1/RTT)*√((3/2)(1/P)). Thus, for the above equation to be the same as that of TCP, α must be equal to $\frac{3\beta}{2-\beta}$ .If TCP increases its window by α per RTT, we can get the window size of TCP in terms of the elapsed time t as follows:

$$W_{tcp(t)} = W_{max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT} \tag{4.4}$$

If cwnd is less than Wtcp(t), then the protocol is in the TCP mode and cwnd is set to Wtcp(t) at each reception of ACK.

4.1.3 Convex Region:

When the window size of CUBIC is greater than W_{max}, it passes through the cubic function's plateau, after which it follows the cubic function's convex profile.Because cwnd is more than the previous saturation point W_{max}, it is possible that network circumstances have changed since the last loss event, signaling that there is more available bandwidth after some flow departures. Because the Internet is extremely asynchronous, changes in available bandwidth are unavoidable. The convex shape guarantees that the window grows slowly at first and then progressively expands in size.We also call this phase as the maximum probing phase since CUBIC is searching for a new W_{max}.Because we do not change the window growth function simply for the convex region, the window growth function for both regions stays unchanged.

4.1.4 Concave Region

If the protocol is not in TCP mode and cwnd is smaller than W_{max} upon getting an ACK in congestion avoidance, the protocol is in the concave region.

4.1.5 Fast Convergence

CUBIC incorporates a heuristic into the protocol to increase the pace of convergence. Existing CUBIC flows can use this heuristic to release (share) bandwidth to incoming flows. With this extra bandwidth, incoming flows can expand.When a packet loss event happens and the fast convergence mechanism is activated, the algorithm compares the prior W_{max} to the current W_{max}. If the current W_{max} is lower than the previous W_{max}, it means that the connection has less available resources.If the current value of W_{max} is smaller than the previous value, $W_{last-max}$, during a loss event, this indicates that the saturation point encountered by this flow is decreasing due to the change in available band-width. Then we reduce W_{max} even further to allow this flow to release additional bandwidth. Because the reduced W_{max} drives the flow to plateau sooner, this action effectively extends the time for this flow to grow its window. This gives the new flow more time to adjust its window size.

4.1.6 Pluggable Congestion Module

CUBIC is a pluggable traffic control module that has been implemented. The hooks that CUBIC uses for its implementation are listed below:

1. bictcp_init: It initializes private variables used for CUBIC algorithm. If initial ssthresh is not 0, then set ssthresh to this value.If initial ssthresh is properly set by users when there is no history information about the end-to-end path,it can improve the start-up behavior of CUBIC significantly.

2. bictcp_cong_avoid: It raises cwnd by determining the difference between the current cwnd value and the expected value of the next RTT round, which is calculated using the cubic root.

3. bictcp_set_state: It resets all the variables when a timeout happens.

4. bictcp_undo_cwnd: It returns the maximum between the current cwnd value and the last max (which is the congestion window before the drop).

5. bictcp_acked: It maintains the minimum delay observed so far. The minimum delay is reset when a timeout happens.

6. bictcp_recalc_ssthresh: If the fast convergence mode is turned on and the current cwnd is smaller than last_max, set last_max to cwnd$*$(1-$\beta/2$).Else set last_max to cwnd$*$(1-β). ssthresh always set to cwnd$*$(1-β) because TCP needs to back off for congestion [8].

4.1.7 Advantages of TCP CUBIC

It has a high bandwidth utilization rate, particularly for small bandwidth-delay product networks. Even in the midst of background traffic,it operates admirably.CUBIC improves on BIC's fairness features while keeping its scalability and stability.Because the growth function is independent of RTT, it guarantees RTT fairness because various RTT flows will still expand their windows at the same rate.When the buffer size is less than the bandwidth-delay product,CUBIC gives good throughput.CUBIC has intra fairness protocol among the same protocol.

4.2 TCP Vegas

TCP Vegas was proposed in 1994 as an alternate source-based Internet congestion control mechanism [10]. Unlike the TCP Reno method, which produces congestion to learn about available network capacity, a Vegas source predicts the start of congestion by monitoring the difference between the rate it expects to observe and the rate it sees. Vegas' method is to change the sending rate of the source (congestion window) to retain a limited number of packets delayed in the routers along the transmission line. TCP Vegas identifies congestion at an early stage by raising the Round-Trip Time (RTT) values of the packets in the connection, as opposed to other flavors such as Reno, New Reno, and others, which detect congestion only after it has occurred through packet loss. The algorithm is strongly reliant on an accurate estimate of the Base RTT value. If the number is too little, the connection's throughput will be less than the available bandwidth, while too big will overrun the connection.

Vegas uses a more advanced bandwidth estimate approach that seeks to prevent congestion rather than react to it. It precisely calculates the number of data packets that a source may deliver based on the measured RTT. Its window adjustment method is divided into three stages: slow start, congestion avoidance, fast retransmit, and fast recovery. The congestion window is updated based on the current phase of execution.

4.2.1 Fast Retransmit

TCP Vegas makes three modifications to TCP's (fast) retransmission technique. TCP Vegas first calculates the RTT for each segment transmitted. Fine-grained clock values are used in the measurements. A timeout duration for each segment is calculated using the fine-grained RTT values. When TCP Vegas receives a duplicate acknowledgment (ACK), it checks to see if the timeout period has elapsed. If this is the case, the section is retransmitted. Second, when TCP Vegas receives a non-duplicate ACK that is the first or second after rapid retransmission, it checks for the timer's expiry and may retransmit another segment. Third, when several segments fail and more than one fast retransmission is attempted, the congestion window is decreased only for the first fast retransmission [4].

4.2.2 Congestion Avoidance Mechanism

TCP Vegas does not continuously increase the congestion window throughout the congestion avoidance phase. Instead, it attempts to detect potential congestion by comparing actual throughput to expected throughput. Vegas calculates the appropriate amount of excess data to keep in the network pipe and adjusts the congestion window size accordingly. It logs the RTT and sets BaseRTT to the shortest round-trip time ever measured. The quantity of additional data (△) is calculated as follows:

$$\triangle = (Expected - Actual) \times BaseRTT \tag{4.5}$$

where Expected throughput is the current congestion window size (CWND) divided by BaseRTT, and Actual throughput represents the CWND divided by the newly measured.When △ is between two thresholds α and β, the CWND remains constant. If △ is more than β, it is interpreted as an indicator of impending congestion, and the CWND is decreased. If, on the other hand, △ is less than α, the connection may be under-utilizing its available bandwidth.As a result, the CWND will be raised. CWND is updated on a per-RTT basis. The rule for adjusting the congestion window is as follows:

$$CWND = \begin{cases} CWND + 1, & if \triangle < \alpha \\ CWND - 1, & if \triangle > \beta \\ CWND & if \alpha \leq \triangle \leq \beta \end{cases} \tag{4.6}$$

4.2.3 Slow-start Mechanism

Vegas plans for a connection to rapidly ramp up to the available bandwidth during the slow-start period. To identify and prevent congestion during slowstart, however, Vegas increases the size of its congestion window only every other RTT. In the interim, the congestion window remains constant, allowing a realistic comparison of expected and actual throughput. During the slow start, a similar congestion detecting process is used to determine when to transition the phase.

4.2.4 Window Control of TCP Vegas

Figure 4.2 illustrates the behavior of TCP vegas.Consider a basic network with a single connection and a single link of capacity C. BaseRTT be the minimum round trip delay.When window < C∗BaseRTT the throughput of this connection is $\frac{window}{baseRTT}$.Here w corresponds to window size where window= C $\times BaseRTT$. When window > w, queue starts to build up and (Expected-Actual)> 0.If window < w+α, TCP Vegas increases the window size by one during the next round trip time. And if window > w+β, it decreases the window size by one. TCP Vegas tries to keep at least α packets but no more than β packets in the queues. The reason behind this is that TCP Vegas attempts to detect and utilize the extra bandwidth whenever it becomes available without congesting the network.When there is only one connection, TCP Vegas's window size converges to a position between w $+\alpha$ and w + β. TCP Vegas does not cause any oscillation in window size once it converges to an equilibrium point [1].

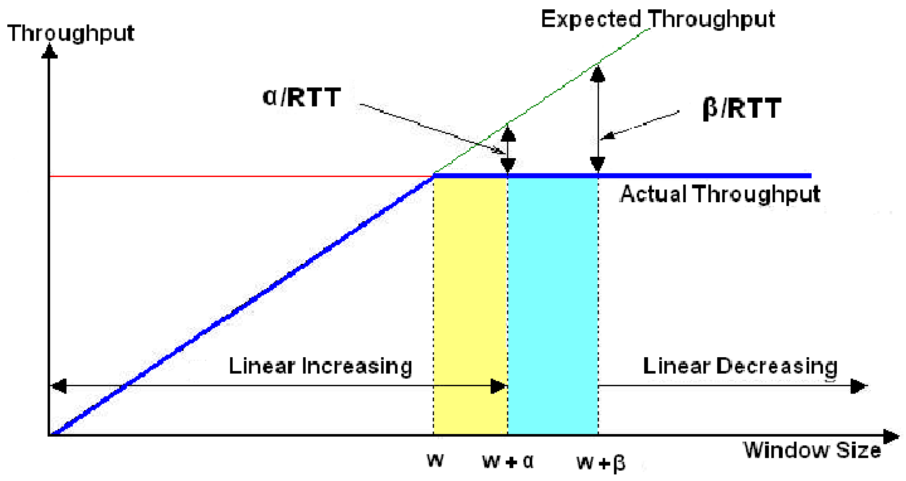


Figure 4.2: Window Control of TCP Vegas

4.2.5 Rerouting

Because TCP Vegas adjusts its window size based on an estimate of the propagation delay, baseRTT, it is critical for a TCP Vegas connection to have an accurate estimation. Rerouting a path may modify the propagation latency of the connection, resulting in a significant drop in throughput.Another critical problem is TCP Vegas's stability. Because each TCP Vegas connection seeks to maintain a few packets in the network, when their estimated propagation delay is o, the connections may accidentally store many more packets in the network, producing continuous congestion [16]. If a switch changes the path of a connection, the end host cannot notice it without an explicit indication from the switch. If the new route has a reduced propagation delay, TCP Vegas is unaffected since certain packets will most likely have a shorter round trip delay and BaseRTT will be modified. However, if the new route for the connection has a longer propagation delay, the connection will be unable to determine whether the increase in round trip time is due to network congestion or a change in the route.Without this knowledge, the end host would perceive an increase in round trip latency as a hint of network congestion and reduce the window size. This, however, is the inverse of what the source should do. When connection i is propagation delay is d_i, the expected number of backlogged packets is w_i - r_i∗d$_i$, where w_i is connection i's window size r_i is the flow rate.Because TCP Vegas tries to retain between α and β packets in the switch buffers, if the propagation delay rises, the window size should be increased to maintain the same amount of packets in the buffer. As TCP Vegas relies on delay estimate, this might have a significant influence on performance. Because network switches do not inform connections of changes in routes, it is necessary for connection sources to be able to detect such changes. The following modification has been proposed by La et al [17]. The modified protocol behaves similarly to TCP Vegas for the first K packets, where k is a prefixed parameter. The sources maintain track of the minimum round trip delay of N successive packets once the ACK for the Kth packet arrives. If the minimum round trip time of the last L.N packets is significantly more than the current baseRTT, the source updates the baseRTT to the minimum round trip time of the latest N packets and resets the congestion window size based on this baseRTT. The essential idea underlying this process is as follows. If the minimum round trip time determined for N packet is regularly substantially more than baseRTT, then the real propagation delay is likely to be greater than the measured baseRTT, and it makes sense to raise baseRTT. However, it is feasible that the rise is due to network congestion. Because the increase in delay leads the source to reduce its window size, the trip delay is mostly caused by the propagation delay of the new route. As a result, the lowest round trip delay of the preceding N packets provides a solid approximation of the new propagation delay [19].

4.2.6 Continuous Congestion

Because TCP Vegas employs baseRTT to estimate route propagation time, its performance is affected by baseRTT correctness. As a result, if the connections overestimate the propagation delay owing to improper baseRTT, it might have a significant influence on TCP Vegas performance. We begin with a situation in which the links overestimate the propagation delays, potentially driving the system into a continuously congested condition [5] Assume that a connection begins when there are many other existing connections, the network is congested, and the queues are full. The packets from the new connection may then encounter round trip delays that are far longer than the path's actual propagation delay due to the queuing delay from other backlogged packets. As a result, the new connection will set the window size to a value that leads it to assume that its expected number of backlogged packets is between α and β, while in fact it has many more backlogged packets due to an incorrect estimate of the path's propagation delay.This situation will be repeated for each new connection, and it is feasible that the system may be under permanent congestion as a result. This is the total opposite of a favorable circumstance. When the network is overloaded, we don't want additional connections to exacerbate the situation. The same thing might happen with TCP Reno or TCP Tahoe. TCP Vegas, on the other hand, is more likely to happen because to its fine-tuned congestion avoidance mechanism. When the network remains consistently congested, the connections perceive the constant rise in round trip time as an increase in propagation delay and update their baseRTT accordingly. This causes a momentary spike in network congestion, and most, if not all, connections fail when they notice the congestion. The congestion level decreases when connections decrease their window widths, allowing the connections to estimate the right baseRTT. Congestion will stay low after most connections have a good measurement of propagation delay.

4.3 YeAH-TCP

One more high speed TCP congestion control algorithm uses a mixed loss/delay approach to calculate congestion windows. Its goal is to maximize efficiency, fairness, and minimize link loss while minimizing the load on the network elements. Several factors are considered, such as bandwidth exploitation efficiency, average packet delay, fairness internal and external, friendliness to Reno, robustness to random loss.

4.3.1 Congestion-Control:

The congestion window (CWND) sets the maximum number of bytes that can be sent out at any given time in TCP. The sender maintains the congestion window, which prevents a link between the sender and the receiver from becoming overburdened with traffic. This is not to be confused with the sender's sliding window, which exists to keep the receiver from becoming overloaded. The congestion window is determined by assessing the amount of traffic on the network.When a connection is established, the congestion window, which is maintained independently by each host, is set to a tiny multiple of the connection's maximum segment size (MSS) [23]. An additive increase/multiplicative decrease (AIMD) strategy dictates more variation in the congestion window. This means that if all segments are received and acknowledgements are received on time, the window size is increased by a constant. Different algorithms will be used.As part of TCP tuning, a system administrator can change the maximum window size limit or the constant introduced during additive increase. The use of the receive window advertised by the receiver also controls the flow of data over a TCP connection. A sender is limited to sending data within its congestion window and the receive window.

4.3.2 Congestion-Control Algorithms in YeAH-TCP:

YeAH-TCP, the data transmission protocol utilized by many Internet services, employs congestion control measures (or algorithms). A TCP algorithm's principal purpose is to avoid transferring more data than the network can handle, or to avoid network congestion. Different algorithms react to network loads in different ways, but they all follow the same premise of preventing network congestion. It is a sender-side high-speed enabled TCP congestion control technique that computes the congestion window using a mixed loss/delay approach. The goal is to provide high efficiency, a minimal RTT and Reno fairness, and link loss resilience while minimizing the strain on network nodes [2]

4.3.3 YeAH-TCP: ALGORITHM DESIGN

We considered several goals when designing YeAH-TCP:

The network's capacity should be fully utilized. This is the most obvious goal, which may be accomplished by changing the congestion window update rules; as explained below, YeAH TCP can use any of the increment rules from other proposals (e.g., STCP, H-TCP, etc.).

The network stress should be less than or equivalent to that caused by Reno TCP. Most high-speed TCPs cause frequent congestion events at the bottleneck router, with a substantially higher number of packet drops in a single congestion event than traditional Reno congestion control, reducing the performance of other traffic sharing the path. Additionally, queue delays and delay jitter are impacted [11].

Non-congestion related (random) packet loss events should not degrade performance significantly; random packet loss cannot be ruled out even in high-speed optical backbones. Although reasonable estimates of this loss depend on the technological situation, we show that even a loss rate of 10^{-7} can cause significant performance reduction. Internally, the algorithm should be RTT fair.

High performance should not be hampered by small network buffers. In high BDP links, a buffer size equal to the bandwidth-delay product, as required by typical Reno congestion control, is not possible. This goal can be achieved by following the Westwood algorithm's reduction policy in the event of packet loss [22].

All of the listed difficulties are addressed with YeAH-TCP. Like Africa TCP, it envisions two separate modes of operation:Fast and Slow. During in the "Fast" mode, YeAH-TCP increases the congestion window based on an aggressive rule (we used the STCP rule since it is relatively easy to build). It works as Reno TCP in Slow mode. According to the projected amount of packets in the bottleneck queue, the state is determined.Let RTT^{base} be the sender's minimum RTT (i.e., a propagation delay estimate) and RTT^{min} be the minimal RTT estimated in the current data window of cwnd packets. RTT^{queue} = RTT^{min} - RTT^{base} is the total expected queuing delay. The number of packets enqueued by the flow can be calculated using RTT^{queue} as follows:

Q = RTT^{queue}.G = RTT^{queue}.(CWDN/RTT^{min}) here, G stands for goodput. The ratio of the queuing RTT to the propagation delay

L= (RTT^{queue}/RTT^{base}) can also be calculated, this reflects the level of network congestion. RTT^{min} is only updated once for each data window.[1]

If Q $<$ Q^{max} and L $<$ $1/\varphi$ the algorithm is in the Fast mode otherwise it is in the Slow mode. Here, Q^{max} and φ are two adjustable parameters. The maximum number of packets that a single flow can keep in the buffers is Q^{max}. In terms of BDP, $1/\varphi$ represents the highest level of buffer congestion.A precautionary decongestion algorithm is used in the Slow mode. When Q $>$ Q^{max}, Q reduces the congestion window and ssthresh is set to cwnd/2. The decongestion granularity is one RTT because RTT^{min} is computed once every RTT.

Q is an estimate of the excess quantity of packets in relation to the minimum cwnd required to exploit the available bandwidth when a single YeAH-TCP competes for the bottleneck link. Without reducing goodput, this number of packets can be eliminated from the actual congestion window. When the number of competing flows grows, each one tries to fill the buffer with the same amount of packets (at maximum Q), regardless of the observed RTT, to achieve internal RTT fairness.Furthermore, preventative decongestion prevents the bottleneck queue from becoming too clogged, minimizing queuing times and packet losses due to buffer overflow. Only when the flows that implement it do not compete with "greedy" sources, such as Reno TCP, is cautious decongestion ideal. When competing with "greedy" flows, precautionary decongestion reduces capacity of the conservative flow by releasing bandwidth to greedy sources.

YeAH-TCP implements a technique to identify if it is competing with "greedy" sources to avoid unfair rivalry with older flows. Consider the example of Reno flows that do not have queue decongestion implemented. Because Reno flows are "greedily" filling up the buffer, the queuing delay increases when Q is bigger than Q^{max} YeAH-TCP attempts to remove packets from the queue. In this scenario, YeAH-TCP will infrequently be in "Fast" mode and will be in "Slow" mode more often.With non-greedy competing flows, on the other hand, the YeAH algorithm will change the state from Fast to Slow anytime buffer content exceeds Q^{max} and then back as soon as the precautionary decongestion kicks in. By calculating the number of RTTs that the algorithm is in each of the two states, it is feasible to discriminate between the two different competition situations [2]

Last but not least, what happens if a packet is lost. When three duplicate ACKs indicate a loss, the current estimate of the bottleneck queue Q can be used to determine the number of packets that should be taken from the congestion window to clear the bottleneck buffer while keeping the pipe full. In principle, this rule is similar to the one employed by Westwood TCP.

4.3.4 DCE(Direct-Code-Execution) for validation of YeAH-TCP:

The goal of the project was to compare the results achieved by simulating linux YeAH with those obtained by ns-3 YeAH implementation utilizing DCE (a module built on top of ns-3). Direct Code Execution (DCE) is a ns-3 module that allows you to run existing implementations of userspace and kernelspace network protocols or programs without having to update the source code.

Features of Direct-Code-Execution:

√ Except for recompiling the code, there is no need to update the source code.
√ The simulation is completed in one step.
√ Direct-Code-Execution uses less RAM.
√ It has two different modes of operation:
DCE employs the ns-3 TCP stacks in basic mode.
√ Advanced mode, in which DCE instead employs a Linux network stack.
√ C, C++, and POSIX socket programs are supported.

4.3.5 Goals of YeAH-TCP:

The network capacity is efficiently utilized by utilizing congestion window rules and other proposals such as STCP , H-TCP. The network strain must be less than or equivalent to that imposed by Reno TCP. This implies that transmission loss should be minimized. TCP friendliness with Reno traffic. YeAH-algorithm TCP's should be able to compete fairly with Reno TCP. Internally, the algorithm must be RTT fair. Lossy links should not affect performance. High performance should not be hampered by small network buffers. Most of the issues raised here should be addressed by YeAH-TCP.

4.4 TCP Westwood PLUS:

TCP Westwood (TCPW) is a sender-only modification to TCP New Reno designed to handle high bandwidth-delay product pathways (big pipes), packet loss due to transmission or other problems (leaky pipes), and dynamic demand (dynamic pipes). TCP Westwood uses information from the ACK stream to properly determine the congestion control parameters: Slow Start Threshold and Congestion Window . TCP Westwood calculates a "Eligible Rate" that the sender uses to adjust Slow Start Threshold and Congestion Window upon loss indication or during its "Agile Probing" phase, a proposed modification to the well-known Slow Start phase. Furthermore, a system known as Persistent Non Congestion Detection (PNCD) has been developed to identify persistent lack of congestion and initiate an Agile Probing phase to efficiently use big dynamic bandwidth [14].

TCP Westwood+ is the next step in the development of TCP Westwood. TCP Westwood+ is a sender-only modification of the TCP Reno protocol stack that improves TCP congestion control performance over wired and wireless networks.After a congestion occurrence, such as three duplicate acknowledgments or a timeout, TCP Westwood+ uses end-to-end bandwidth estimation to establish the congestion window and slow start threshold. By suitably low-pass filtering the rate of returned acknowledgment packets, the bandwidth is approximated.The logic behind this method is straightforward: unlike TCP Reno, which blindly halves the congestion window after three duplicate ACKs, TCP Westwood+ adaptively establishes a sluggish start threshold and a congestion window that considers the bandwidth consumed at the moment congestion occurs. In wired networks, TCP Westwood+ considerably improves throughput and fairness across wireless lines when compared to TCP Reno/New Reno. TCP Westwood, unlike TCP Reno, which blindly halves the congestion window after three duplicate ACKs, tries to choose a slow start threshold (ssthresh) and congestion window (cwin) that are commensurate with the effective bandwidth consumed at the time congestion occurs.This method is known as quicker recovery. The suggested approach is especially useful over wireless networks, where intermittent losses caused by radio channel issues are sometimes misconstrued as a symptom of congestion by conventional TCP schemes, resulting in excessive window reduction. Throughput performance and fairness have both improved in experimental tests.TCP Reno friendliness was also observed in a series of trials, indicating that TCP Reno connections are not starved by TCPW connections. TCPW is particularly effective in mixed wired and wireless networks, with throughput gains of up to 550 percent reported. Finally, TCPW performs nearly as well as localized link layer techniques like the well-known Snoop scheme, but without the O/H of a specific link layer protocol.

4.4.1 Congestion-Control Algorithms in TCP Westwood plus:

TCP-Westwood is a sender-side-only TCP Reno modification designed to manage huge bandwidth-delay product paths with probable packet loss due to transmission or other problems, as well as dynamic demand. TCP Westwood looks for information in the ACK stream to properly determine the congestion control parameters, such as the Slow Start Threshold (ssthresh) and the Congestion Window (cwin).TCP-Westwood calculates a 'eligibility rate,' which the sender uses to adjust ssthresh and cwin when a loss is detected, or during its 'agile probing' phase, which is a proposed modification to the slow start phase. Furthermore, a system known as Continuous Non Congestion Detection was designed to identify a persistent lack of congestion and induce an agile probing phase to use high dynamic bandwidth [13].

TCP Westwood+ is a sender-only modification of the TCP Reno/ NewReno congestion control protocol stack that improves TCP congestion control performance, particularly over wireless networks. After a congestion occurrence, such as three duplicate acknowledgments or a timeout, TCPW uses end-to-end bandwidth estimation to define the congestion window and slow start threshold.By suitably low-pass filtering the rate of returned acknowledgment packets, the bandwidth is approximated. The logic behind this method is straightforward: unlike TCP Reno, which blindly halves the congestion window after three duplicate ACKs, TCP Westwood+ adaptively establishes a sluggish start threshold and a congestion window that considers the bandwidth consumed at the moment congestion occurs. TCP Westwood improves fairness in wired networks and throughput over wireless lines when compared to TCP (New) Reno.

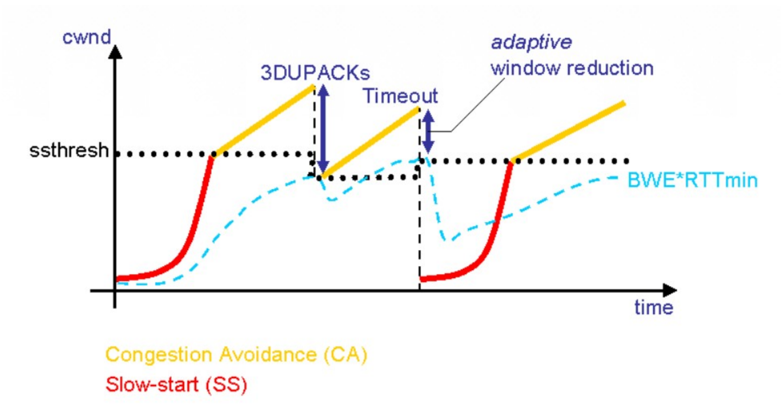


Figure 4.3: Congestion control in TCP Westwood

4.4.2 TCP WESTWOOD: ALGORITHM GUIDELINES

In this part, we'll look at how the congestion management algorithm on the sender side of a TCP connection can leverage bandwidth estimate to achieve a speedier recovery after a congestion event. First, we sketch down the algorithm in its most basic form. Then we'll go over the specific form we've used. The congestion window dynamics during slow start and congestion avoidance are unchanged, since they rise exponentially and linearly, respectively, as in current TCP Reno, as will be discussed. After a congestion episode, the congestion window (cwin) and the slow start threshold (ssthresh) are configured using the estimated bandwidth BWE. Remember that the basic job of cwin and ssthresh in TCP congestion control is to increase and decrease cwin to track the available bandwidth–delay product that ssthresh should reflect [13].

The fact that network routers may simply enforce fair queueing on FIFO queues by implementing simple queueing rules such as RED, WRED, or FRED is another important advantage of using BWE as an implicit feedback to set cwin and ssthresh. Several academics have proposed droppers in the past to assign available bandwidth to different flows based on queueing policies. While TCP Westwood does not rely on intermediate node information, it can still benefit from these queueing schemes if they exist, thanks to the exact flow-by-flow bandwidth allocation that results. Overall, if some type of fair sharing is included in the network, TCPWestwood performance increases, albeit this issue will be examined in a separate paper.After n duplicate ACKs and a coarse timeout expiration, we begin by detailing the basic algorithm behavior.

$\surd$ Algorithm after n duplicate ACKS

```
if (n DUPACKs are received)
if (cwin > ssthresh) /* congestion avoid. */
ssthresh = f1(BWE*RTTmin);
cwin = ssthresh;
endif
```

```
if (cwin<ssthresh) /*slow start */
ssthresh= f2(BWE*RTTmin)
if (cwin > ssthresh)
cwin = ssthresh
endif
endif
endif
```

We probe for extra available bandwidth during the congestion avoidance phase. As a result, receiving n DUPACKS indicates that the network capacity has been reached (or that, in the case of wireless links, one of more segments were dropped due to sporadic losses). Thus, the congestion window is set equal to the ssthresh, the slow start threshold is set equal to the available pipe size, which is BWE RTTmin, and the congestion avoidance phase is initiated again to gently probe for fresh available bandwidth. The f1 function adds one degree of freedom to the algorithm, which can be used to fine-tune it. We're still probing for available bandwidth throughout the slow start phase. As a result, the slow start threshold is set using the BWE obtained after n duplicate ACKs. The congestion window is made equal to the slow start threshold only if cwin>ssthresh after ssthresh has been set. In other words, as in the current implementation of TCP Reno, cwin still has an exponential increase during slow start. The f2 function adds another degree of freedom to the algorithm, which we may utilize to fine-tune it [13].

√ Algorithm after coarse timeout expiration

```
if (coarse timeout expires)
if (cwin>ssthresh) /* congestion avoid. */
ssthresh = f3(BWE*RTTmin);
if (ssthresh < 2)
ssthresh = 2;
cwin = 1;
else
cwin = f4(BWE*RTTmin);
endif
endif
if (cwin<ssthresh) /* slow start */
ssthresh = f5(BWE*RTTmin)
if (ssthresh < 2) ssthresh = 2;
cwin = 1;
else
cwin = f6(BWE*RTTmin)
endif
endif
endif
```

The algorithm's logic is straightforward once more. Following a timeout, the cwin and ssthresh are set using one of the functions fi, i=3,6 depending on the phase the algorithm is in at the time of the timeout. It's worth noting that employing the general functions fi, i=1,6 gives the algorithm six degrees of freedom to tweak. In the next sections, we'll look into and simulate a sampling of these functions, as well as provide default values.

4.4.3 TCP WESTWOOD: ALGORITHM IMPLEMENTATION

In this part, we show how to create a simple fi function implementation. $\surd$ Algorithm after 3 duplicate ACKS

```
if (3 DUPACKs are received)
    if (cwin<ssthresh) /* slow start */
        a = a + 0.25;
        if (a > 4)
            a = 4;
        endif
    endif
    if (cwin > ssthresh) /* congestion avoid. */
        a = 1;
    endif
    ssthresh = (BWE*RTTmin)/(pkt$_{s}ize * 8 * a$);
    reset cwin to ssthresh, if larger
    if (cwin > ssthresh)
        cwin = ssthresh;
    endif
endif
```

Inspection of the code reveals that f1 is simply chosen as an identity function during congestion avoidance, i.e. f1(x) = x. During a slow start, however, f2 is chosen as f2(x) = x=a. When 3 DUPACKs are received in slow start, an increases from 1 to 4 in 0.25 increments, however when 3 DUPACKs are received in congestion avoidance, an is set to 1. A is set to 1 during connection setup. The goal of the threshold reduction factor an is to prevent an overestimation of available bandwidth, which happens frequently during protracted periods of congestion. Indeed, the larger the reduction factor gets when a triple DUPACK is received during slow start (an signal that ssthresh was set too high). Following the same logic, if congestion is discovered during congestion avoidance, an is returned to 1: plainly, ssthresh was set correctly, and there is no need to lessen BWE's impact.

$\surd$Algorithm after coarse timeout expiration

```
if (coarse timeout expires)
    if (cwin > ssthresh) /* slow start */
        a = a + 1;
        if (a < 4)
            a = 4;
        endif
    endif
    if (cwin > ssthresh) /* congestion avoid. */
        a = 1;
    endif
    ssthresh = (BWE*RTTmin)/(pktsize*8*a);
    if (ssthresh < 2)
        ssthresh = 2;
    cwin = 1;
endif
```

endif

In this scenario, f3(x) = x is picked for the function f3, which is used to set ssthresh when a timeout occurs during congestion avoidance. f4(x) = 1 is the value of the function f4. When a timeout occurs during the slow start phase, the function f5 is chosen as f5(x) = x=a, where an increases from 1 to 4, in steps of 1 (as opposed to 0.25 in the triple DUPACK instance), and an is set to 1 when a timeout occurs in congestion avoidance. f6 has also been set to 1. After a timeout, the congestion window is reset to 1, like TCP Reno does. This option is cautious since it does not fully utilize the BWE information to prevent the congestion window from shrinking to 1 in the event of intermittent losses caused by wireless connection interference rather than congestion. This decision was made for a reason: fairness. We believe it is critical to keep TCP's cyclic nature when using drop-tail FIFO queuing, permitting traffic load oscillations on each TCP connection. Indeed, this behavior ensures that bandwidth resources are shared fairly amongst different connections bottle necked at the same FIFO queue without compromising the algorithm's stability. When network nodes implement RED or WRED, different values for cwin and ssthresh may be proposed after a timeout, however this is a topic for future research. New mechanisms can also be invented to transition between congestion avoidance and sluggish start, i.e., a technique to boost ssthresh, by employing a bandwidth estimation filter during network under-utilization.

4.4.4 Goals of Westwood Plus

We presented a new version of the TCP protocol in this task, with the goal of increasing its performance in the face of random or occasional losses. Simulated testing of the new version revealed a significant increase in goodput in almost all cases. Our changes can be seen as a step forward in the transition from TCP Tahoe to TCP Reno. TCP Tahoe was changed to TCP Reno by adding fast recovery, which allows the congestion window to be shrunk after three repeated ACKs. After a loss, TCP Tahoe resets cwin to one, while TCP RENO halves cwin after three duplicate ACKs. TCPWestwood now includes "faster" recovery to avoid over-shrinking cwin after three duplicate ACKs by taking into consideration TCP's end-to-end bandwidth estimation. As a result, the changes needed to implement TCP Westwood are similar to those made to shift from TCP Tahoe to TCP Reno. More work is being done, particularly in terms of compatibility with other TCP Tahoe or Reno connections. Furthermore, improvements to the bandwidth estimation method as well as various algorithm tuning factors are being investigated.

Chapter 5

Results and Discussion

5.1 Network Diagram

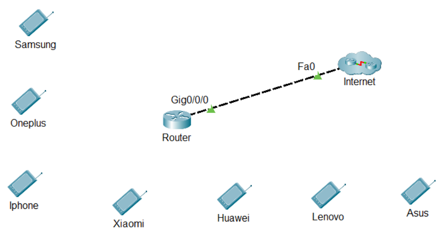


Figure 5.1: Network Topology

This network topology is designed to compare different variants of TCP, where end devices work as station nodes and the router works as an Access point node. The router is connected to the internet. So that the station devices or nodes can send or receive throughput over the router.

5.2 Simulation Parameters

NS3 or Network Simulator version 3 is software program which replicates the behaviour of real network. It is achieved by calculating the interaction by different network devices such as routes, switches, Nodes, Access Points, Links etc. There is no windows operating version of this software. It can be install in Linux operating system. There are lots of example in NS3 which can be used for different networks design and simulation. We choose wifi-tcp.cc file from NS3 examples. There are some basic details of this code. They are

1. There are two nodes n1 and n2 . N1 is access point node and N2 is station node.
2. There are some command values which can be changed.
3. It is 64 bit.

4. The counting time or simulation time is initiated with simulator time of NS3.
5. The wifi standard of this code is 802.11n which is a dual band standard. It can support both 5GHz and 2.4 GHz.

After the basic details, we learned about the command and command line arguments of the code. Some of the command are changeable, some are not. Now we will see the changeable commands of the code using terminal.

- **payloadSize:-** It determines how much data bits the protocol can carry.
- **datarate:-** It determines the rate of data.Usually we kept it 100Mbps.
- **tcpVariant:-** It determines the variant of TCP.
- **simulationTime:-** It is the running time of a simulation.
- **pcaptracing:-** It will generate packet capture file after a simulation.

Now we will see the command arguments that is not changeable through terminal. We can not change them. We can modify them as our needs. They are

- "Double Cur" here Cur is the variable name.
- "argc" it means arguments count.
- "argv" it means argument Vector.
- 'segmentsize" It is based on payloadSize.
- "Legacy channel" it means existing wifi channel.
- "Yanswifi" it means Yet another network simulator wifi. It is mainly used for giving name purpose.
- "ConstantRateWifi Manager" It means everyone will get equal data rate.
- "Constant Position Mobility" It means that the nodes are constant.
- "Sink" It means there are destination nodes and ports.
- "Wifi helper" It helps in setting a wifi for particular modem.
- "uin32" unsigned integer of 32 bits.
- "P2P Devices" it determines the ethernet ports

To run the code of NS3, we have to type some command lines in terminal. We can not open NS3 application by clicking any icon. We have to open the terminal and put some commands for accessing NS3.The commands are

- ./waf
- ./waf –run scratch/wifi-tcp
- ./waf –run "scratch/wifi-tcp–tcpvariant=TcpCubic

- ./waf –run"scratch/wifi-tcp–payloadSize=1472"
- ./waf –run"scratch/wifi-tcp–simulationTime=60"

After that, we will learn to modify the NS3 code to maximize the station point. We want to see the performance variation of different variants of Transmission control protocol. We have to change many things of the code to set the station points. We saw only one station node in this code. The procedure of increasing nodes are

- Creating the nodes.
- Creating the channels.
- Installing the channels on top of the node.
- Installing internet toots on top of the node.
- Allocating IP addresses with subnet musk.
- Creating interface for Ip addresses.
- Selecting port of the server.
- Installing server on top of the node.
- Specifying the the attributes of the clients.
- Installing the client app on node individually.

By following above instruction, we modified the code wifi-tcp.cc. Then, we got the variation of TCP variants. Before modifying, all the throughput of the variants were same except TCP Vegas. After increasing station point, we got the desire throughput.

5.3 Variant Performance Graphs

Number of Nodes	Cubic	Vegas	Westwood Plus	YEAH
1	52.8324	54.441	52.4974	52.8326
2	51.7418	44.7767	51.4566	51.6962
3	50.101	42.4786	49.6445	50.0853
4	48.6414	42.1528	48.5083	48.5709
5	48.1981	41.7053	48.1638	48.1646
6	47.3642	41.9106	47.2439	47.2539
7	46.9861	41.7012	46.6981	46.831

This is the average throughput data for different variants of TCP. We took throughput for seven different nodes. We plotted all four variants with seven nodes. The difference was quite good.

5.3.1 Simulation Result

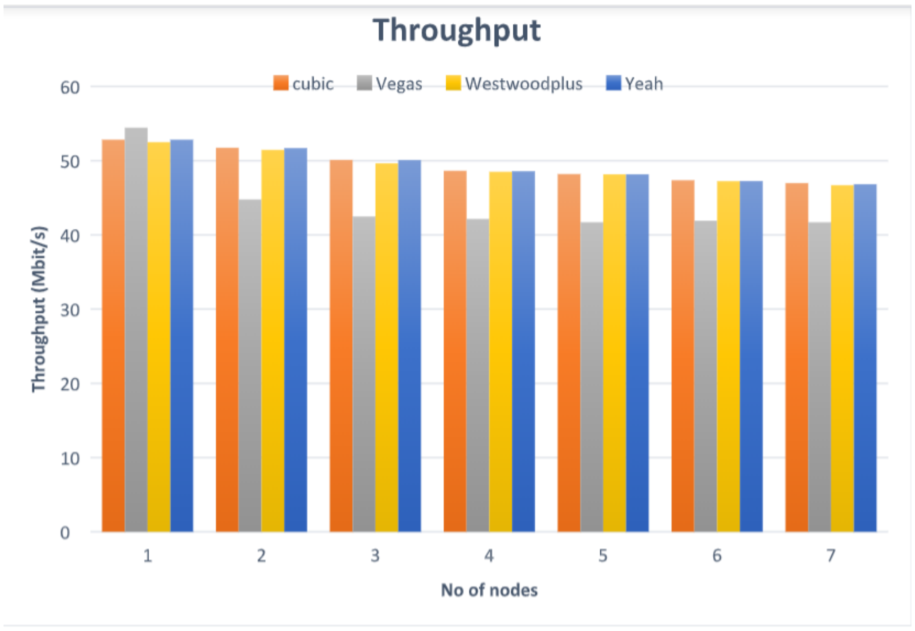


Figure 5.2: Node Vs Throughput

This is the throughput comparison graph for different variants of TCP for different station nodes. Here, we took throughput for every single node for four different variants.

5.3.2 TCP Vegas

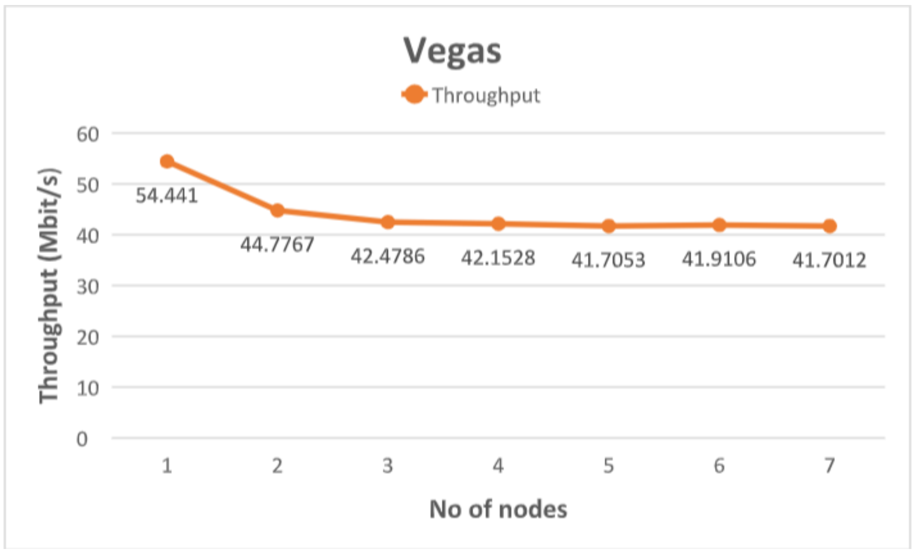


Figure 5.3: Node Vs Throughput

TCP Vegas maintains high throughput as node increases.It decreases slowly.After node 3, the throughput is almost constant.The simulation time was 60 seconds.

5.3.3 TCP YEAH

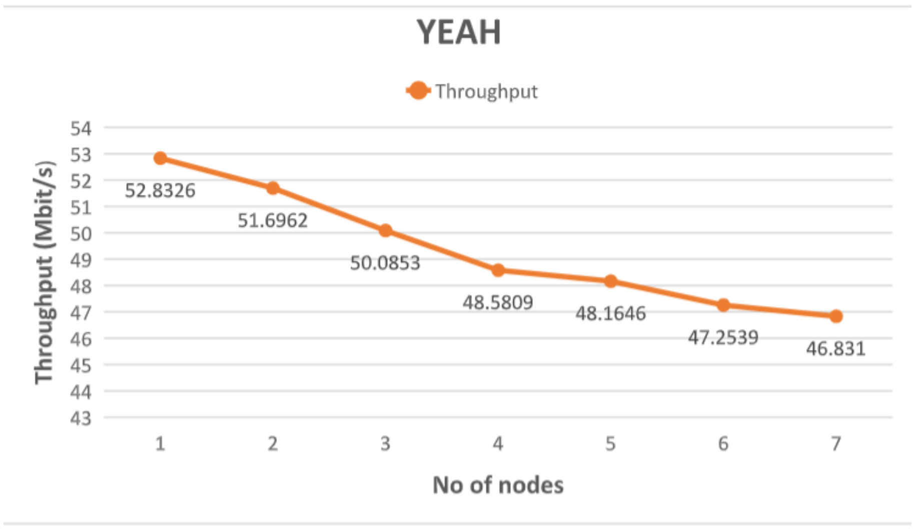


Figure 5.4: Node Vs Throughput

TCP Yeah shows high throughput for single node. As the node increase, the throughput decreases. But, It keeps high throughput compare to other variants. The simulation time was 60 Seconds.

5.3.4 TCP Westwood Plus

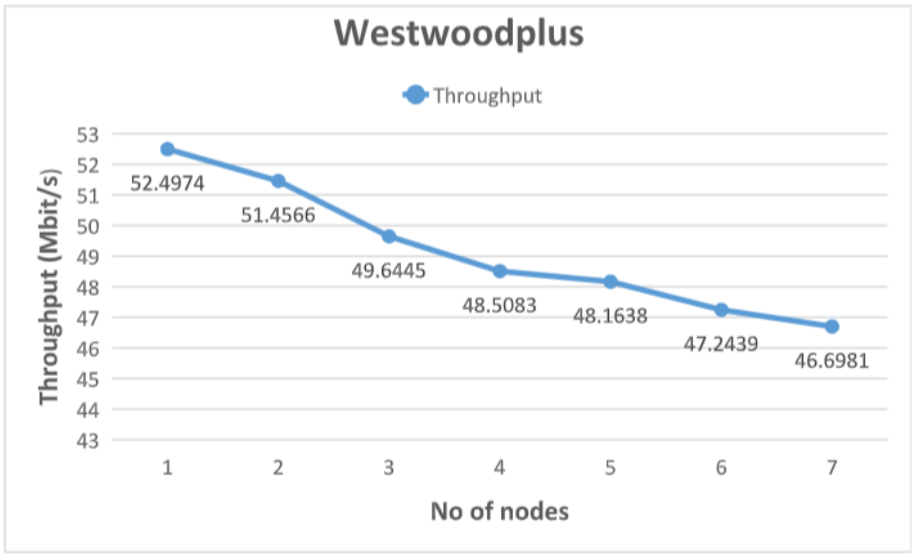


Figure 5.5: Node Vs Throughput

TCP Westwood Plus shows high throughput as the number of node decreases. The simulation time was 60 Seconds. It maintains high speed as the node increases. Though the throughput decrease as node increases, but not severely.

5.3.5 TCP Cubic

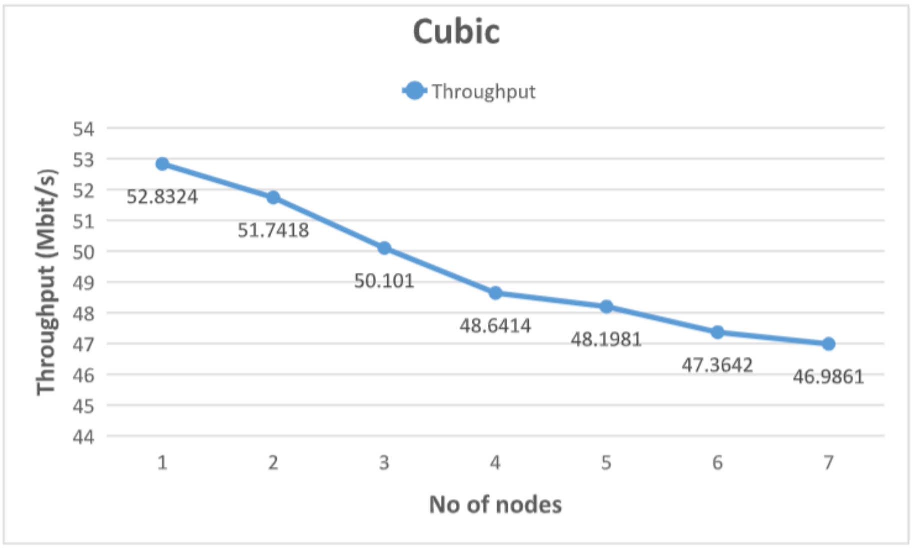


Figure 5.6: Node Vs Throughput

TCP Cubic is the default protocol of Linux operating system. Throughput is decreasing as node increasing. It keeps the high throughput value as node increases. The simulation time was 60 Seconds.

5.4 Discussion

The simulation findings do not demonstrate a clear winner among the methods studied. Because each tested context had its own set of unique characteristics, it was believed that various protocols would be better suited to dealing with diverse network situations. However, one apparent benefit of these findings is a decent sense of which methods are dominating.

BIC-TCP has been improved into CUBIC. It increases TCP friendliness and RTT fairness by simplifying BIC-TCP window management. In terms of the amount of time since the last loss occurrence, CUBIC employs a cubic rise function. When the cubic window growth function is slower than Standard TCP, CUBIC operates like Standard TCP to be fair. Furthermore, because of the protocol's real-time nature, the window expansion rate is unaffected by RTT, making it TCP-friendly on both short and long RTT channels. For each stationary node, TCP CUBIC works better. Though the difference in throughput among the variants is very little, still TCP CUBIC wins.

In TCP Vegas, the throughput is quite low compared to the other variants. But from our simulation result, we can observe that in spite of being lower throughput, with the increasing stationary nodes, the throughput did not drop much. In TCP Westwood Plus and TCP Yeah, the throughput decreases with the increasing stationary nodes quite sharply. We find that CUBIC addresses the limitations of existing TCPs and provides appropriate throughput for the data rate of 100Mbps.In future work, we may try to extend our research on establishing some algorithms which may increase the throughput of the variants.

Bibliography

[1] Ghassan A Abed, Mahamod Ismail, and Kasmiran Jumari. Characterization and observation of (transmission control protocol) tcp-vegas performance with different parameters over (long term evolution) lte networks. *Scientific Research and Essays*, 6(9):2003–2010, 2011.

[2] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. Yeah-tcp: yet another highspeed tcp. In *Proc. PFLDnet*, volume 7, pages 37–42, 2007.

[3] Steven M Bellovin. Security problems in the tcp/ip protocol suite. volume 19, pages 32–48. ACM New York, NY, USA, 1989.

[4] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, 1994.

[5] Lawrence S Brakmo and Larry L Peterson. Performance problems in bsd4. 4tcp. *ACM SIGCOMM Computer Communication Review*, 25(5):69–86, 1995.

[6] Han Cai, DY Eun, Sangtae Ha, Injong Rhee, and Lisong Xu. Stochastic ordering for internet congestion control and its applications. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 910–918. IEEE, 2007.

[7] Douglas E Comer. *Internetworking with TCP/IP*. Addison-Wesley Professional, 2013.

[8] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[9] Nelson E Hastings and Paul A McLean. Tcp/ip spoofing fundamentals. In *Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pages 218–224. IEEE, 1996.

[10] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

[11] Cheng Jin, David X Wei, and Steven H Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490–2501. IEEE, 2004.

[12] M Kalpana and T Purusothaman. Performance evaluation of exponential tcp/ip congestion control algorithm. *IJCSNS*, 9(3):312, 2009.

[13] Vasudev I Kanani and Mr Krunal J Panchal. Performance analyses of tcp westwood 1. 2014.

[14] Ehab A Khalil. Simulation-based comparisons of tcp congestion control. *International Journal of Advances in Engineering & Technology*, 4(2):84, 2012.

[15] Bruno YL Kimura, Demetrius CSF Lima, and Antonio AF Loureiro. Packet scheduling in multipath tcp: Fundamentals, lessons, and opportunities. *IEEE Systems Journal*, 15(1):1445–1457, 2020.

[16] Richard J La, Jeonghoon Mo, Jean Walrand, and Venkat Anantharam. A case for tcp vegas and gateways using game theoretic approach, 1998.

[17] Richard J La, Jean Walrand, and Venkatachalam Anantharam. *Issues in TCP vegas.* Citeseer, 1999.

[18] Peter Loshin. *TCP/IP clearly explained.* Elsevier, 2003.

[19] Jeonghoon Mo, Richard J La, Venkat Anantharam, and Jean Walrand. Analysis and comparison of tcp reno and vegas. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 3, pages 1556–1563. IEEE, 1999.

[20] Esmond Pitt. *Fundamental Networking in Java.* Springer Science & Business Media, 2005.

[21] Pasi Sarolahti and Alexey Kuznetsov. Congestion control in linux tcp. In *USENIX Annual Technical Conference, FREENIX Track*, pages 49–62, 2002.

[22] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings-IEEE INFOCOM*, 2006.

[23] IETF TSVWG. Highspeed tcp for large congestion windows. 2003.

[24] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. *ACM SIGCOMM Computer Communication Review*, 24(5):45–60, 1994.

[25] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM transactions on Networking*, 14(6):1246–1259, 2006.

[26] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM transactions on Networking*, 14(6):1246–1259, 2006.