UCS2501 COMPUTER NETWORKS

ASSIGNMENT-2

TEAM

R.Akila	3122215001006
Apurva Narayan	3122215001011
Arvind Minjur	3122215001012
Ashwini Parvatha GS	3122215001016
A Bhavana	3122215001019
Charumathi P	3122215001020

PROBLEM STATEMENT

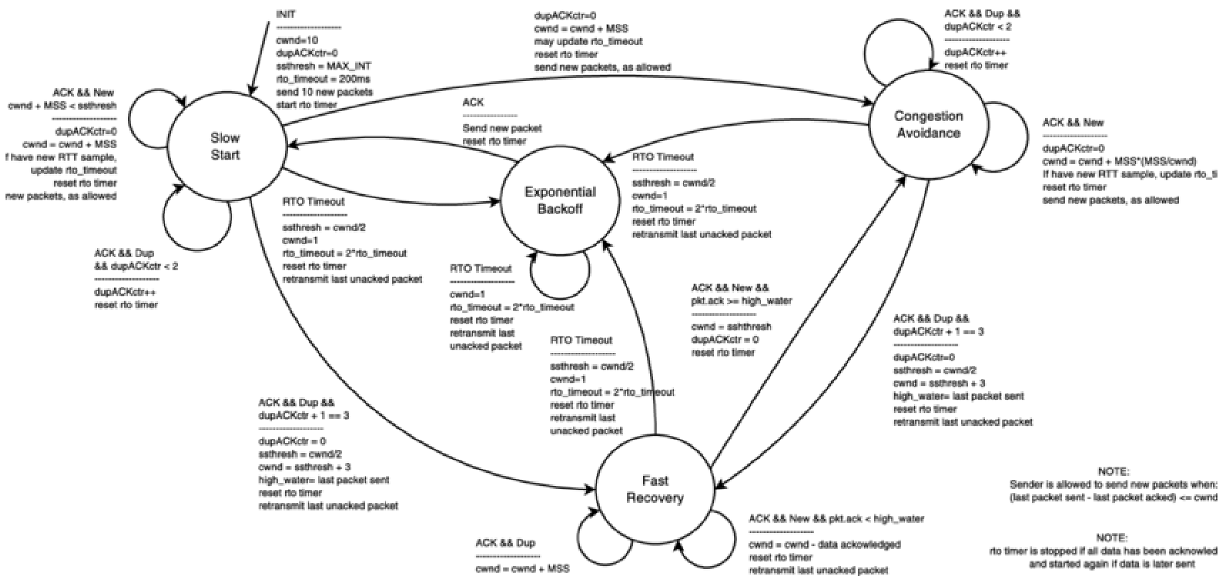
Using NS2, evaluate the goodput performance of TCP Vegas, TCP New Reno.

Plot the congestion window.

TCP NEW RENO

DEFINITION

TCP NewReno is a modest tweak to Fast Recovery which greatly improves handling of the case when two or more packets are lost in a windowful. It is generally considered to be a part of contemporary TCP Reno. It is described in terms of Estimated FlightSize rather than in terms of cwnd inflation and deflation. If two data packets are lost and the first is retransmitted, the receiver will acknowledge data up to just before the second packet, and then continue sending dupACKs of this until the second lost packet is also retransmitted. These ACKs of data up to just before the second packet are sometimes called partial ACKs, because retransmission of the first lost packet did not result in an ACK of all the outstanding data. The NewReno mechanism uses these partial ACKs as evidence to retransmit later lost packets, and also to keep pacing the Fast Recovery process.



CODE

```
set ns [new Simulator]



set namfile [open newreno.nam w]

$ns namtrace-all $namfile

set tracefile [open newreno.tr w]

$ns trace-all $tracefile
```

```
proc finish {} {
    global ns namfile tracefile
    $ns flush-trace
    close $namfile
    close $tracefile
    exit 0
}
set A [$ns node]
set R [$ns node]
set B [$ns node]


$ns duplex-link $A $R 10Mb 10ms DropTail
$ns duplex-link $R $B 800Kb 50ms DropTail


$ns queue-limit $R $B 7
```

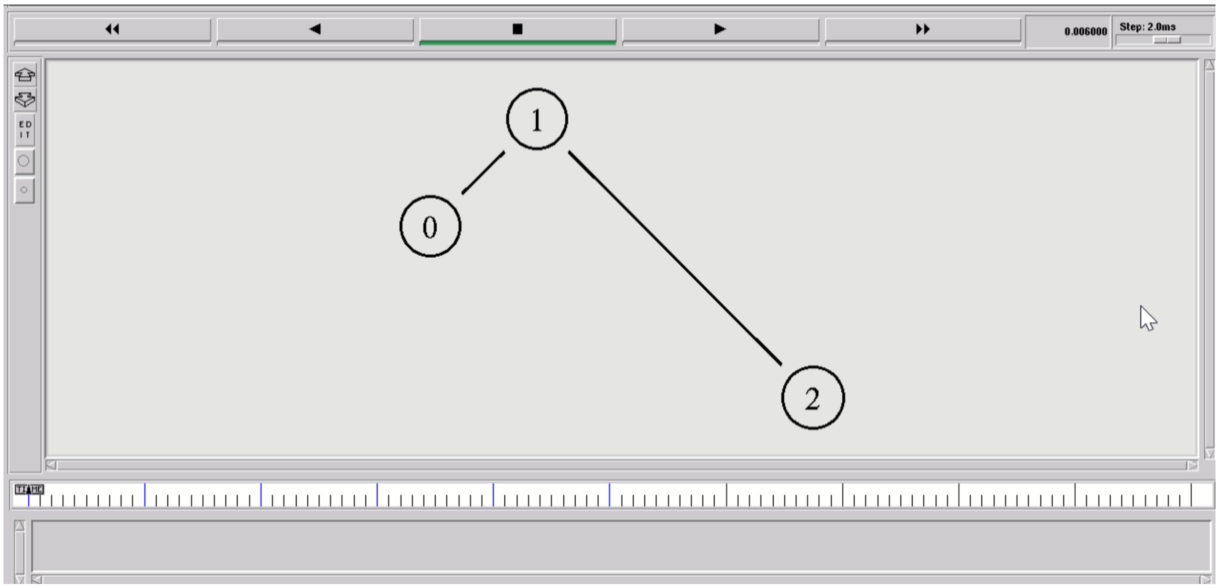
```
$ns color 0 Blue

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

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

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



set tcpNewReno [new Agent/TCP/Newreno]

$tcpNewReno set class_ 1

$tcpNewReno set window_ 100

$tcpNewReno set packetSize_ 960

$ns attach-agent $A $tcpNewReno


$tcpNewReno attach $tracefile

$tcpNewReno tracevar cwnd_

$tcpNewReno tracevar ssthresh_

$tcpNewReno tracevar ack_

$tcpNewReno tracevar maxseq_
```

```
$tcpNewReno set fid_ 0

set end1 [new Agent/TCPSink]
$ns attach-agent $B $end1

$ns connect $tcpNewReno $end1

set myftp1 [new Application/FTP]
$myftp1 attach-agent $tcpNewReno
$ns at 0.0 "$myftp1 start"
$ns at 10.0 "finish"

proc plotWindow {tcpSource outfile} {
    global ns
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
```

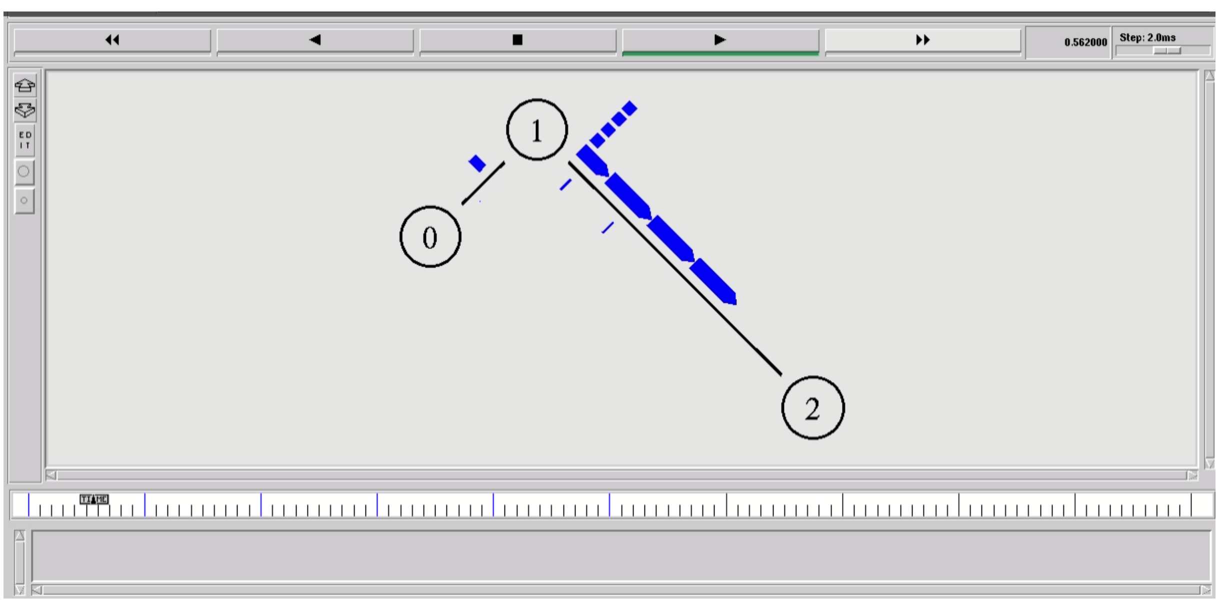
```
    puts $outfile "$now $cwnd"

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

}


set outfileNewReno [open "newreno.xg" w]

$ns at 0.0 "plotWindow $tcpNewReno $outfileNewReno"

$ns  at  10.1  "exec  xgraph  -lw  2  -geometry  800x400  -x1  'RTT
(seconds)' -y1 'Congestion Window Size(MSS)' newreno.xg"


after 1000 {

    exec nam newreno.nam

}


$ns run
```

OUTPUT

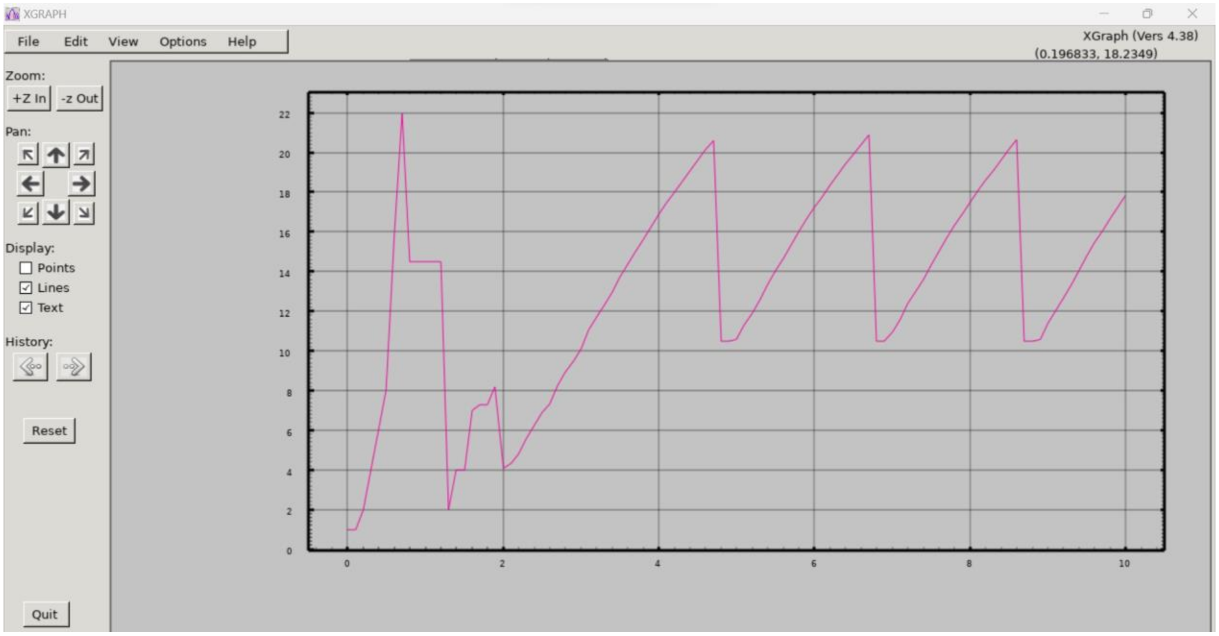
TOPOLOGY



SIMULATION



XGRAPH



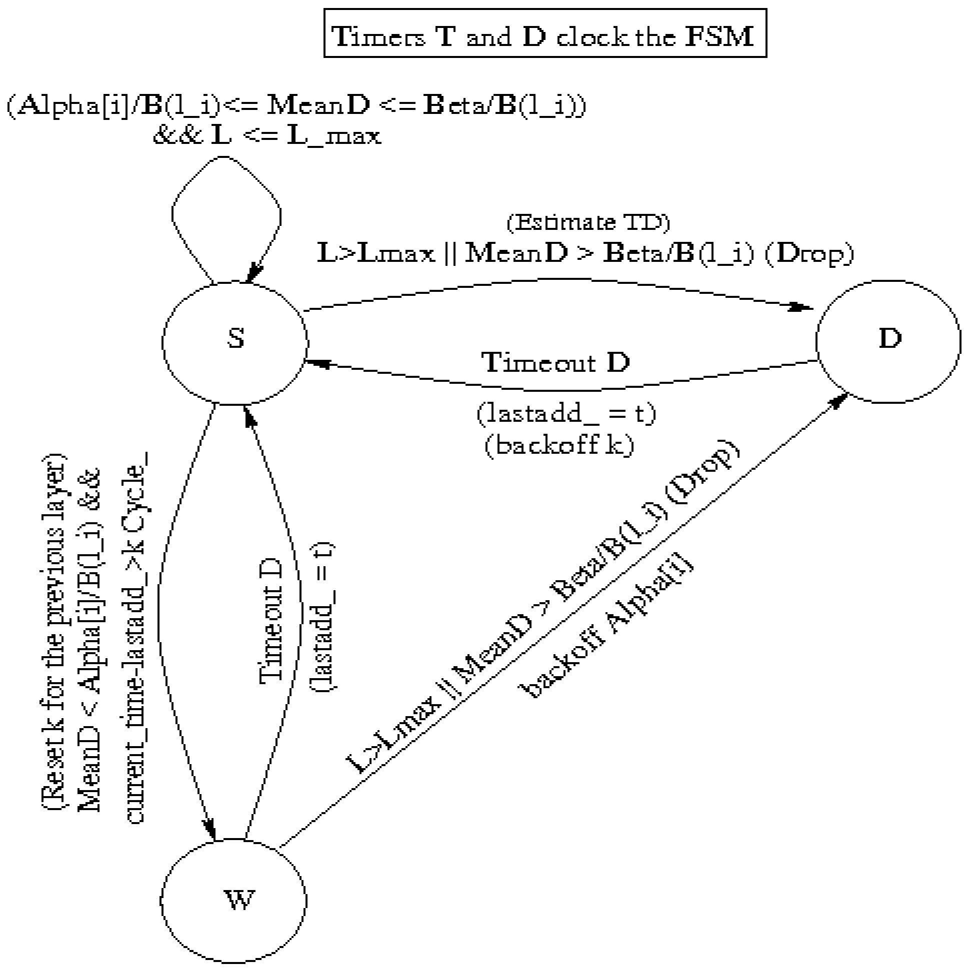
TCP VEGAS

DEFINITION

TCP Vegas is a congestion control algorithm designed to improve the efficiency and fairness of data transmission in computer networks. Developed as an alternative to traditional TCP congestion control mechanisms, Vegas employs a different approach by focusing on measuring the variation of the round-trip time (RTT) rather than packet loss as an indicator of network congestion.

By utilizing a more nuanced metric, Vegas aims to achieve better throughput and reduced latency. The algorithm dynamically adjusts its transmission rate based on the observed changes in RTT, allowing it to respond more promptly to network conditions.

This proactive nature enables TCP Vegas to provide a smoother and more stable performance compared to conventional TCP variants, making it particularly suitable for high-speed, low-latency networks where traditional TCP algorithms might fall short. While not as widely adopted as some other TCP variants, Vegas continues to be of interest in research and development efforts seeking to optimize network protocols for diverse applications.



CODE

```
set ns [new Simulator]



set namfile [open vegas.nam w]

$ns namtrace-all $namfile

set tracefile [open vegas.tr w]

$ns trace-all $tracefile



proc finish {} {

    global ns namfile tracefile

    $ns flush-trace

    close $namfile

    close $tracefile

    exit 0

}



set A [$ns node]
```

```
set R [$ns node]

set B [$ns node]


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

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


$ns queue-limit $R $B 7


$ns color 0 Blue

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

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

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


set tcpVegas [new Agent/TCP/Vegas]

$tcpVegas set class_ 0

$tcpVegas set window_ 100

$tcpVegas set packetSize_ 960
```

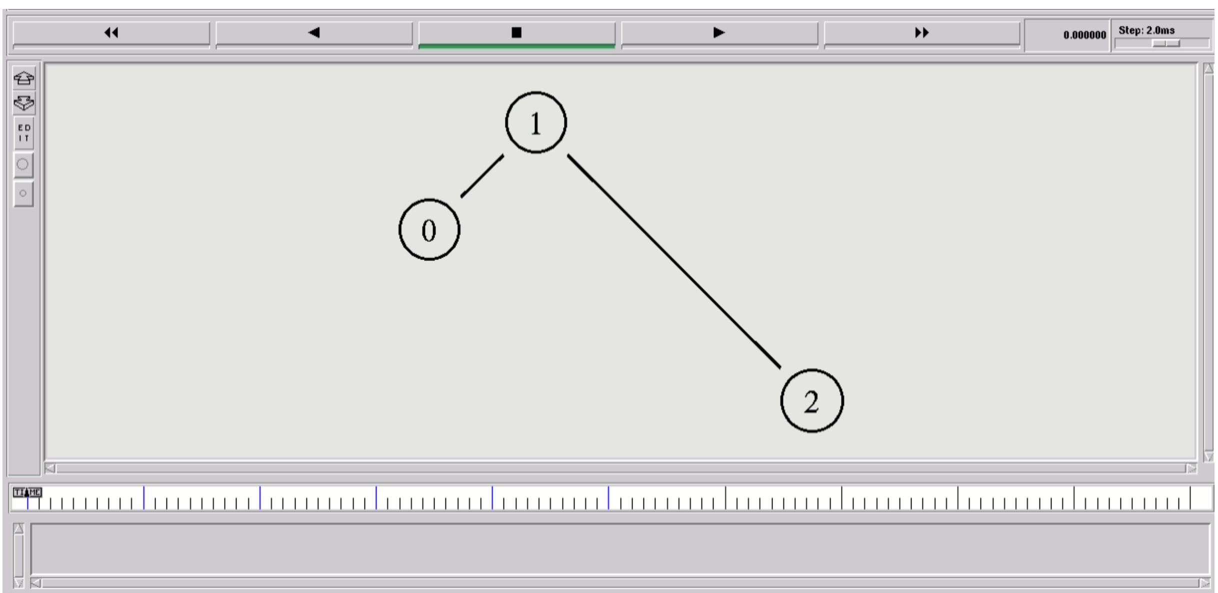
```
$ns attach-agent $A $tcpVegas


$tcpVegas attach $tracefile

$tcpVegas tracevar cwnd_

$tcpVegas tracevar ssthresh_

$tcpVegas tracevar ack_

$tcpVegas tracevar maxseq_

$tcpVegas set fid_ 0


set end0 [new Agent/TCPSink]

$ns attach-agent $B $end0


$ns connect $tcpVegas $end0


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_]


    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 -x1 'RTT (seconds)' -y1
'Congestion Window Size(MSS)' vegas.xg"
```

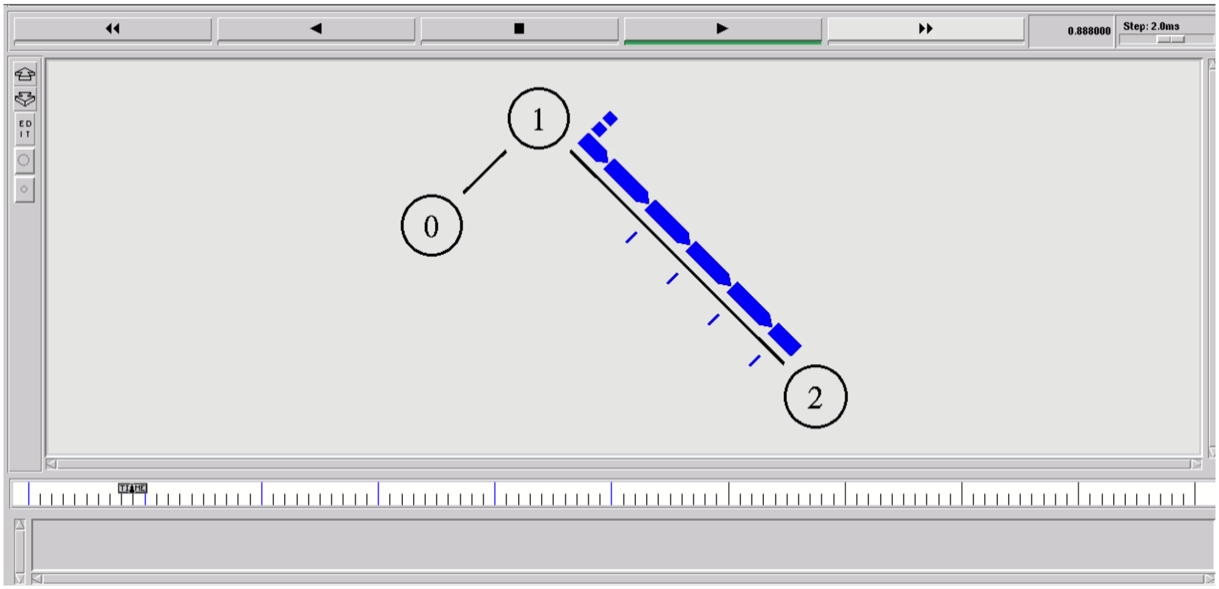
```
after 1000 {

    exec nam vegas.nam

}



$ns run
```

OUTPUT

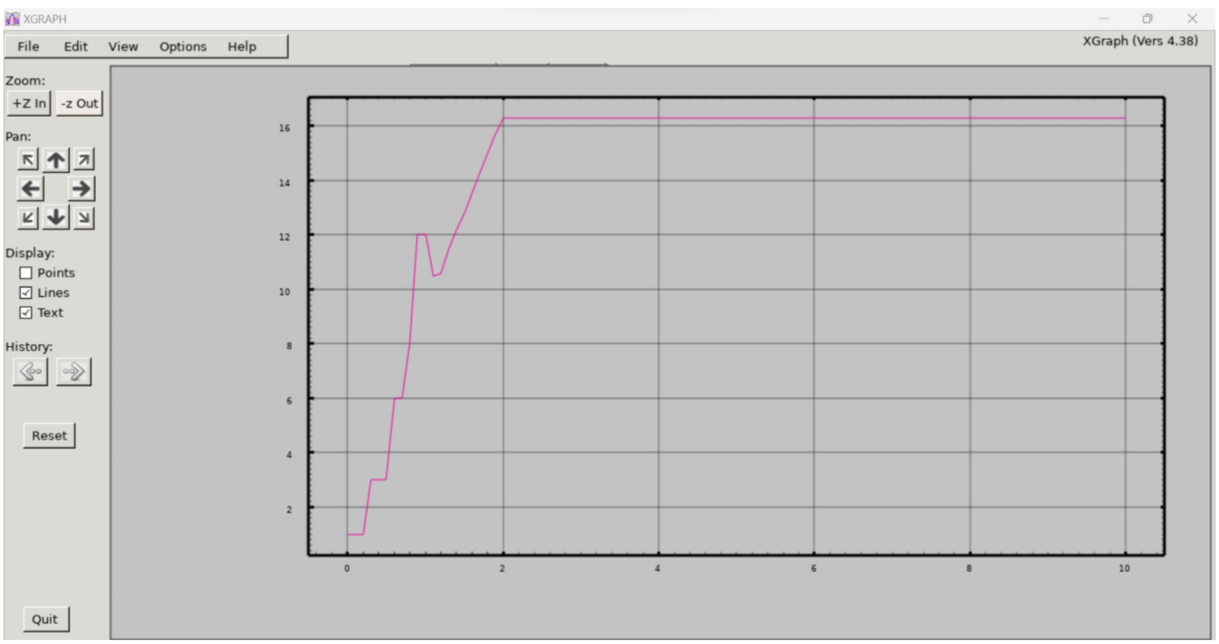
TOPOLOGY



SIMULATION



XGRAPH



LEARNING OUTCOMES:

- Definitions of congestion control frameworks like TCP NewReno and Vegas were studied
- The frameworks were simulated using the NS-2 application
- The behavior and working of these algorithms were further examined by plotting their performance.