**PROJECT REPORT ON**

**Exploring and Comparing the Various Performance Measures and the Congestion Control Mechanisms of TCP Variants Over Various Networks**

Submitted in partial fulfilment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**Submitted by**

123003004 – P A ADHITYA NARAYAN – CSE



**Under the Guidance of**

**Prof. Sasikala Devi. N**

**School of Computing**

**SASTRA DEEMED TO BE UNIVERSITY**

(A University established under section 3 of the UGC Act, 1956)

Tirumalaisamudram

Thanjavur – 613401

December (2021)

**SHANMUGHA**

**ARTS, SCIENCE, TECHNOLOGY & RESEARCH ACADEMY**

**(SASTRA DEEMED TO BE UNIVERSITY)**

**(A University Established under section 3 of the UGC Act, 1956)**

**TIRUMALAISAMUDRAM, THANJAVUR – 613401**

****

**BONAFIDE CERTIFICATE**

Certified that this project work entitled “**Exploring and Comparing the Various Performance Measures and the Congestion Control Mechanisms of TCP Variants over Various Networks**” submitted to the Shanmugha Arts, Science, Technology, & Research Academy (SASTRA Deemed to be University), Tirumalaisamudram – 613401 by P A Adhitya Narayan (123003004), CSE in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in their respective programme. This work is an original and independent work carried out under my guidance, during the period August 2021 – December 2021**.**

**Prof. Sasikala Devi. N ASSOCIATE DEAN**

**SCHOOL OF COMPUTING**

Submitted for Project Viva Voce held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Examiner – I Examiner - II**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| S. NO | TOPIC | Page  No. |
| 1 | ACKNOWLEDGEMENTS | 4 |
| 2 | ABSTRACT | 5 |
| 3 | CHAPTER 1 INTRODUCTION | 7 |
| 5 | CHAPTER 2 ELABORATE DETAILS ON THE PROJECT | 9 |
| 6 | CHAPTER 3 SOURCE CODE | 15 |
| 7 | CHAPTER 4 RESULTS | 34 |
| 8 | CHAPTER 5 PERFORMANCE EVALUATION | 40 |
| 10 | CHAPTER 6 REFERENCES | 45 |

**ACKNOWLEDGEMENTS**

First of all, I would like to thank God Almighty for his endless blessings.

I would like to express my sincere gratitude to **Dr S. Vaidyasubramaniam, Vice – Chancellor** for his encouragement during the span of my academic life at SASTRA Deemed University.

I would forever remain grateful and I would like to thank **Dr A. Umamakeswari, Dean, School of Computing and R. Chandramouli**, **Registrar** for their overwhelming support provided during my course span in SASTRA Deemed University.

I am extremely grateful to **Dr Shankar Sriram, Associate Dean, School of Computing** for his constant support, motivation and academic help extended for the past three years of my life in School of Computing.

I would specifically thank and express my gratitude to **N. Sasikala Devi, Associate Professor, School of Computing** for providing me an opportunity to do this project and for her guidance and support to successfully complete the project.

I also thank all the Teaching and Non-Teaching faculty, and all other people who have directly or indirectly helped me through their support, encouragement and all other assistance extended for completion of my project and for successful completion of all courses during my academic life at SASTRA Deemed University.

Finally, I thank my parents and all others who have helped me acquire this interest in project and aided me in completing it within the deadline without much struggle.

**ABSTRACT**

The TCP (Transmission Control Protocol) is a transport layer protocol, and a part of the Internet Protocol suite. It promises a reliable connection between network nodes, but packet loss is an integral part of any networking implementation, which primarily happens due to congestion. Even though TCP contains several mechanisms to maintain reliability such as slow start, congestion avoidance, fast retransmit, and fast recovery, it has not been up to the mark with environments like high-speed communication and communication over different media. Thus, we do further analysis and development of congestion control algorithms. In this paper, we have explored the reliability and robustness of TCP variants (Tahoe, Reno, Vegas) based on different parameters over various networks. We also have compared the different congestion control and avoidance mechanisms of these variants to show how they affect the throughput and efficiency of different network environments. Ns2 software was used for simulating the various networks with the various TCP variants. based on the simulation results, we have analysed the performance of the various TCP variants based on factors like bandwidth, delay, throughput, etc.

**KEY WORDS:**  TCP, Tahoe, Reno, Vegas, Congestion, Congestion Window, Fast-Recovery, Fast-Retransmit, Reliability, Slow Start, TCP variants, Bandwidth, Delay, Throughput.

**ABBREVIATIONS**

TCP Transmission Control Protocol

PDR Packet Delivery Ratio

RTT Round Trip Time

MSS Maximum Segment Size

**CHAPTER 1: INTRODUCTION**

We have seen that TCP is an application layer protocol, which provides a reliable connection between nodes. We are going to see about various variants of TCP and compare them according to their performance.

The various performance measures that we are going to use to compare the variants are:

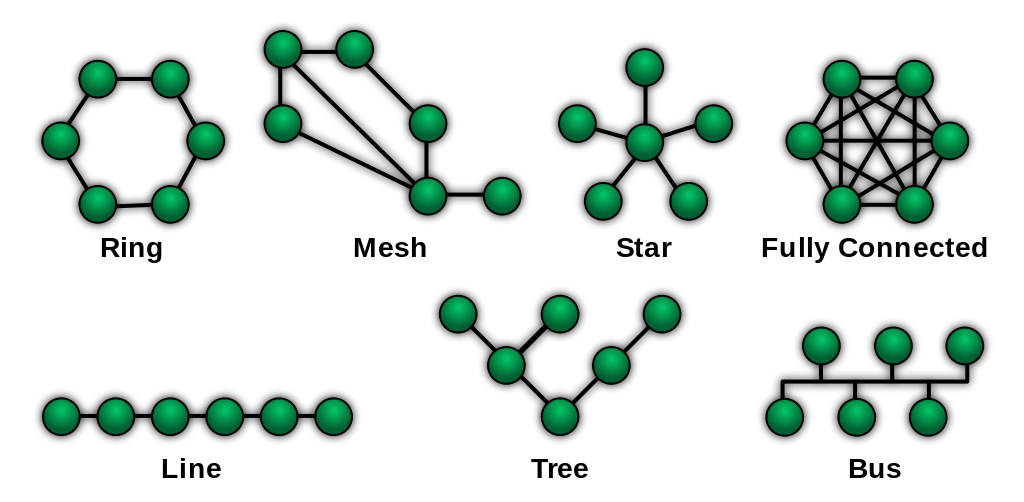
* Packet Delivery Ratio (PDR): Defined as the ratio of number of packets delivered in total to the total number of packets sent from source node to destination node in the network. (Expressed as a percentage)
* Delay: Specifies the latency for a bit of data to travel across the network from source to destination. There are four types of delays – processing delay, queuing delay, transmission delay, and propagation delay. The overall delay is the sum of these four delays.
* Throughput: It is a measure of how many units of information a system can process in a given amount of time. Typically measures in bits per second (bps), megabits per second (Mbps), or gigabits per second (Gbps).

Packet loss in a network generally occurs due to congestion. Congestion is defined as the reduced quality of service that occurs when a network node or link is carrying more data than it can handle. Lack of sufficient bandwidth, as we will see later, seems to be the main cause of this congestion.

Fortunately, TCP comes with a few congestion control techniques to minimize delays and packet loss. The congestion control scheme of TCP is as follows:

* Slow start phase: In this phase, after every RTT the congestion window size increments exponentially.
* Congestion avoidance phase: This phase starts after the threshold value also denoted a *ssthresh*. The size of the congestion window increases additively.
* Congestion Detection phase: If congestion occurs, the congestion window size is decreased. Occurrence of a congestion can be found out only by a need of retransmission. Retransmission can occur in one of two cases: when the RTO timer times out or when three duplicate ACKs are received. There are separate ways to deal with these two cases as we will see later.

In this project we have compared the various congestion control techniques and performance of the various variants of TCP across various wired network topologies, namely – Dumbbell, Star, Ring, and Mesh.

Fig. 1.1. Various Topologies

We have used Ns2 as the simulation tool for this project.

**CHAPTER 2: ELABORATE DETAILS ON THE PROJECT**

As mentioned in the CHAPTER 1: INTRODUCTION we are going to compare the variants of TCP along the various performance measures like

* Packet Delivery Ratio (PDR):

(No. of packets delivered/ No. of packets sent) x 100 %

* Delay: Delay comes in four types:
* Propagation delay (dp): It is the time that it takes for a bit to reach from one end of a link to the other. The delay depends on the separation (S) between the sender and the receiver, and also the propagation speed (V) of the signal. It is calculated as:

dp = S / V

* Transmission delay (dt): Refers to the time it takes to transmit a data packet onto the outgoing link. The delay is determined by the size of the packet and the bandwidth of the outgoing link. If a packet consists of T bits and the link has a bandwidth of B bits per second, then the transmission delay is equal to:

dt = T / B

* Queuing delay (dq): Refers to the time that a packet waits to be processed in the buffer of a node. The delay depends on the rate of arrival of the incoming packets, and the transmission capacity of the outgoing link, and the nature of the network’s traffic.
* Processing delay (dpr): Time taken by a switch to process the packet header. The delay depends on the processing speed of the switch.

Total Delay (dtot):

dtot = dp + dt + dq + dpr

* Throughput: It is the rate at which data is processed and transferred from the source node to the destination node in a network. It is usually measured in bps, Mbps, or Gbps.

Now let us see about the congestion control techniques in general, and then we will see about the TCP variants in specific:

TCP responds to congestion by reducing the sender window size, the amount of reduction being determined by the following two parameters:

* Receiver window size: Sender should not send data greater than receiver window size. Otherwise, it leads to dropping of TCP segments, which causes retransmission. Receiver dictates its window size to the sender through TCP header in the packet.
* Congestion Window size: Sender window should not send data greater than congestion window size. Otherwise, it leads to dropping of TCP segments which causes retransmission. Different variants of TCP incorporate different ways to calculate the size of the congestion window. Congestion window is known only to the sender node and is not sent over the network.

So,

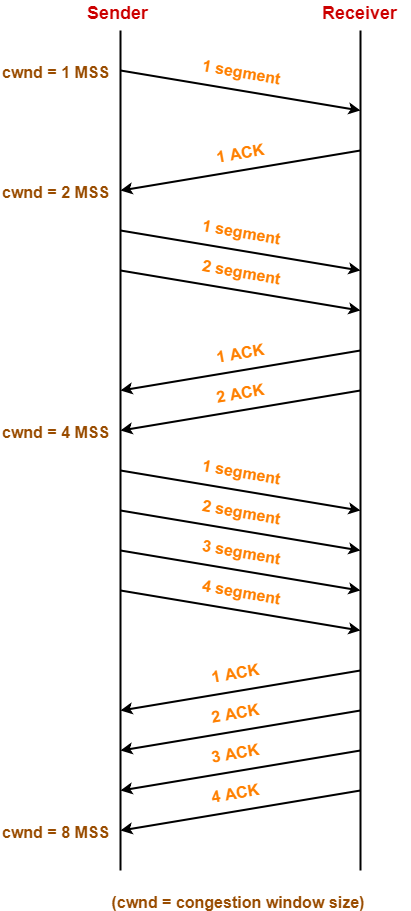
Sender window size = Min (Receiver window size, Congestion window size)

TCP’s generalised policy for taking care of congestion consists of the following three phases:

* Slow Start Phase: Initially, sender sets the congestion window size as 1 MSS (Maximum Segment Size). After receiving each acknowledgement, sender increases the congestion window size by 1 MSS. In this phase, the congestion window size increases exponentially.

*cwnd = cwnd + MSS*

This is shown in the following picture:

Fig. 2.1. Slow Start Phase

After 1 RTT, cwnd = (2)1 = 2 MSS

After 2 RTT, cwnd = (2)2 = 4 MSS

After 3 RTT, cwnd = (2)3 = 8 MSS and so on...

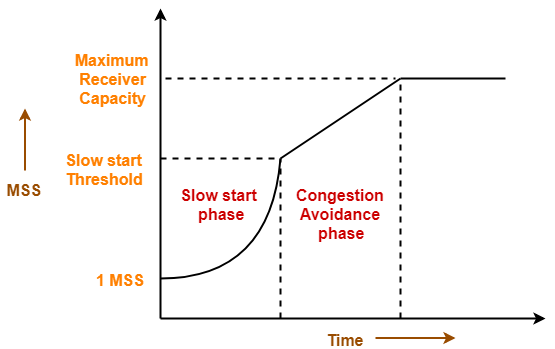
This phase remains into play until the congestion window size reaches the slow start threshold. Threshold is defined as the half of maximum number of TCP segments that the receiver window can accommodate.

*ssthresh = (Receiver window size/ MSS) / 2*

* Congestion Avoidance Phase: After reaching the threshold, the sender increases the congestion window size linearly. On receiving each acknowledgement, sender increases the congestion window size by 1.

*cwnd = cwnd + 1*

This phase remains active until the congestion window size becomes equal to the receiver window size.

Fig. 2.2. Phases of congestion control

* Congestion Detection Phase: When sender detects the loss of segments, it reacts in different ways depending on how the loss is detected:
* Detection on timeout: Timeout Timer expires before receiving the acknowledgement for a segment. This case indicates a stronger chance of congestion in the network.

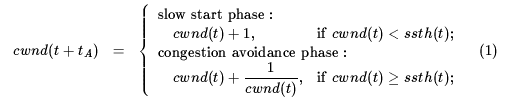
In this case, the sender reacts by setting the slow start threshold to half of the current congestion window size, decreasing the congestion window size to 1 MSS, and resuming the slow – start phase.

* Detection on receiving 3 Duplicate Acknowledgements: Sender receives 3 duplicate acknowledgements for a segment. This case suggests a weaker possibility of congestion in the network.

In this case, the sender reacts by assigning the slow start threshold to half of the present congestion window size, decreasing the congestion window size to slow start threshold, and continuing the congestion avoidance phase.

Now let us look at the TCP variants and their congestion control mechanisms in detail.

* **TCP Reno**: In TCP Reno, the window size cyclically changes in a typical situation. The window size continuously increases until packet loss occurs. TCP Reno consists of two phases in increasing its *cwnd* size: slow start phase and congestion avoidance phase. When an ACK (acknowledgement) packet is received by TCP at the sender side at time t + tA, the current window size *cwnd (t + tA)* is updated from *cwnd(t)* as follows:



where *ssth(t)* is a threshold value at which TCP changes its phase from slow start phase to congestion avoidance phase. When packet loss is detected by retransmission timeout expiration, the congestion windowand the threshold are updated as:



On the other hand, when TCP detects packet loss by a fast retransmit algorithm, it changes *cwnd(t)* and *ssth(t)* as:



TCP Reno then commences into a fast recovery phase if the packet loss is found by the fast retransmit way. In this phase, the window size is increased by one when a duplicate acknowledgement is received. On the other hand, the congestion window is restored to the threshold when the non-duplicate acknowledgement packet corresponding to the retransmitted packet is received.

* **TCP Vegas**: TCP Vegas controls its window size by observing RTTs of packets that the sender host has sent before. If observed RTTs become large, TCP Vegas senses that the network starts to be congested, and throttles the window size. If RTTs become small, the sender of TCP Vegas determines that the network is free from the congestion, and increases the *cwnd* size again. In the congestion avoidance phase, the *cwnd* size is updated as:

****

where *rtt* is an observed round-trip time, *base\_rtt* is the minimum value of observed RTTs, and α and b are some constant values.

TCP Vegas has another add-on in its congestion control algorithm: a slow slow-start phase. The rate of increasing its window size in slow start phase is one half of that in TCP Tahoe and TC Reno. Specifically, the *cwnd* size is incremented every other time an acknowledgement packet is received. Note that the congestion control mechanism used by TCP Vegas indicates that if observed RTTs of the packets are identical, the window size remains unchanged.

**CHAPTER 3: SOURCE CODE**

**tcptahoedumbbell.tcl :**

#Simulation end time

set val(stop) 20.0

#Create a ns simulator

set ns [new Simulator]

#Coloe codes

$ns color 1 Green

$ns color 2 Blue

$ns color 3 Red

$ns color 4 Yellow

#Open the NS trace file

set tracefile [open tcptahoedumbelltrace.tr w]

$ns trace-all $tracefile

#Open the NAM trace file

set namfile [open tcptahoedumbellnam.nam w]

$ns namtrace-all $namfile

#create nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

$n3 shape box

set n4 [$ns node]

$n4 shape box

set n5 [$ns node]

set n6 [$ns node]

set n7 [$ns node]

set n8 [$ns node]

set n9 [$ns node]

#create links between the nodes

$ns duplex-link $n3 $n4 100.0Mb 50ms DropTail

$ns queue-limit $n3 $n4 10

$ns duplex-link $n0 $n3 1.0Mb 50ms DropTail

$ns queue-limit $n0 $n3 50

$ns duplex-link $n1 $n3 1.0Mb 50ms DropTail

$ns queue-limit $n1 $n3 50

$ns duplex-link $n2 $n3 1.0Mb 50ms DropTail

$ns queue-limit $n2 $n3 50

$ns duplex-link $n4 $n6 0.2Mb 200ms DropTail

$ns queue-limit $n4 $n6 5

$ns duplex-link $n4 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n4 $n5 5

$ns duplex-link $n4 $n7 0.2Mb 200ms DropTail

$ns queue-limit $n4 $n7 5

$ns duplex-link $n8 $n3 1.0Mb 50ms DropTail

$ns queue-limit $n8 $n3 50

$ns duplex-link $n4 $n9 0.2Mb 200ms DropTail

$ns queue-limit $n4 $n9 5

#Give node position for NAM

$ns duplex-link-op $n3 $n4 orient right

$ns duplex-link-op $n4 $n6 orient right

$ns duplex-link-op $n4 $n5 orient right-up

$ns duplex-link-op $n0 $n3 orient right-down

$ns duplex-link-op $n1 $n3 orient right

$ns duplex-link-op $n2 $n3 orient right-up

$ns duplex-link-op $n4 $n7 orient right-down

$ns duplex-link-op $n8 $n3 orient right-up

$ns duplex-link-op $n4 $n9 orient right-down

#Setting nodes physical properties

$n0 color "green"

$n5 color "green"

$n1 color "blue"

$n6 color "blue"

$n2 color "red"

$n7 color "red"

$n8 color "yellow"

$n9 color "yellow"

#Setup TCP connections

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n5 $sink0

$ns connect $tcp0 $sink0

$tcp0 set packetSize\_ 1500

$tcp0 set fid\_ 1

set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n6 $sink1

$ns connect $tcp1 $sink1

$tcp1 set fid\_ 2

set tcp2 [new Agent/TCP]

$ns attach-agent $n2 $tcp2

set sink2 [new Agent/TCPSink]

$ns attach-agent $n7 $sink2

$ns connect $tcp2 $sink2

$tcp2 set fid\_ 3

set tcp3 [new Agent/TCP]

$ns attach-agent $n8 $tcp3

set sink3 [new Agent/TCPSink]

$ns attach-agent $n9 $sink3

$ns connect $tcp3 $sink3

$tcp3 set fid\_ 4

#setup Application over TCP connection

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp1

set smtp0 [new Application/Traffic/Exponential]

$smtp0 attach-agent $tcp2

$smtp0 set packetSize\_ 210

$smtp0 set burst\_time\_ 50ms

$smtp0 set idle\_time\_ 50ms

$smtp0 set rate\_ 100k

set http0 [new Application/Traffic/Exponential]

$http0 attach-agent $tcp0

$http0 set packetSize\_ 210

$http0 set burst\_time\_ 50ms

$http0 set idle\_time\_ 50ms

$http0 set rate\_ 100k

set cbr0 [new Application/Traffic/CBR]

$cbr0 attach-agent $tcp3

$cbr0 set packetSize\_ 210

$cbr0 set burst\_time\_ 50ms

$cbr0 set idle\_time\_ 50ms

$cbr0 set rate\_ 100k

proc finish {} {

global ns tracefile namfile

$ns flush-trace

close $tracefile

close $namfile

exec nam tcptahoedumbellnam.nam &

exit 0

}

proc record {} {

set ns [new Simulator instance]

set time 0.1

set nowtime [$ns now]

$ns at [expr $nowtime + $time] "record"

}

#Scheduling the events

$ns at 0.1 "$ftp0 start"

$ns at 0.1 "$smtp0 start"

$ns at 0.1 "$http0 start"

$ns at 0.1 "$cbr0 start"

$ns at 19.9 "$ftp0 stop"

$ns at 19.9 "$smtp0 stop"

$ns at 19.9 "$http0 stop"

$ns at 19.9 "$cbr0 stop"

$ns at 0.0 "record"

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "finish"

$ns at $val(stop) "puts \"done\" ; $ns halt"

#Run the simulator

$ns run

**tcptahoestar.tcl:**

#Simulation end time

set val(stop) 20.0

#Create a ns simulator

set ns [new Simulator]

#Coloe codes

$ns color 1 Green

$ns color 2 Blue

$ns color 3 Red

$ns color 4 Yellow

#Open the NS trace file

set tracefile [open tcptahoestartrace.tr w]

$ns trace-all $tracefile

#Open the NAM trace file

set namfile [open tcptahoestarnam.nam w]

$ns namtrace-all $namfile

#create nodes

set n0 [$ns node]

$n0 shape box

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

set n6 [$ns node]

set n7 [$ns node]

set n8 [$ns node]

#create links between the nodes

$ns duplex-link $n0 $n1 1.0Mb 50ms DropTail

$ns queue-limit $n0 $n1 50

$ns duplex-link $n0 $n2 1.0Mb 50ms DropTail

$ns queue-limit $n0 $n2 50

$ns duplex-link $n0 $n3 1.0Mb 50ms DropTail

$ns queue-limit $n0 $n3 50

$ns duplex-link $n0 $n4 1.0Mb 50ms DropTail

$ns queue-limit $n0 $n4 50

$ns duplex-link $n0 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n0 $n5 5

$ns duplex-link $n0 $n6 0.2Mb 200ms DropTail

$ns queue-limit $n0 $n6 5

$ns duplex-link $n0 $n7 0.2Mb 200ms DropTail

$ns queue-limit $n0 $n7 5

$ns duplex-link $n0 $n8 0.2Mb 200ms DropTail

$ns queue-limit $n0 $n8 5

#Give node position for NAM

$ns duplex-link-op $n0 $n1 orient up

$ns duplex-link-op $n0 $n2 orient right-up

$ns duplex-link-op $n0 $n3 orient right

$ns duplex-link-op $n0 $n4 orient right-down

$ns duplex-link-op $n0 $n5 orient down

$ns duplex-link-op $n0 $n6 orient left-down

$ns duplex-link-op $n0 $n7 orient left

$ns duplex-link-op $n0 $n8 orient left-up

#Setting nodes physical properties

$n1 color "green"

$n5 color "green"

$n2 color "blue"

$n6 color "blue"

$n3 color "red"

$n7 color "red"

$n4 color "yellow"

$n8 color "yellow"

#Setup TCP connections

set tcp0 [new Agent/TCP]

$ns attach-agent $n1 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n5 $sink0

$ns connect $tcp0 $sink0

$tcp0 set packetSize\_ 1500

$tcp0 set fid\_ 1

set tcp1 [new Agent/TCP]

$ns attach-agent $n2 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n6 $sink1

$ns connect $tcp1 $sink1

$tcp1 set fid\_ 2

set tcp2 [new Agent/TCP]

$ns attach-agent $n3 $tcp2

set sink2 [new Agent/TCPSink]

$ns attach-agent $n7 $sink2

$ns connect $tcp2 $sink2

$tcp2 set fid\_ 3

set tcp3 [new Agent/TCP]

$ns attach-agent $n4 $tcp3

set sink3 [new Agent/TCPSink]

$ns attach-agent $n8 $sink3

$ns connect $tcp3 $sink3

$tcp3 set fid\_ 4

#setup Application over TCP connection

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

set smtp0 [new Application/Traffic/Exponential]

$smtp0 attach-agent $tcp1

$smtp0 set packetSize\_ 210

$smtp0 set burst\_time\_ 50ms

$smtp0 set idle\_time\_ 50ms

$smtp0 set rate\_ 100k

set http0 [new Application/Traffic/Exponential]

$http0 attach-agent $tcp2

$http0 set packetSize\_ 210

$http0 set burst\_time\_ 50ms

$http0 set idle\_time\_ 50ms

$http0 set rate\_ 100k

set cbr0 [new Application/Traffic/CBR]

$cbr0 attach-agent $tcp3

$cbr0 set packetSize\_ 210

$cbr0 set burst\_time\_ 50ms

$cbr0 set idle\_time\_ 50ms

$cbr0 set rate\_ 100k

proc finish {} {

global ns tracefile namfile

$ns flush-trace

close $tracefile

close $namfile

exec nam tcptahoestarnam.nam &

exit 0

}

proc record {} {

set ns [new Simulator instance]

set time 0.1

set nowtime [$ns now]

$ns at [expr $nowtime + $time] "record"

}

#Scheduling the events

$ns at 0.1 "$ftp0 start"

$ns at 0.1 "$smtp0 start"

$ns at 0.1 "$http0 start"

$ns at 0.1 "$cbr0 start"

$ns at 19.9 "$ftp0 stop"

$ns at 19.9 "$smtp0 stop"

$ns at 19.9 "$http0 stop"

$ns at 19.9 "$cbr0 stop"

$ns at 0.0 "record"

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "finish"

$ns at $val(stop) "puts \"done\" ; $ns halt"

#Run the simulator

$ns run

**tcptahoering.tcl:**

#Simulation end time

set val(stop) 20.0

#Create a ns simulator

set ns [new Simulator]

#Coloe codes

$ns color 1 Green

$ns color 2 Blue

$ns color 3 Red

$ns color 4 Yellow

#Open the NS trace file

set tracefile [open tcptahoeringtrace.tr w]

$ns trace-all $tracefile

#Open the NAM trace file

set namfile [open tcptahoeringnam.nam w]

$ns namtrace-all $namfile

#create nodes

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

set n6 [$ns node]

set n7 [$ns node]

set n8 [$ns node]

#create links between the nodes

$ns duplex-link $n1 $n2 1.0Mb 50ms DropTail

$ns queue-limit $n1 $n2 50

$ns duplex-link $n2 $n3 0.2Mb 200ms DropTail

$ns queue-limit $n2 $n3 5

$ns duplex-link $n3 $n4 1.0Mb 50ms DropTail

$ns queue-limit $n3 $n4 50

$ns duplex-link $n4 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n4 $n5 5

$ns duplex-link $n5 $n6 1.0Mb 50ms DropTail

$ns queue-limit $n5 $n6 50

$ns duplex-link $n6 $n7 0.2Mb 200ms DropTail

$ns queue-limit $n6 $n7 5

$ns duplex-link $n7 $n8 1.0Mb 50ms DropTail

$ns queue-limit $n7 $n8 50

$ns duplex-link $n8 $n1 0.2Mb 200ms DropTail

$ns queue-limit $n8 $n1 5

#Give node position for NAM

$ns duplex-link-op $n1 $n2 orient right

$ns duplex-link-op $n2 $n3 orient right-down

$ns duplex-link-op $n3 $n4 orient down

$ns duplex-link-op $n4 $n5 orient left-down

$ns duplex-link-op $n5 $n6 orient left

$ns duplex-link-op $n6 $n7 orient left-up

$ns duplex-link-op $n7 $n8 orient up

$ns duplex-link-op $n8 $n1 orient right-up

#Setting nodes physical properties

$n1 color "green"

$n5 color "green"

$n2 color "blue"

$n6 color "blue"

$n3 color "red"

$n7 color "red"

$n4 color "yellow"

$n8 color "yellow"

#Setup TCP connections

set tcp0 [new Agent/TCP]

$ns attach-agent $n1 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n5 $sink0

$ns connect $tcp0 $sink0

$tcp0 set packetSize\_ 1500

$tcp0 set fid\_ 1

set tcp1 [new Agent/TCP]

$ns attach-agent $n2 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n6 $sink1

$ns connect $tcp1 $sink1

$tcp1 set fid\_ 2

set tcp2 [new Agent/TCP]

$ns attach-agent $n3 $tcp2

set sink2 [new Agent/TCPSink]

$ns attach-agent $n7 $sink2

$ns connect $tcp2 $sink2

$tcp2 set fid\_ 3

set tcp3 [new Agent/TCP]

$ns attach-agent $n4 $tcp3

set sink3 [new Agent/TCPSink]

$ns attach-agent $n8 $sink3

$ns connect $tcp3 $sink3

$tcp3 set fid\_ 4

#setup Application over TCP connection

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

set smtp0 [new Application/Traffic/Exponential]

$smtp0 attach-agent $tcp1

$smtp0 set packetSize\_ 210

$smtp0 set burst\_time\_ 50ms

$smtp0 set idle\_time\_ 50ms

$smtp0 set rate\_ 100k

set http0 [new Application/Traffic/Exponential]

$http0 attach-agent $tcp2

$http0 set packetSize\_ 210

$http0 set burst\_time\_ 50ms

$http0 set idle\_time\_ 50ms

$http0 set rate\_ 100k

set cbr0 [new Application/Traffic/CBR]

$cbr0 attach-agent $tcp3

$cbr0 set packetSize\_ 210

$cbr0 set burst\_time\_ 50ms

$cbr0 set idle\_time\_ 50ms

$cbr0 set rate\_ 100k

proc finish {} {

global ns tracefile namfile

$ns flush-trace

close $tracefile

close $namfile

exec nam tcptahoeringnam.nam &

exit 0

}

proc record {} {

set ns [new Simulator instance]

set time 0.1

set nowtime [$ns now]

$ns at [expr $nowtime + $time] "record"

}

#Scheduling the events

$ns at 0.1 "$ftp0 start"

$ns at 0.1 "$smtp0 start"

$ns at 0.1 "$http0 start"

$ns at 0.1 "$cbr0 start"

$ns at 19.9 "$ftp0 stop"

$ns at 19.9 "$smtp0 stop"

$ns at 19.9 "$http0 stop"

$ns at 19.9 "$cbr0 stop"

$ns at 0.0 "record"

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "finish"

$ns at $val(stop) "puts \"done\" ; $ns halt"

#Run the simulator

$ns run

**tcptahoemesh.tcl:**

#Simulation end time

set val(stop) 20.0

#Create a ns simulator

set ns [new Simulator]

#Coloe codes

$ns color 1 Green

$ns color 2 Blue

$ns color 3 Red

$ns color 4 Yellow

#Open the NS trace file

set tracefile [open tcptahoemeshtrace.tr w]

$ns trace-all $tracefile

#Open the NAM trace file

set namfile [open tcptahoemeshnam.nam w]

$ns namtrace-all $namfile

#create nodes

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

set n6 [$ns node]

set n7 [$ns node]

set n8 [$ns node]

set n9 [$ns node]

#create links between the nodes

$ns duplex-link $n1 $n2 1.0Mb 50ms DropTail

$ns queue-limit $n1 $n2 50

$ns duplex-link $n2 $n3 1.0Mb 50ms DropTail

$ns queue-limit $n2 $n3 50

$ns duplex-link $n3 $n6 5.0Mb 100ms DropTail

$ns queue-limit $n3 $n6 25

$ns duplex-link $n6 $n9 5.0Mb 100ms DropTail

$ns queue-limit $n6 $n9 25

$ns duplex-link $n9 $n8 1.0Mb 50ms DropTail

$ns queue-limit $n9 $n8 50

$ns duplex-link $n8 $n7 1.0Mb 50ms DropTail

$ns queue-limit $n8 $n7 50

$ns duplex-link $n7 $n4 5.0Mb 100ms DropTail

$ns queue-limit $n7 $n4 25

$ns duplex-link $n4 $n1 5.0Mb 100ms DropTail

$ns queue-limit $n4 $n1 25

$ns duplex-link $n1 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n1 $n5 5

$ns duplex-link $n7 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n7 $n5 5

$ns duplex-link $n3 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n3 $n5 5

$ns duplex-link $n9 $n5 0.2Mb 200ms DropTail

$ns queue-limit $n9 $n5 5

#Give node position for NAM

$ns duplex-link-op $n1 $n2 orient right

$ns duplex-link-op $n1 $n5 orient right-down

$ns duplex-link-op $n2 $n3 orient right

$ns duplex-link-op $n3 $n6 orient down

$ns duplex-link-op $n6 $n9 orient down

$ns duplex-link-op $n9 $n8 orient left

$ns duplex-link-op $n8 $n7 orient left

$ns duplex-link-op $n7 $n4 orient up

$ns duplex-link-op $n4 $n1 orient up

#Setting nodes physical properties

$n1 color "green"

$n9 color "green"

$n2 color "blue"

$n8 color "blue"

$n3 color "red"

$n7 color "red"

$n4 color "yellow"

$n6 color "yellow"

#Setup TCP connections

set tcp0 [new Agent/TCP]

$ns attach-agent $n1 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n9 $sink0

$ns connect $tcp0 $sink0

$tcp0 set packetSize\_ 1500

$tcp0 set fid\_ 1

set tcp1 [new Agent/TCP]

$ns attach-agent $n2 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n8 $sink1

$ns connect $tcp1 $sink1

$tcp1 set fid\_ 2

set tcp2 [new Agent/TCP]

$ns attach-agent $n3 $tcp2

set sink2 [new Agent/TCPSink]

$ns attach-agent $n7 $sink2

$ns connect $tcp2 $sink2

$tcp2 set fid\_ 3

set tcp3 [new Agent/TCP]

$ns attach-agent $n4 $tcp3

set sink3 [new Agent/TCPSink]

$ns attach-agent $n6 $sink3

$ns connect $tcp3 $sink3

$tcp3 set fid\_ 4

#setup Application over TCP connection

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

set smtp0 [new Application/Traffic/Exponential]

$smtp0 attach-agent $tcp1

$smtp0 set packetSize\_ 210

$smtp0 set burst\_time\_ 50ms

$smtp0 set idle\_time\_ 50ms

$smtp0 set rate\_ 100k

set http0 [new Application/Traffic/Exponential]

$http0 attach-agent $tcp2

$http0 set packetSize\_ 210

$http0 set burst\_time\_ 50ms

$http0 set idle\_time\_ 50ms

$http0 set rate\_ 100k

set cbr0 [new Application/Traffic/CBR]

$cbr0 attach-agent $tcp3

$cbr0 set packetSize\_ 210

$cbr0 set burst\_time\_ 50ms

$cbr0 set idle\_time\_ 50ms

$cbr0 set rate\_ 100k

proc finish {} {

global ns tracefile namfile

$ns flush-trace

close $tracefile

close $namfile

exec nam tcptahoemeshnam.nam &

exit 0

}

proc record {} {

set ns [new Simulator instance]

set time 0.1

set nowtime [$ns now]

$ns at [expr $nowtime + $time] "record"

}

#Scheduling the events

$ns at 0.1 "$ftp0 start"

$ns at 0.1 "$smtp0 start"

$ns at 0.1 "$http0 start"

$ns at 0.1 "$cbr0 start"

$ns at 19.9 "$ftp0 stop"

$ns at 19.9 "$smtp0 stop"

$ns at 19.9 "$http0 stop"

$ns at 19.9 "$cbr0 stop"

$ns at 0.0 "record"

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "finish"

$ns at $val(stop) "puts \"done\" ; $ns halt"

#Run the simulator

$ns run

**AWK Scripts:**

**throughput.awk:**

BEGIN {

rcvdSize = 0;

startTime = 0.5;

stopTime = 5.0;

}

{ event = $1;

time = $2;

node\_id = $3;

pktsize = $6;

level = $4;

if (event == "s") {

if (time < startTime) {

startTime = time;

}

}

if (event == "r") {

if (time > stopTime) {

stopTime = time;

}

rcvdSize += pktsize;

}

}

END {

printf("Average Throughput[kbps] = %.2f\tStartTime = %.2f\t StopTime = %.2f\n",

(rcvdSize/(stopTime - startTime)) \* (8 / 1000), startTime, stopTime);

}

**pktloss.awk:**

BEGIN {

recieve = 0;

drop = 0;

total = 0;

ratio1 = 0.0;

ratio2 = 0.0;

}

{ if ($1 == "r" && $4 == 3)

{

recieve++;

}

if ($1 == "d")

{

drop++;

}

}

END {

total = recieve + drop;

ratio1 = (recieve / total);

ratio2 = (drop / total);

printf("Total packets sent = %d\n", total);

printf("Packets recieved = %d\n", recieve);

printf("Packets dropped = %d\n", drop);

printf("Delivery rate = %f\n", ratio1);

printf("Loss rate = %f\n", ratio2);

}

**avgdelay.awk:**

BEGIN {

highest\_packet\_id = 0;

avg=0.0

}

{

action = $1;

time = $2;

node\_1 = $3;

node\_2 = $4;

src = $5;

flow\_id = $8;

node\_1\_address = $9;

node\_2\_address = $10;

seq\_no = $11;

packet\_id = $12;

if ( packet\_id > highest\_packet\_id ) highest\_packet\_id = packet\_id;

if ( start\_time[packet\_id] == 0 ) start\_time[packet\_id] = time;

if ( action != "d" ) {

if ( action == "r" ) {

end\_time[packet\_id] = time;

}

} else {

end\_time[packet\_id] = -1;

}

}

END {

for ( packet\_id = 0; packet\_id <= highest\_packet\_id; packet\_id++ ) {

start = start\_time[packet\_id];

end = end\_time[packet\_id];

packet\_duration = end - start;

avg+=packet\_duration;

#if ( start < end ) printf("%f %f %f\n",packet\_id, start, packet\_duration);

}

avg=avg/highest\_packet\_id;

printf("avg end to end delay %f\n",avg);

}

**CHAPTER 4: RESULTS**

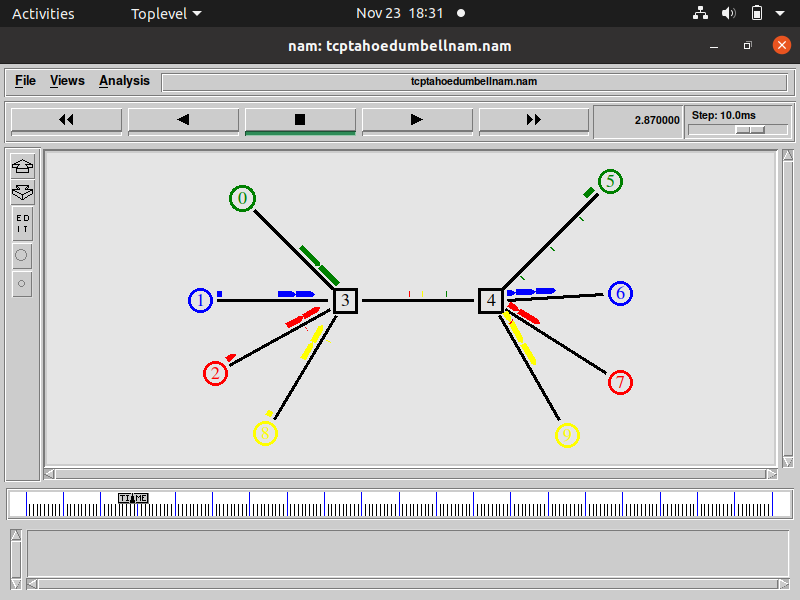


Fig. 4.1. Simulation screen at intermediate time of TCP Tahoe dumbbell topology

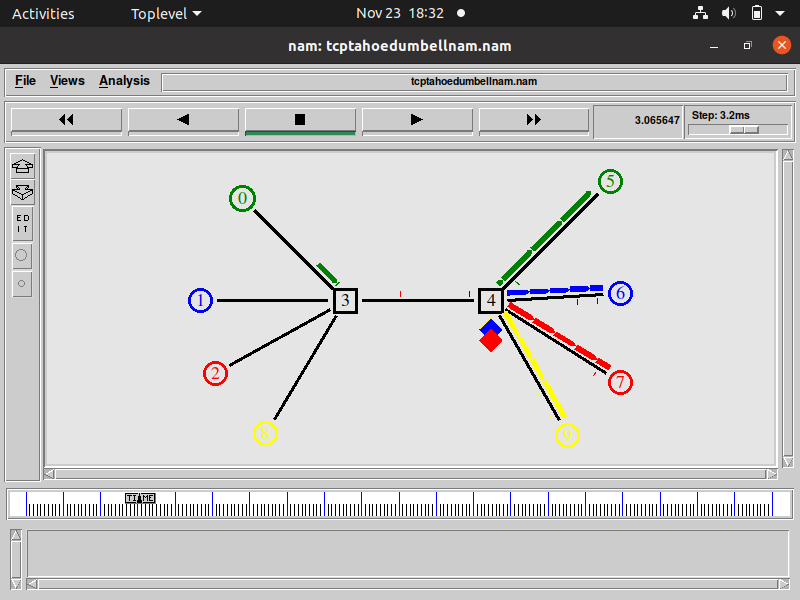


Fig. 4.2. Simulation screen when packet loss occurs due to congestion in TCP Tahoe dumbbell topology

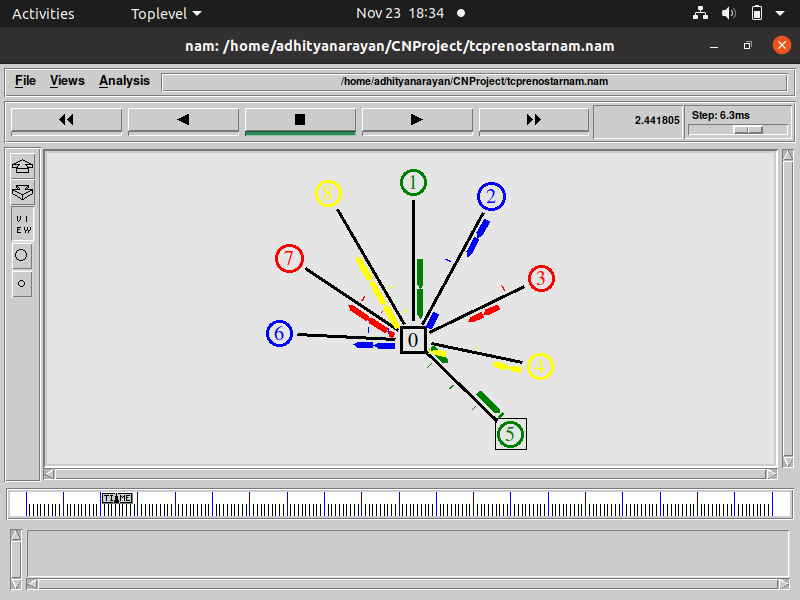


Fig 4.3 Simulation screen at intermediate time of TCP Reno Star topology

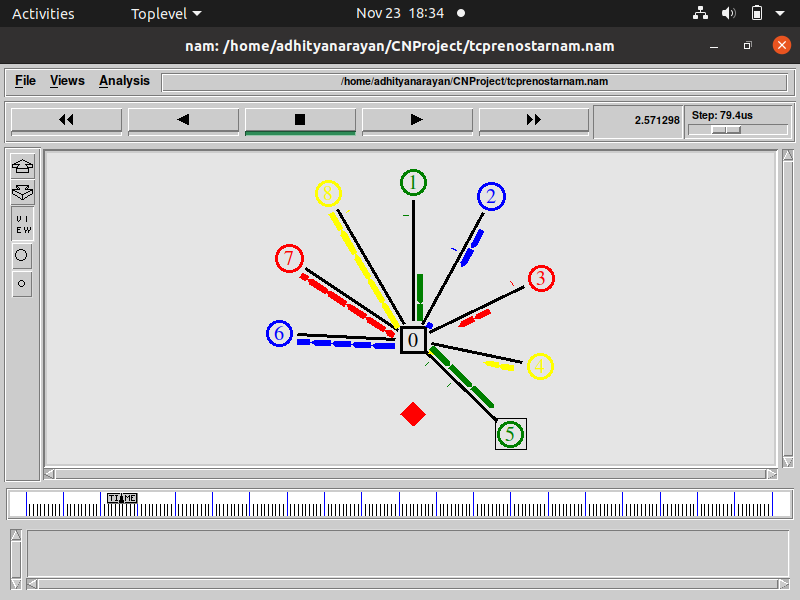


Fig 4.4. Simulation screen when packet loss occurs due to congestion in TCP Reno Star topology

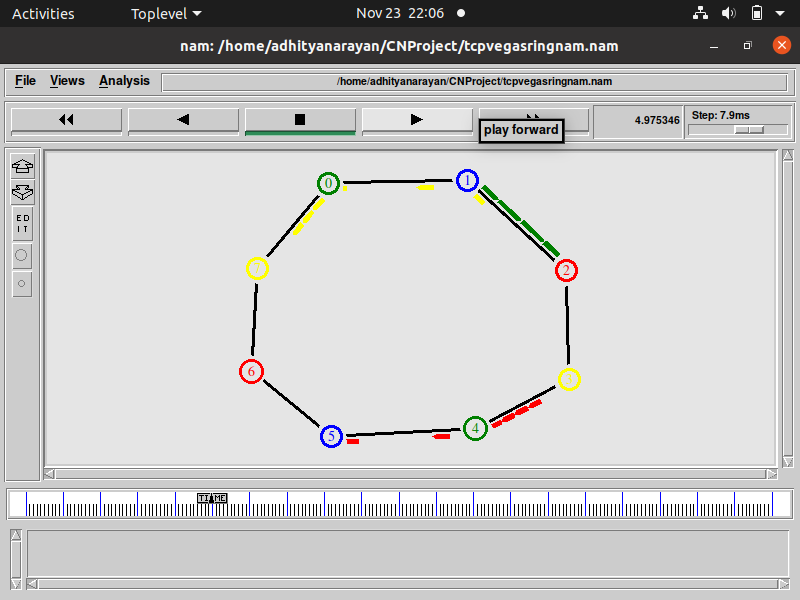


Fig 4.5. Simulation screen at intermediate time of TCP Vegas Ring topology

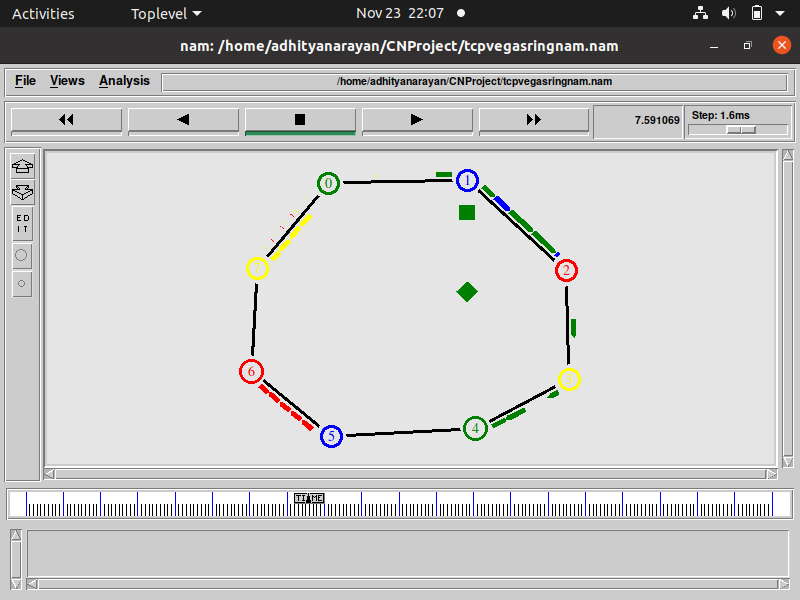


Fig. 4.6. Simulation screen when packet loss occurs due to congestion in TCP Vegas Star topology

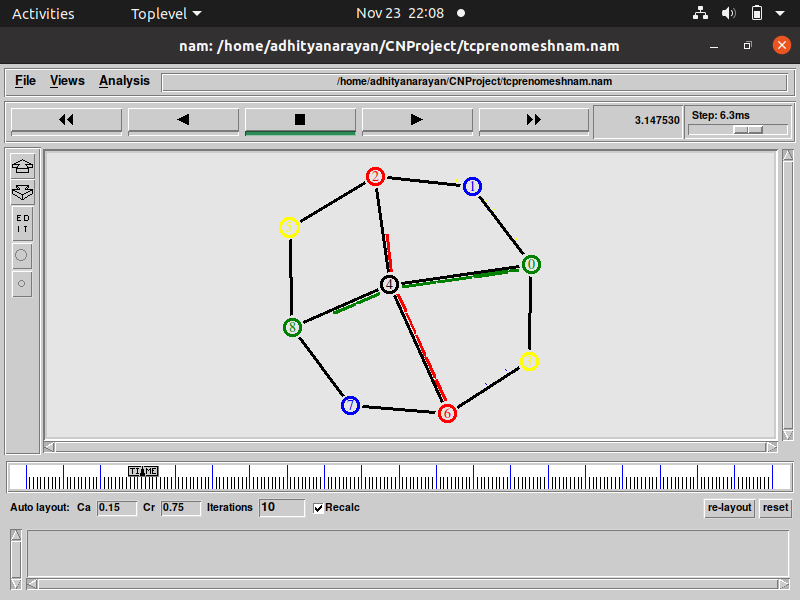


Fig. 4.7. Simulation screen at intermediate time of TCP Reno Mesh topology

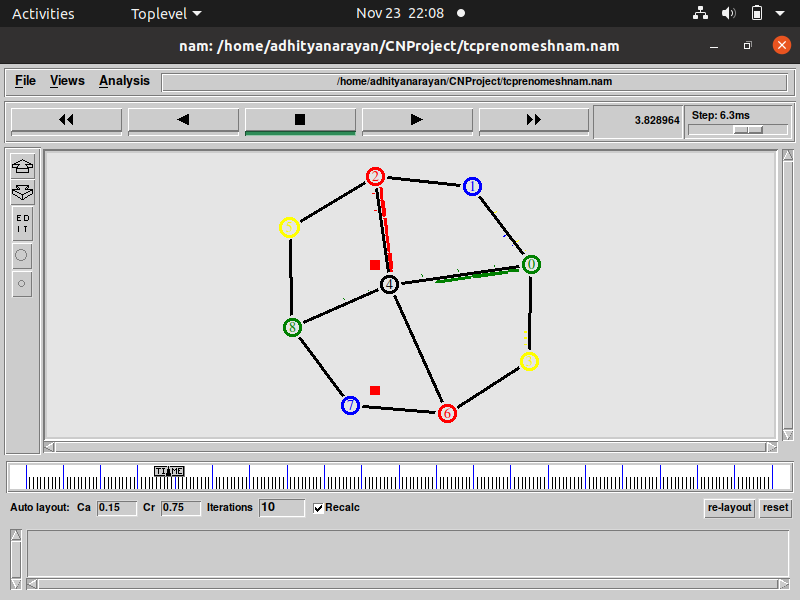


Fig. 4.8. Simulation screen when packet loss occurs due to congestion in TCP Reno Mesh Topology

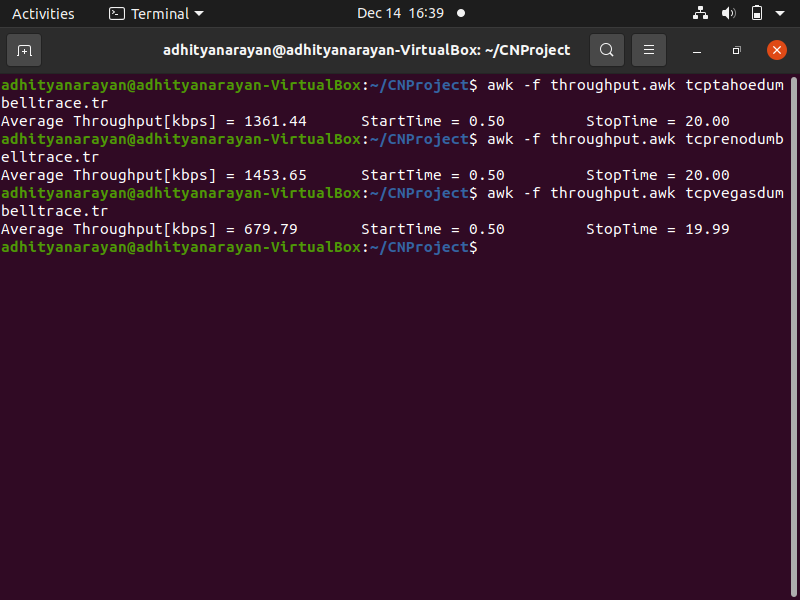


Fig. 4.9. Sample output of the awk script analysing on the throughput of dumbbell topology across various variants of TCP

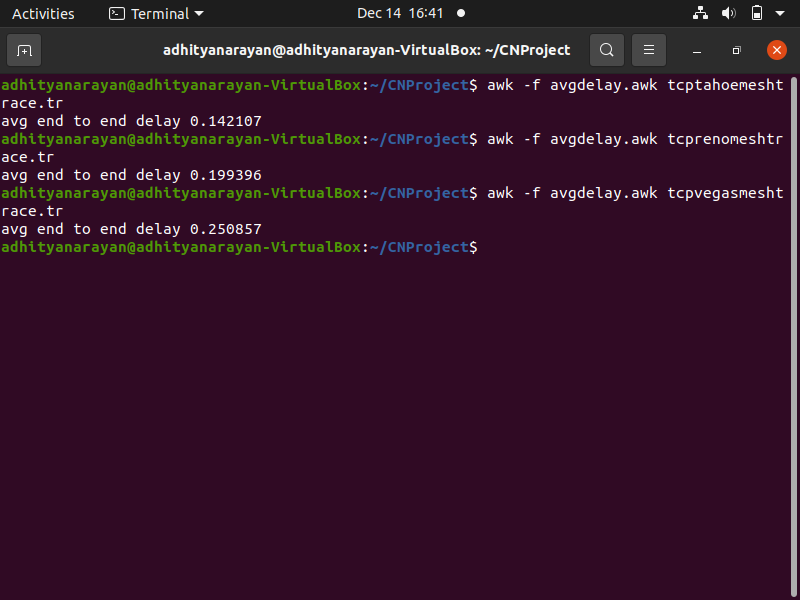
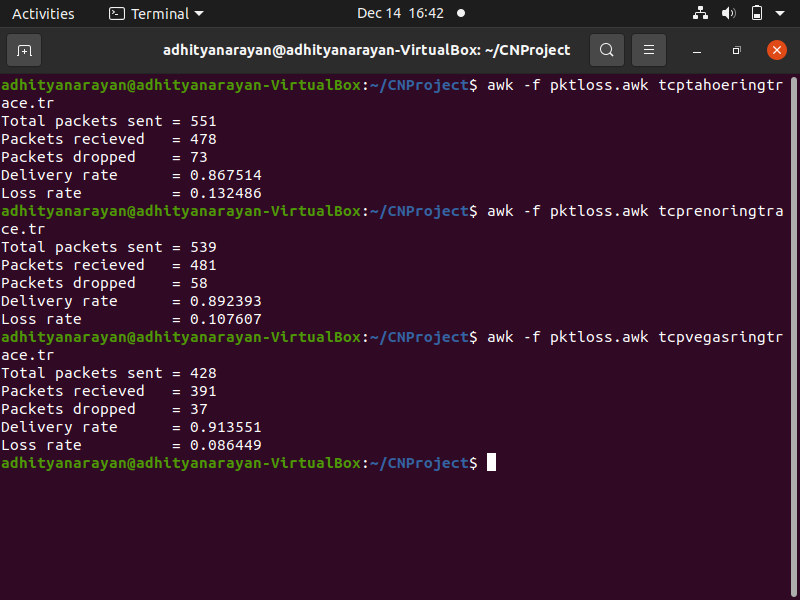


Fig. 4.10. Sample output of the awk script analysing the average delay of mesh topology across various variants of TCP

Fig. 4.11. Sample output of the awk script analysing the delivery ratio of ring topology across various variants of TCP

**CHAPTER 5: PERFORMANCE EVALUATION**

As you can see from the results, we have successfully simulated the working of the TCP variants, namely Tahoe, Reno, and Vegas and have even looked at how they perform by running awk scripts on the trace files.

Now coming to the whole notion of this work, we see and compare the performance of these TCP variants based on the data we have collected after analysing the trace files using awk scripts. The data is first put as table below, and then we have separated the data into bar graphs for ease of comparison, as anything we see visually, we tend to understand more.

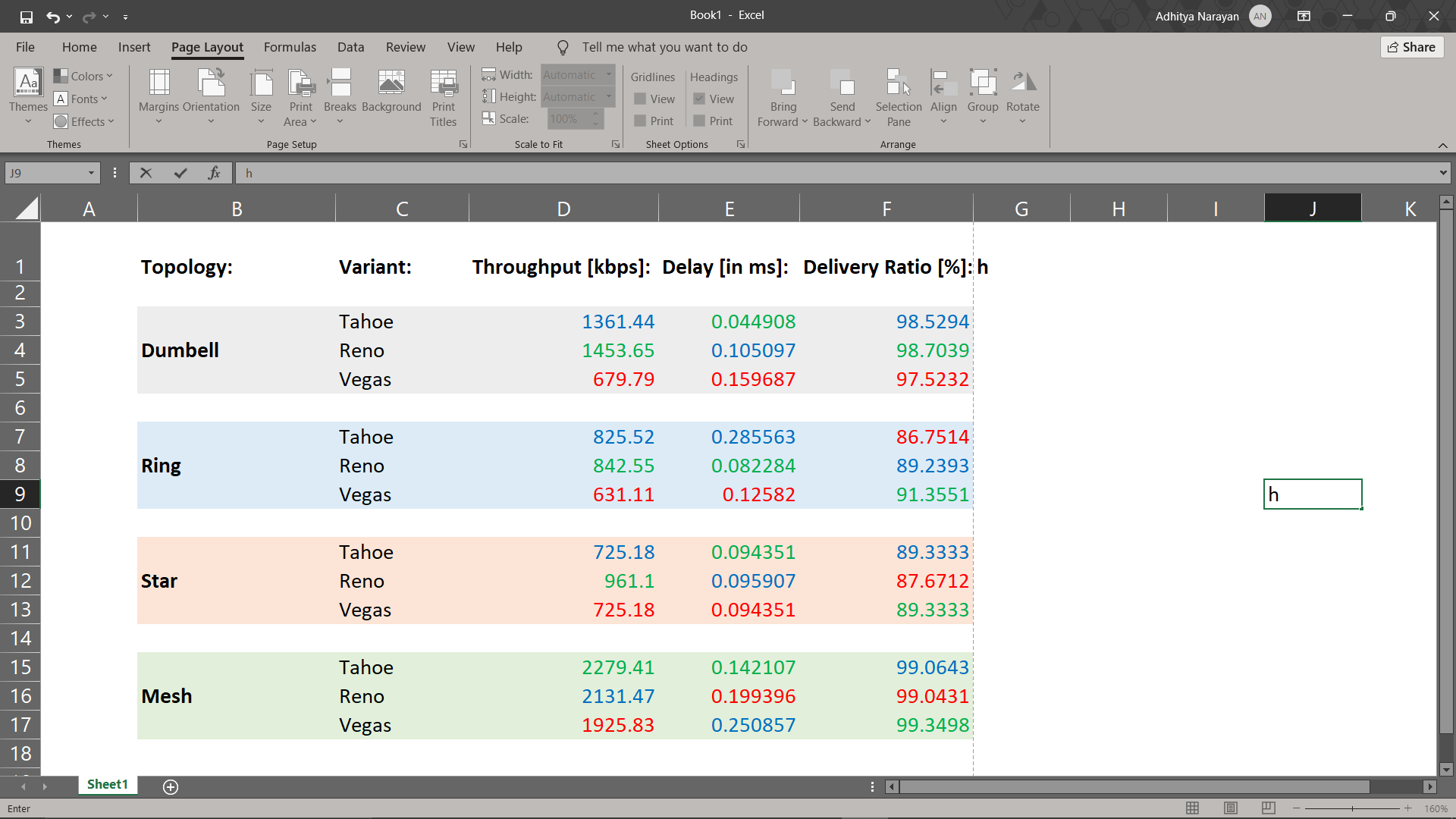
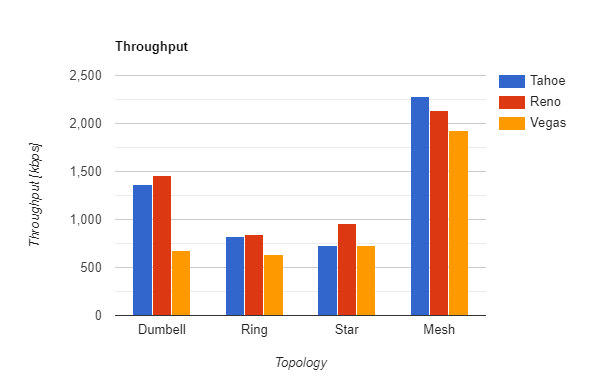


Fig 5.1 Comparison of the variants of TCP based on throughput, delay, and delivery ratio

The best performance obtained in every category is marked in green, while the worst one is marked in red. Looking at the table, we can plot graphs and compare the variants parameter-wise.

**Throughput Comparison:**

 Fig. 5.2 Comparison of Throughput

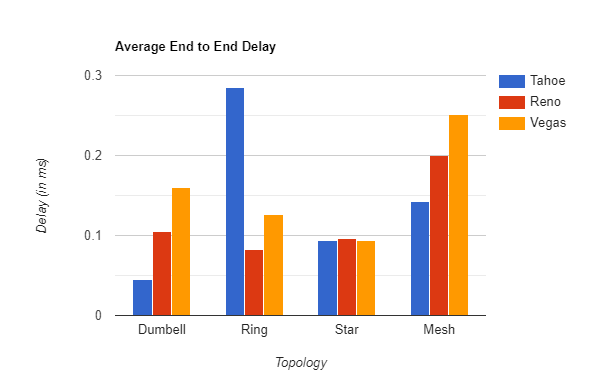
Looking at the graph, we can see that for Dumbbell topology, Reno gives us the best throughput. (For exact value, check table) But also, Tahoe is almost same as Reno. But Vegas lacks behind drastically here, giving us only half the throughput as the other two.

For Ring topology, again the scenarios remain almost same as for Dumbbell, only difference being Vegas now gives almost 80% of what the other two give.

For Star topology, Reno leads the race by a couple of hundred kbps, while the other two, Tahoe and Vegas perform the same.

For Mesh topology, surprisingly, the order of performance turns to Tahoe > Reno > Vegas. Across all topologies, Vegas doesn’t match the throughput of the other two, while Reno leads in 3 out of 4 topologies. So, for Throughput, it is wise to say that **TCP Reno gives us the best possible throughput** (among these three) most of the times.

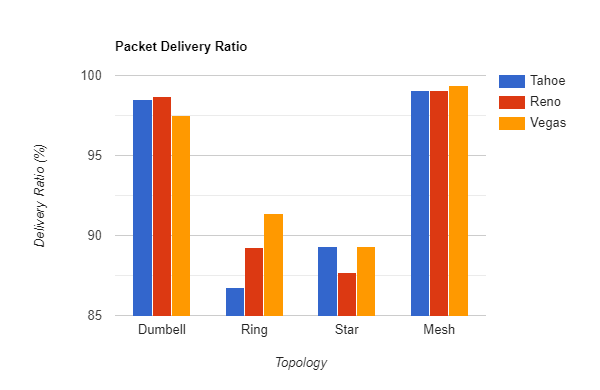
**End to End Delay Comparison:**

 **Fig 5.3 Comparison of Average End to End Delay**

Lower the delay, better the performance. So going by that, for Dumbbell topology, Tahoe performs the best, and for Ring topology, it’s the opposite. Tahoe is the worst one, that too by a huge margin. We have Reno doing good job for Ring topology. For Star, both Tahoe and Vegas perform good, while Reno lacks behind minutely.

For Mesh, again, we have Tahoe performing the best, while Vegas is not proving to be useful here. So, we can say that, **TCP Tahoe provides the lowest optimum delay**, given that it leads in the above 3 out of 4 cases.

**Packet Delivery Ratio Comparison:**

 **Fig 5.4 Comparison of Packet Delivery Ratio**

Looking at the table, for Dumbbell, we can say that, all three do good work, with a ratio of >95%, but even among that, TCP Reno does the best. For Ring, TCP Vegas shocks us all by doing a great job, the best among the three. Again, for Star, both Tahoe and Star are doing the best, while Reno lacks behind. For mesh, again, even though all of them provide a delivery ratio of >99%, Vegas is the best among the three. So it is safe to quote here that **TCP Vegas provides the best Packet Delivery Ratio** among the three, as it leads in 3 out of the above 4 cases.

So, as we saw above, each variant outperforms the other in one or the other parameter. Tahoe minimizes the delay, Reno provides maximum throughput, and Vegas provides the most reliable data transfer. These are all the desired characteristics in each of them. If we look at their demerits, Vegas has the lowest throughput, Tahoe is not as reliable as Vegas, etc. So, we can see that everything comes with a cost. If throughput is more, delay and packet delivery is compromised, and if its reliable, throughput is not much, and so on.

So, everything just lies in this trade-off between the various performance parameters. Someone who wants all the packets to be delivered without fail most of the times, and is willing to do it a little slowly may use Vegas, while someone who prefers a fast connectivity more then reliability may use Tahoe, etc. The only thing that matters in the design of a TLP such as TCP is the trade-off between these performance measures.

**CHAPTER 6: REFERENCES**

1. <https://www.ripublication.com/acst17/acstv10n6_17.pdf>
2. <https://www.researchgate.net/publication/44259574_TCP_Variants_and_Network_Parameters_A_Comprehensive_Performance_Analysis>
3. <https://ieeexplore.ieee.org/document/8229880>