

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

III year CSE – V Semester

**UCS 2501 Computer Networks**

**Team Project**

# TITLE:PERFORMANCE OF TCP VEGAS AND TCP NEW RENO

**Academic year : 2023-2024**

**Batch: 2021 - 2025**

Faculty Incharge**:** Dr. S.V.Jansi Rani

Team members

Sam Matthew Hendry – 3122 21 5001 091

G Vaishal Raj — 3122 21 5001 119

Kevin John Daniel – 3122 21 5001 701

S Rayhan Abdurrahman – 3122 21 5001 079

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CONTENT** | **PAGE NO** |
| 1 | Problem Statement | 3 |
| **2** | Protocol / Method Explanation | 4 |
| **3** | Code | 9 |
| **4** | Output | 11 |
| **5** | Learning outcomes | 13 |
| **6** | Readme File | 14 |
| **7** | Github link | 15 |

PROBLEM DEFINITION:-

* In this project, we aim to evaluate the goodput performance of TCP Vegas and TCP New Reno, using NS2.
* The primary goals include creating a network, deploying TCP agents, creating congestion windows, and plotting the graphs using XGraph.

PROJECT OVERVIEW:-

**1. Creating a Network**

Creation of the network nodes, establishing duplex links and defining the orientation of the network.

**2. Deploying TCP Agents**

We deploy TCP Agents, first a TCP Vegas agent, and then a TCP New Reno agent. We create a sending and a receiving agent for both.

**3. Creating Congestion Windows**

Congestion windows are created for each protocol, and they are plotted on a graph. An outfile is created for each of them, having a .xg extension.

**4. Plotting on Graph**

The outfiles are then interpreted using Graph, and parameters such as colour, thickness, range of x axis, etc can be adjusted.

**5. Documentation**

Provide comprehensive documentation that provides overview about the project, define the components used and configuration required and a readme file on how to run the project.

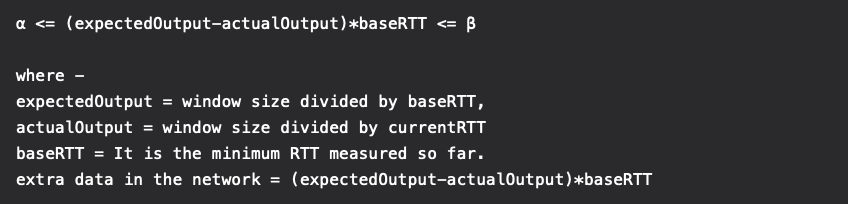
PROTOCOL/METHOD EXPLANATION:-

What is TCP Vegas:-

It is a completely different approach to bandwidth management and TCP was the first attempt at it. And is based on congestion detection before packet loss occurs. TCP Vegas controls the congestion window by measuring the roundtrip times of the packets. In this, we find the extra data which is measured by subtracting expected data and the actual data in the network. This extra data is compared with two thresholds i.e alpha α and beta β accordingly the window size is increased or decreased.

So the steps in the algorithm are as follows.

* The sender measures the expected flow rate i.e cwnd/BaseRTT.
* Then the sender finds the current flow rate by using the actual roundtrip time of the packet.
* After that sender computer the extra data in the queue i.e extra data = (expected – actual) \* BaseRTT

So, the formula used in TCP Vegas is as follows.

So 3 cases are there as follows.

**Case-1 :**   
 If extra data in the network is greater than Beta then the window size is decreased.

cwnd = cwnd - 1

**Case-2 :**    
 If extra data in the network is less than alpha then the window size is increased.

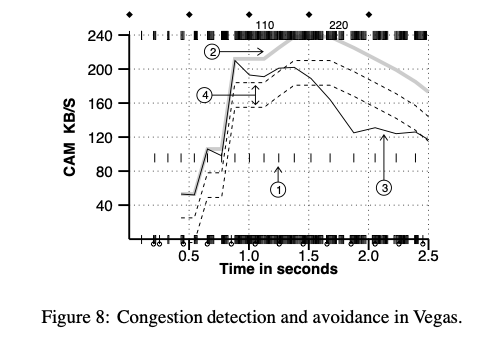
cwnd = cwnd + 1

**Case-3 :**  
 If extra data lies between closed intervals [alpha, beta] then the window size is not changed.

cwnd = cwnd

Need for TCP Vegas

TCP Reno and TCP Tahoe models can determine the congestion in the network only when there some packet loss occurred in the system. So in these models, we have compensated for the packet to sense the congestion in the network. In these models when packet loss occurs the window size is decreased and the system enters the congestion avoidance phase.

While TCP Vegas senses the congestion in the network before any packet loss occurs and instantly it decreases the window size. So, TCP Vegas handles the congestion without any packet loss occurring.

What is TCP New Reno:-

TCP New Reno is the extension of TCP Reno. It overcomes the limitations of Reno. TCP Reno is the second variant of the TCP which came up with an in-built congestion algorithm. Congestion handling was not an integral part of the original TCP/IP suite. TCP Reno is the extension of [TCP Tahoe](https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno/), and NewReno is the extension of TCP Reno. In Reno, when packet loss occurs, the sender reduces the cwnd by 50% along with the ssthresh value. This would allow the network to come out of the congestion state easily. But Reno suffered from a very critical backlog which hurts its performance.

When multiple packets are dropped in the same congestion window (say 1-10) then every time it knows about a packet loss it reduces the cwnd by 50%. So, for 2 packet loss, it will reduce the cwnd by 4 times (50% twice). But, one reduction of 50% per congestion window was enough for recovering all those lost packets. Say cwnd=1024 and 10 packets are dropped in this window, Reno will reduce cwnd by 50% 10 times, finally cwnd=1024/210 = 1, it is astonishing. It would take 10 RTTs by the sender to again grow its cwnd up to 1024 using [slow start](https://www.geeksforgeeks.org/slow-start-backoff-algorithm-for-ad-hoc/) leave alone the AIMD algorithm.

It uses the concept of partial acknowledgement. When the sender receives the ACK of the first retransmitted packet then it does not consider it a “New ACK” unlike TCP Reno. NewReno checks if all the previously transmitted packets of that particular window are ACKed or not. If there are multiple packets lost in the same congestion window then the receiver would have been sending the duplicate ACKs only even after receiving the retransmitted packet. This will make it clear to the sender that all the packets are not reached the receiver and hence sender will not consider that ACK as new. It will consider it a partial ACK because only a partial window is being ACKed not the whole. Reno used to come out of the fast recovery phase after receiving a new ACK, but NewReno considers that ACK as partial and does not come out of the fast recovery phase. It wisely makes the decision of ending the fast recovery phase when it receives the Cumulative ACK of the entire congestion window.

Therefore, NewReno will detect multiple packet loss immediately and does not come out of the Fast recovery phase prematurely unlike Reno.

pasted-image.tiff

pasted-image.tiffCODE:-

# Create a simulator object

set ns [new Simulator]

# Open the nam file basic1.nam and the variable-trace file basic1.tr

set namfile [open basic1.nam w]

$ns namtrace-all $namfile

set tracefile [open basic1.tr w]

$ns trace-all $tracefile

# Define a 'finish' procedure

proc finish {} {

global ns namfile tracefile

$ns flush-trace

close $namfile

close $tracefile

exit 0

}

# Create the network nodes

set A [$ns node]

set R [$ns node]

set B [$ns node]

# Create a duplex link between the nodes

$ns duplex-link $A $R 10Mb 10ms DropTail

$ns duplex-link $R $B 800Kb 50ms DropTail

# The queue size at $R is to be 7, including the packet being sent

$ns queue-limit $R $B 7

# some hints for nam

# color packets of flow 0 red

$ns color 0 Red

$ns duplex-link-op $A $R orient right

$ns duplex-link-op $R $B orient right

$ns duplex-link-op $R $B queuePos 0.5

# Create a TCP sending agent (TCP Vegas) and attach it to A

set tcpVegas [new Agent/TCP/Vegas]

$tcpVegas set class\_ 0

$tcpVegas set window\_ 100

$tcpVegas set packetSize\_ 960

$ns attach-agent $A $tcpVegas

# Let's trace some variables

$tcpVegas attach $tracefile

$tcpVegas tracevar cwnd\_

$tcpVegas tracevar ssthresh\_

$tcpVegas tracevar ack\_

$tcpVegas tracevar maxseq\_

# Create a TCP receive agent (a traffic sink) and attach it to B

set end0 [new Agent/TCPSink]

$ns attach-agent $B $end0

# Connect the traffic source with the traffic sink

$ns connect $tcpVegas $end0

# Schedule the connection data flow; start sending data at T=0, stop at T=10.0

set myftp [new Application/FTP]

$myftp attach-agent $tcpVegas

$ns at 0.0 "$myftp start"

$ns at 10.0 "finish"

proc plotWindow {tcpSource outfile} {

global ns

set now [$ns now]

set cwnd [$tcpSource set cwnd\_]

# the data is recorded in a file called congestion.xg (this can be plotted

# using xgraph or gnuplot. this example uses xgraph to plot the cwnd\_

puts $outfile "$now $cwnd"

$ns at [expr $now+0.1] "plotWindow $tcpSource $outfile"

}

set outfileVegas [open "vegas.xg" w]

$ns at 0.0 "plotWindow $tcpVegas $outfileVegas"

$ns at 10.1 "exec xgraph -lw 2 -geometry 800x400 vegas.xg"

# Repeat the above steps for TCP New Reno

# Create a TCP sending agent (TCP New Reno) and attach it to A

set tcpNewReno [new Agent/TCP/Newreno]

$tcpNewReno set class\_ 1

$tcpNewReno set window\_ 100

$tcpNewReno set packetSize\_ 960

$ns attach-agent $A $tcpNewReno

# Let's trace some variables

$tcpNewReno attach $tracefile

$tcpNewReno tracevar cwnd\_

$tcpNewReno tracevar ssthresh\_

$tcpNewReno tracevar ack\_

$tcpNewReno tracevar maxseq\_

# Create a TCP receive agent (a traffic sink) and attach it to B

set end1 [new Agent/TCPSink]

$ns attach-agent $B $end1

# Connect the traffic source with the traffic sink

$ns connect $tcpNewReno $end1

# Schedule the connection data flow; start sending data at T=0, stop at T=10.0

set myftp1 [new Application/FTP]

$myftp1 attach-agent $tcpNewReno

$ns at 0.0 "$myftp1 start"

$ns at 10.0 "finish"

set outfileNewReno [open "newreno.xg" w]

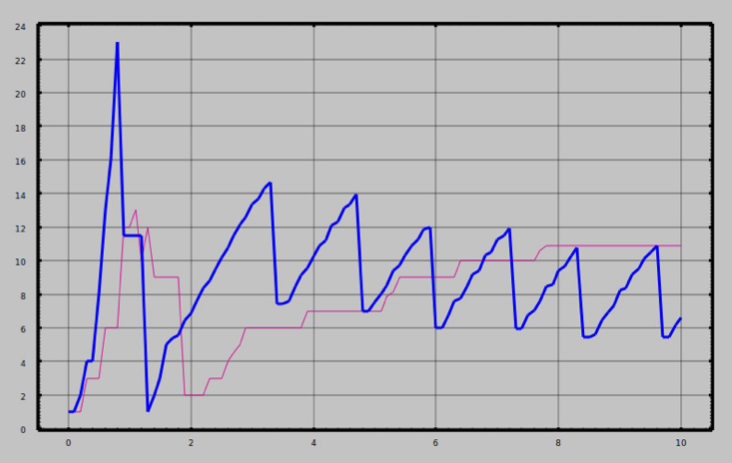
$ns at 0.0 "plotWindow $tcpNewReno $outfileNewReno"

$ns at 10.1 "exec xgraph -lw 2 -geometry 800x400 newreno.xg"

# Run the simulation

$ns run

OUTPUT:-



LEARNING OUTCOMES:-

1. IP Addressing Mastery:

Gain proficiency in assigning IP addresses, distinguishing between private and public IP address spaces, and understanding subnetting principles.

1. Router Configuration Expertise:

Develop skills in configuring routers with multiple interfaces, implementing dynamic routing protocols (RIP), and establishing inter-router connectivity.

1. Switch Management and LAN Connectivity:

Learn how to configure switches to connect devices within a Local Area Network (LAN) and comprehend the basics of LAN design.

1. NAT Implementation Competence:

Acquire the ability to implement Network Address Translation (NAT) for enabling communication between private and public IP addresses, including the creation of static NAT mappings.

1. Network Topology Design Proficiency:

Understand the essentials of designing basic network topologies, identifying components such as routers, switches, and PCs, and recognizing their interconnections for optimal network functionality.

**README FILE**

**Requirements:-**

1. Cisco packet tracer

**Configuration:-**

1.Access the router's command-line interface.

2.Configure the necessary interfaces with appropriate IP addresses. For example:

interface Serial0/0

ip address 200.200.200.1 255.255.255.0

no shutdown

exit

interface FastEthernet0/0

ip address 192.168.1.1 255.255.0.0

no shutdown

exit

3.Set up static NAT mappings for internal devices:

ip nat inside source static 192.168.1.10 200.200.200.3

ip nat inside source static 192.168.1.11 200.200.200.4

4.Define NAT Inside and Outside interfaces:

interface Serial0/0

ip nat outside

exit

interface FastEthernet0/0

ip nat inside

exit

5.Save the configuration.

**Usage:-**

Make sure the router is powered on and connected to the network. The static NAT mappings should now be in effect.

Connectivity Testing:-

To test the connectivity, initiate ping tests between devices:

1.From PC1, ping an external device using its public IP address.

2.From an external device, ping PC1 using its private IP address.

[Github link](https://github.com/MeSubashh/Static-Nat-.git)