CS 261 - COMPUTER NETWORKS

B. Tech CSE (V Semester)

Project Report

Group No.: 4
Section: S4-S5

Submitted By:

Kumari Renuka(U101115FCS111)
Rishabh Kumar Kandoi (U101115FCS283)
Shailesh Mohta (U101115FCS305)
Tanmay Patil (U101115FCS164)
…….

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NIIT UNIVERSITY, NEEMRANA

23 November 2017

TABLE OF CONTENTS

Contents *Page no.*

INTRODUCTION

PART A

TOPIC: IMPLEMENTING CONGESTION CONTROL IN TCP

- Rishabh Kumar Kandoi

PART B

TOPIC: TCL/FTP OVER A N-NODES AD-HOC NETWORK WITH DSDV ROUTING PROTOCOL

- Kumari Renuka

PART C

TOPIC: TCL SCRIPT TO CREATE MIME TRAFFIC AND ANALYSIS THE SAME

- Shailesh Mohta
- Tanmay Patil

PART A

TOPIC: IMPLEMENTING CONGESTION CONTROL IN TCP

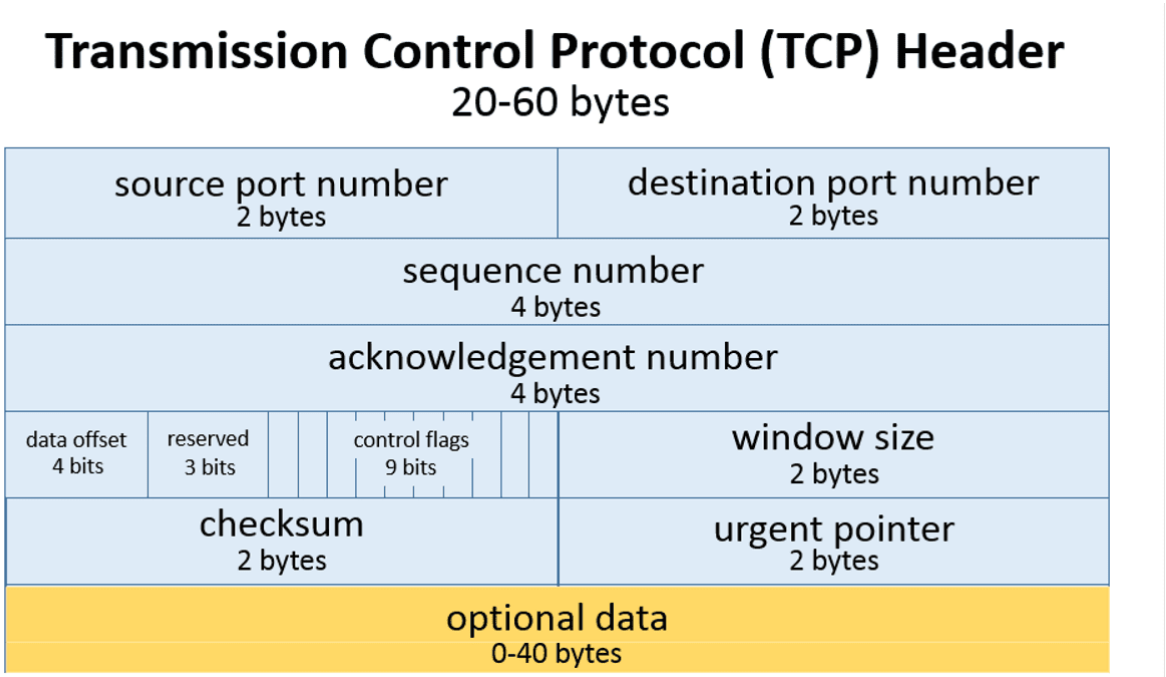
2.1 Introduction to Protocol

TCP is a transport layer protocol used by applications that require guaranteed delivery. It is a sliding window protocol that **provides:**
1. Handling for both timeouts and retransmissions.
2. Communication service at an intermediate level between an application program and the Internet Protocol.
3. Host-to-host connectivity at the Transport Layer of the Internet model.
4. Full duplex virtual connection between two endpoints. Each endpoint is defined by an IP address and a TCP port number.

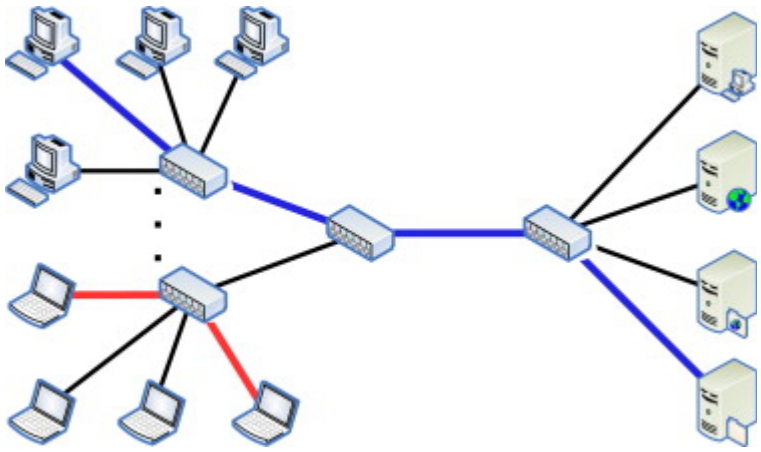
The byte stream is transfered in segments. The window size determines the number of bytes of data that can be sent before an acknowledgement from the receiver is necessary.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests re-transmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, the source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

Below is the header format of TCP:



Major Internet **applications** such as the World Wide Web, email, remote administration, peer-to-peer file sharing and streaming media applications rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.



In TCP, the **congestion window** is one of the factors that determines the number of bytes that can be outstanding at any time. Above diagram is an example in which congestion may occur when all clients sends data at the same time to the server. The congestion window is maintained by the sender. Note that this is not to be confused with the TCP window size which is maintained by the receiver. The congestion window is a means of stopping a link between the sender and the receiver from becoming overloaded with too much traffic. It is calculated by estimating how much congestion there is on the link.

2.2 Evaluation Features / Parameters

TCL SCRIPT FOR CONGESTION CONTROL IN TCP

```
set ns [new Simulator]
set f [ open congestion.tr w ]
$ns trace-all $f
set nf [ open congestion.nam w ]
$ns namtrace-all $nf
$ns color 1 Red
$ns color 2 Blue
$ns color 3 White
$ns color 4 Green
#to create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# to create the link between the nodes with bandwidth, delay and queue
$ns duplex-link $n0 $n2   2Mb  10ms DropTail
$ns duplex-link $n1 $n2   2Mb  10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 200ms DropTail
```

```
$ns duplex-link $n3 $n4 0.5Mb  40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb  30ms DropTail

# Sending node with agent as Reno Agent
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp1
set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp2
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp3
set tcp4 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp4

$tcp1 set fid_ 1
$tcp2 set fid_ 2
$tcp3 set fid_ 3
$tcp4 set fid_ 4

# receiving (sink) node
set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4

# establish the traffic between the source and sink
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
```

```
$ns connect $tcp3 $sink3
$ns connect $tcp4 $sink4

# Setup a FTP traffic generator on "tcp"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP

set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
$ftp4 set type_ FTP

set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
set p1 [new Agent/Ping]
$ns attach-agent $n4 $p1

#Connect the two agents
$ns connect $p0 $p1

# Method call from ping.cc file
Agent/Ping instproc recv {from rtt} {
$self instvar node_
```

```
puts "node [$node_ id] received ping answer from \
$from with round-trip-time $rtt ms."
}
# start/stop the traffic
$ns at 0.2 "$p0 send"
$ns at 0.3 "$p1 send"
$ns at 0.5   "$ftp1 start"
$ns at 0.6   "$ftp2 start"
$ns at 0.7 "$ftp3 start"
$ns at 0.8   "$ftp4 start"
$ns at 66.0 "$ftp4 stop"
$ns at 67.0  "$ftp3 stop"
$ns at 68.0 "$ftp2 stop"
$ns at 70.0 "$ftp1 stop"
$ns at 70.1 "$p0 send"
$ns at 70.2 "$p1 send"

# Set simulation end time
$ns at 80.0 "finish"

# procedure to plot the congestion window
# cwnd_ used from tcp-reno.cc file
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.
  puts  $outfile  "$now $cwnd_"
  $ns at [expr $now+0.1] "plotWindow $tcpSource  $outfile"
}
```

```
set outfile [open  "congestion.xg"  w]
$ns  at  0.0  "plotWindow $tcp1  $outfile"
proc finish {} {
 exec nam congestion.nam &
 exec xgraph congestion.xg -geometry 300x300 &
exit 0
}
# Run simulation
$ns run
```

1. Congestion window

-- TCP and its Algorithms (Slow-Start, Congestion Avoidance, Fast Retransmit and Fast Recovery) :

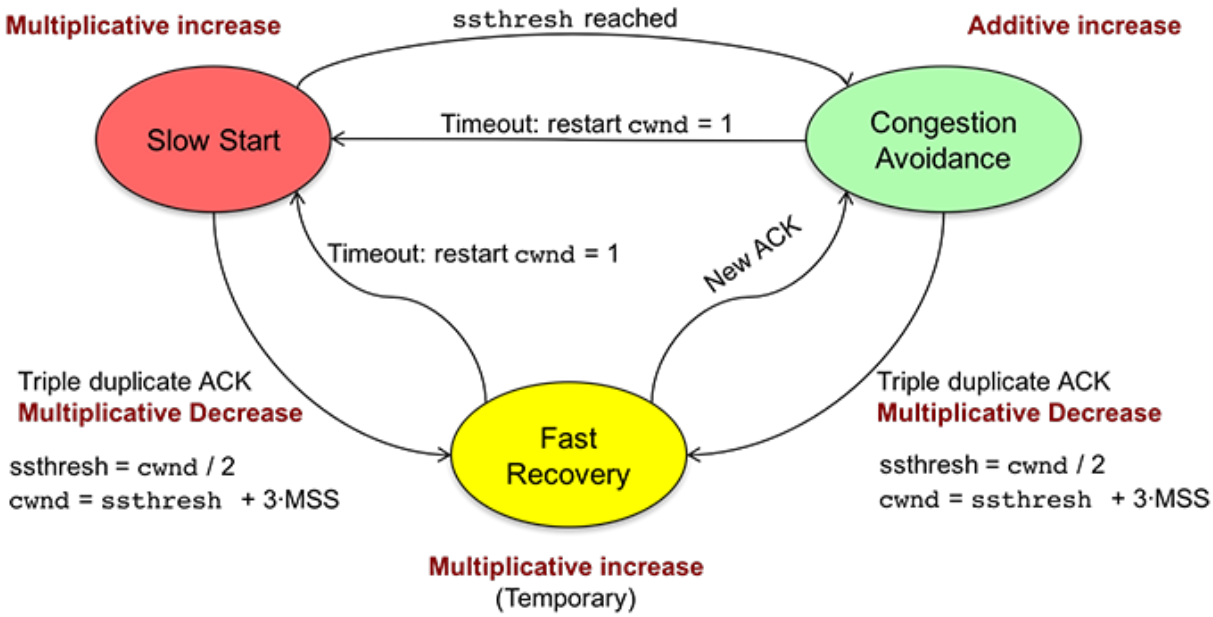
TCP is a complex transport layer protocol containing four interwined algorithms: Slow-start, congestion avoidance, fast retransmit and fast recovery.

In **Slow-start phase**, TCP increases the congestion window each time an acknowledgement is received, by number of packets acknowledged. This strategy effectively doubles the TCP congestion window for every round trip time (RTT).

When the congestion window exceeds a threshold named **ssthresh**, it enters **congestion avoidance** phase. TCP congestion window is increased by 1 for each RTT until a loss event occurs.

TCP maintains a timer after sending out a packet, if no acknowledgement is received after the timer is expired, the packet is considered as lost. However, this might take too long for TCP to realize a packet is lost and take action. A **fast retransmit algorithm** is proposed to make use of duplicate ACKs to detect packet loss. In fast retransmit, when an acknowledgement packet with the same sequence number is received a specified number of times (normally set to 3), TCP sender is reasonably confident that the TCP packet is lost and will retransmit the packet.

Fast recovery is closely related to fast retransmit. When a loss event is detected by TCP sender, a fast retransmit is performed. If fast recovery is used, TCP sender will not enter slow-start phase, instead it will reduce the congestion window by half, and "inflates" the congestion window by calculating usable window using min(awin, cwnd+ndup), where awin is the receiver's window, cwnd is the congestion window, and ndup is number of dup ACK received. When an acknowledgement of new data (called recovery ACK) is received, it returns to congestion avoidance phase.



Calculation:

Used TCP Reno for window size calculation. cwnd_ variable was used to get the window size from the tcp-reno.cc file.

Starts with slow start, in which:

cwnd = cwnd + 1;

This increases exponentially until packet starts dropping. When packets drops, it enters into congestion avoidance mode, where:

ssthresh = cwnd/2;

Then, **cwnd = cwnd + 1/cwnd;**

i.e., increases almost linearly.

TCP Reno: When triple duplicate ACKs are received, it will halve the congestion window, perform a fast retransmit, and enters fast recovery. If a timeout event occurs, it will enter slow-start. TCP Reno is effective to recover from a single packet loss, but it still suffers from performance problems when multiple packets are dropped from a window of data.

2. Round Trip Time

Round-trip time (RTT), also called round-trip delay, is the time required for a signal pulse or packet to travel from a specific source to a specific destination and back again. This time delay includes the propagation times for the paths between the two communication endpoints.

In the context of computer networks, the signal is generally a data packet, and the RTT is also known as the **ping time**. An internet user can determine the RTT by using the ping command.

Calculation:

Ping Agent was used to calculate RTT. A method of ping.cc called “recv” was being called for the same.

RTT = avg (2 x propagation time)

It helps in calculation of **Timeout,** which is calculated by:

Timeout = RTT + Back-off time

3. Throughput

TCP throughput, which is the rate that data is successfully delivered over a TCP connection, is an important metric to measure the quality of a network connection.
Throughput needs to be maximised.

Calculation:

Throughput (in bits/sec) = Number of Packets received * 8 / RTT

4. Drop Ratio

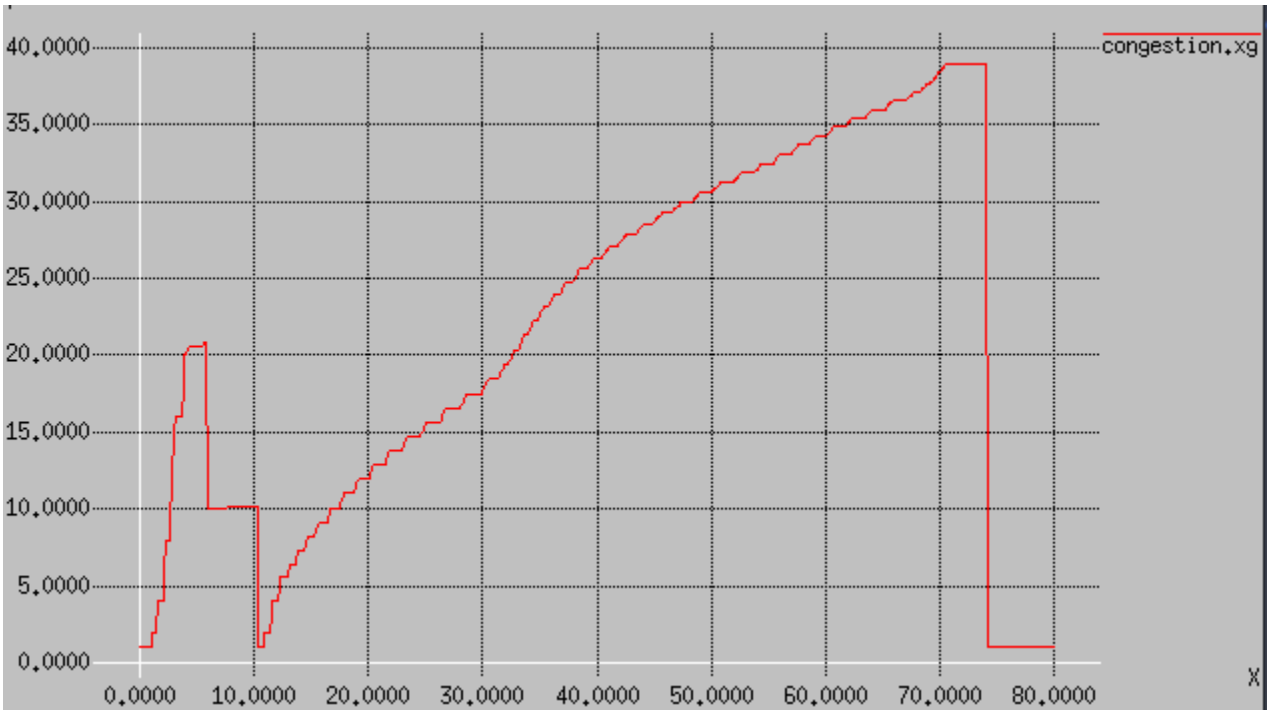
Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is typically caused by network congestion. Packet loss is measured as a percentage of packets lost with respect to packets sent.

Calculation:

Calculated number of packets dropped in different time intervals.

2.3 Graphical Analysis of Protocol

1. Congestion window

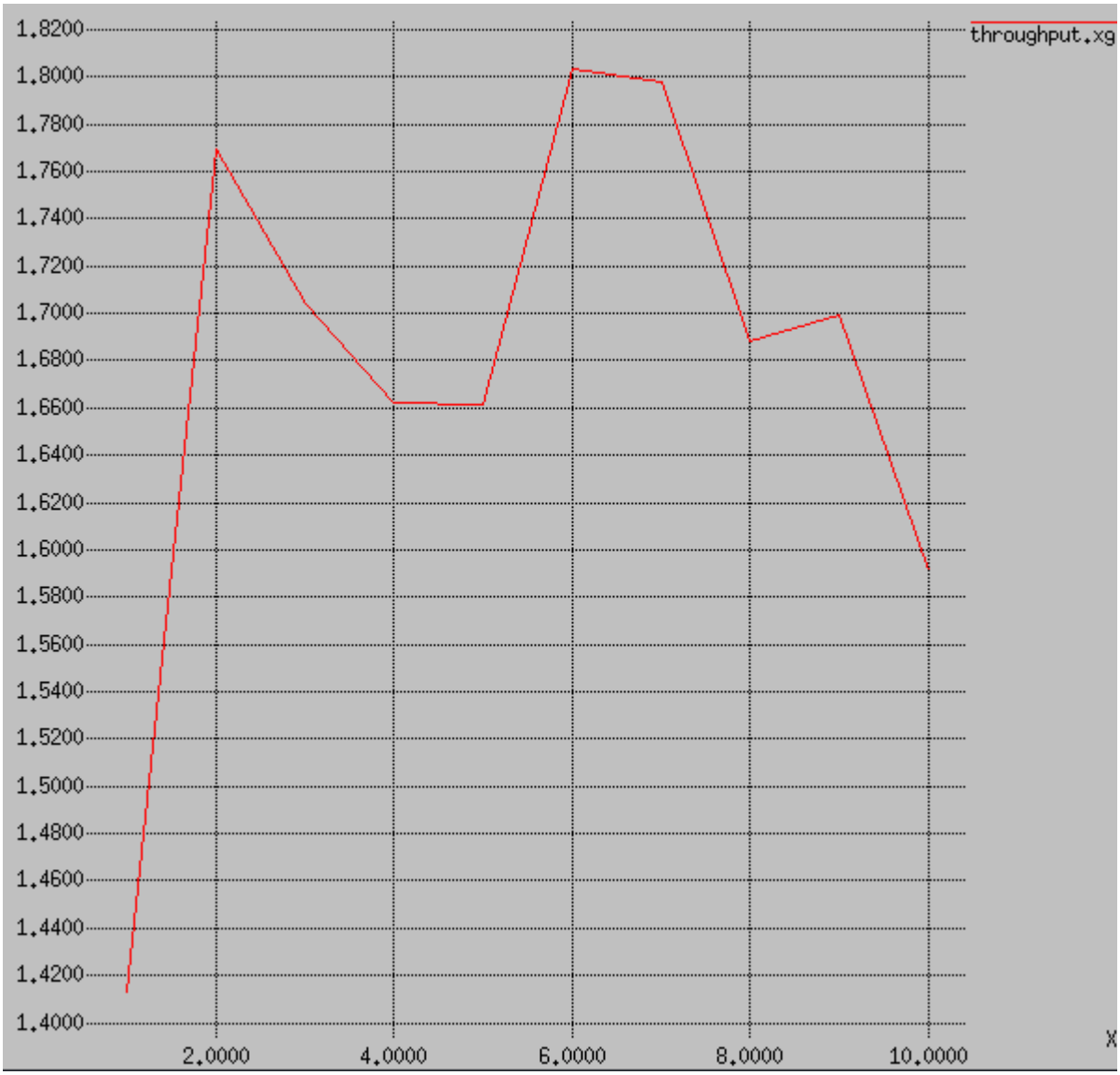


Graph is between cwnd_ v/s time ($now)

Inference:

(i) This graph shows the phase of slow start from 0 to 5,000 (approx).
(ii) It shows drop in window size when packets starts dropping, from 5,000 to 10,000.
(iii) Then from 10,000, approxiamately linear increase in window size is observed (since cwnd = cwnd + 1/cwnd).
(iv) Then window size again drops to '1' when termination signal is sent.

2. Throughput

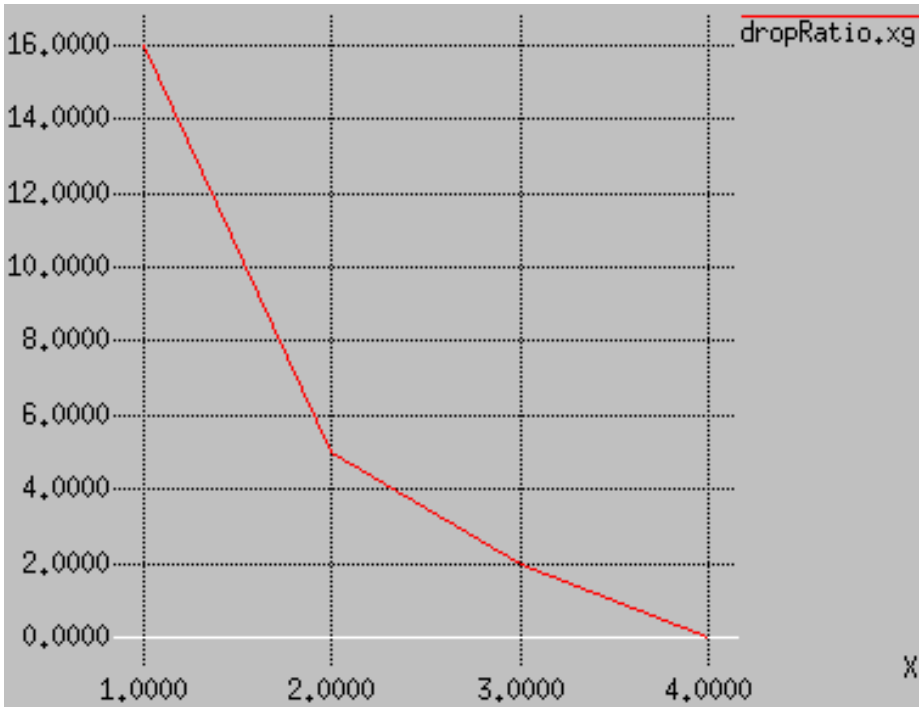


Graph is between throughput (in bits/sec) v/s time (in sec)

Inference:

(i) This graph shows that throughput varies itself according to change in window size. This is so because, when window size crosses threshold value, packets starts dropping, leading to decrease in throughput.
(ii) When window size becomes stable, it is always expected to give maximum throughput value, which in our case is around 1,5900 bits/sec.

3. <u>Drop Ratio</u>



Graph is between Num of packets dropped v/s time (in sec)

Inference:

(i) This graph shows that as congestion window adjusts it's size, it tries to minimize the packets dropped in the process.
(ii) With increase in time, if number of packets dropped would be decreased, then only the connection would be reliable, which is what exactly happening in our case, though not ideal.

PART B

TOPIC: TCL/FTP OVER A N-NODES AD-HOC NETWORK WITH DSDV ROUTING PROTOCOL

2.1 Introduction to DSDV Protocol

Destination Sequenced Distance Vector (DSDV) is a hop-by-hop vector routing protocol requiring each node to periodically broadcast routing updates. This is a table driven algorithm based on modifications made to the Bellman-Ford routing mechanism. Each node in the network maintains a routing table that has entries for each of the destinations in the network and the number of hops required to reach each of them. Each entry has a sequence number associated with it that helps in identifying stale entries. This mechanism allows the protocol to avoid the formation of routing loops. Each node periodically sends updates tagged throughout the network with a monotonically increasing even he number to advertise its location. New route broadcasts contain the address of the destination, the number of hops to reach the destination, the sequence number of the information received regarding the destination, as well as a new sequence number unique to the broadcast. The route labeled with the most recent sequence number is always used. When the neighbors of the transmitting node receive this update, they recognize that they are one hop away from the source node and include this information in their distance vectors. Every node stores the “next routing hop” for every reachable destination in their routing table. The route used is the one with the highest sequence number i.e. the most recent one. When a neighbor B of A finds out that A is no longer reachable, it advertises the route to A with an infinite metric and a sequence number one greater than the latest sequence number for the route forcing any nodes with B on the path to A, to reset their routing tables.

Routing table updates in DSDV are distributed by two different types of update packets:

- Full dump: This type of update packet contains all the routing information available at a node. As a consequence, it may require several Network Protocol Data Units (NPDUs) to be transferred if the routing table is large. Full dump packets are transmitted infrequently if the node only experiences occasional movement.
- Incremental: This type of update packet contains only the information that has changed since the latest full dump was sent out by the node. Hence, incremental packets only consume a fraction of the network resources compared to a full dump.

The routing table is in the form of the following given below-

DESTINATION	METRICS	NEXT HOP	SEQUENCE NUMBER

Consider the following graph. Figure 1 shows an example of an ad hoc network before and after the movement of the mobile nodes.

FIG1: AN AD-HOC NETWORK

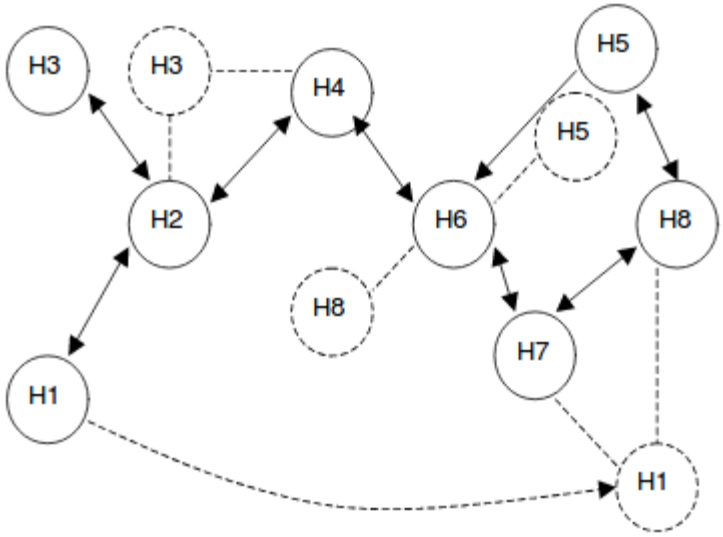


Table 1 is the routing table of the node H6 at the moment before the movement of the nodes. The Install time field in the routing table helps to determine when to delete stale routes.

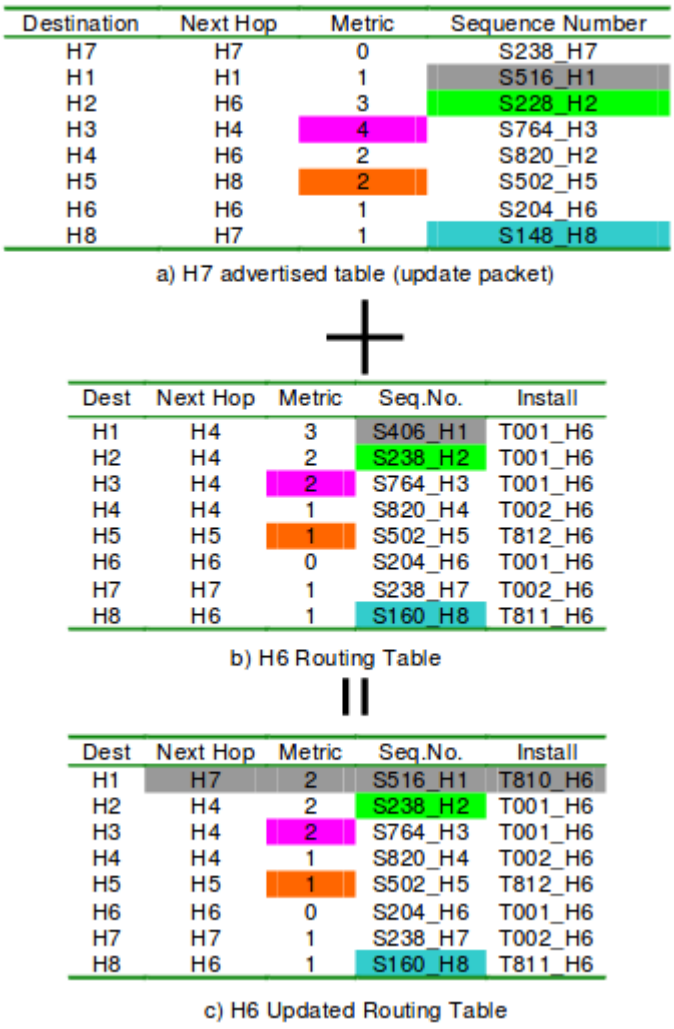
TABLE 1: ROUTING TABLE OF NODE H6

Dest	Next Hop	Metric	Seq.No.	Install
H1	H4	3	S406_H1	T001_H6
H2	H4	2	S128_H2	T001_H6
H3	H4	3	S564_H3	T001_H6
H4	H4	1	S710_H4	T002_H6
H5	H7	3	S392_H5	T001_H6
H6	H6	0	S076_H6	T001_H6
H7	H7	1	S128_H7	T002_H6
H8	H7	2	S050_H8	T002_H6

In the routing information updating process, the original node tags each update packet with a sequence number to distinguish stale updates from the new one. The sequence number is a monotonically increasing number that uniquely identifies each update from a given node. As a result, if a node receives an update from another node, the sequence number must be equal

or greater than the sequence number of the corresponding node already in the routing table, or else the newly received routing information in the update packet is stale and should be discarded. If the sequence number of one node in the newly received routing information update packet is same as the corresponding sequence number in the routing table, then the metric will be compared and the route with the smallest metric will be use.

FIG2: AN EXAMPLE OF UPDATING ROUTE INFORMATION



Destination	Next Hop	Metric	Sequence Number
H7	H7	0	S238_H7
H1	H1	1	S516_H1
H2	H6	3	S228_H2
H3	H4	4	S764_H3
H4	H6	2	S820_H2
H5	H8	2	S502_H5
H6	H6	1	S204_H6
H8	H7	1	S148_H8

a) H7 advertised table (update packet)

Dest	Next Hop	Metric	Seq.No.	Install
H1	H4	3	S406_H1	T001_H6
H2	H4	2	S238_H2	T001_H6
H3	H4	2	S764_H3	T001_H6
H4	H4	1	S820_H4	T002_H6
H5	H5	1	S502_H5	T812_H6
H6	H6	0	S204_H6	T001_H6
H7	H7	1	S238_H7	T002_H6
H8	H6	1	S160_H8	T811_H6

b) H6 Routing Table

Dest	Next Hop	Metric	Seq.No.	Install
H1	H7	2	S516_H1	T810_H6
H2	H4	2	S238_H2	T001_H6
H3	H4	2	S764_H3	T001_H6
H4	H4	1	S820_H4	T002_H6
H5	H5	1	S502_H5	T812_H6
H6	H6	0	S204_H6	T001_H6
H7	H7	1	S238_H7	T002_H6
H8	H6	1	S160_H8	T811_H6

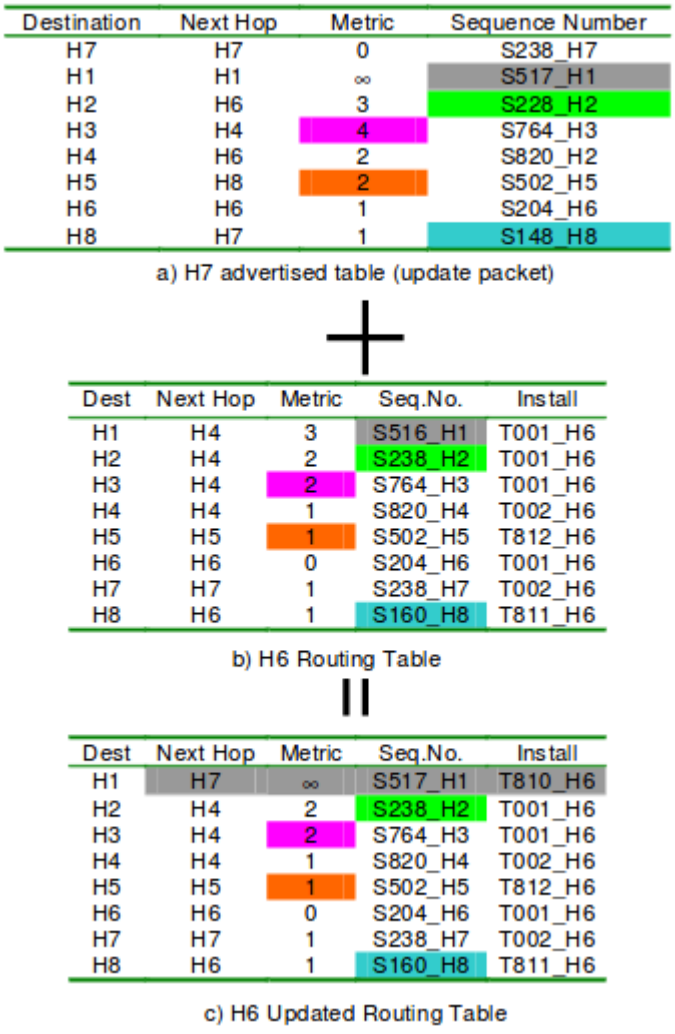
c) H6 Updated Routing Table

The figure shows how actually the routing table of the node H6 is getting updated. Firstly, the sequence number is considered in order to update the routing table. If the sequence numbers are equal then the route with minimum metric value is selected and updating takes place accordingly.

Responding to Topology Changes -

Links can be broken when the mobile nodes move from place to place or have been shut down etc. . The metric of a broken link is assigned infinity. When a link to next hop has broken, any route through that next hop is immediately assigned an infinity metric and an updated sequence number. Because link broken qualifies as a significant route change, the detecting node will immediately broadcast an update packet and disclose the modified routes.

FIG 3: AN EXAMPLE OF LINK BROKEN



Destination	Next Hop	Metric	Sequence Number
H7	H7	0	S238_H7
H1	H1	∞	S517_H1
H2	H6	3	S228_H2
H3	H4	4	S764_H3
H4	H6	2	S820_H2
H5	H8	2	S502_H5
H6	H6	1	S204_H6
H8	H7	1	S148_H8

a) H7 advertised table (update packet)

Dest	Next Hop	Metric	Seq.No.	Install
H1	H4	3	S516_H1	T001_H6
H2	H4	2	S238_H2	T001_H6
H3	H4	2	S764_H3	T001_H6
H4	H4	1	S820_H4	T002_H6
H5	H5	1	S502_H5	T812_H6
H6	H6	0	S204_H6	T001_H6
H7	H7	1	S238_H7	T002_H6
H8	H6	1	S160_H8	T811_H6

b) H6 Routing Table

Dest	Next Hop	Metric	Seq.No.	Install
H1	H7	∞	S517_H1	T810_H6
H2	H4	2	S238_H2	T001_H6
H3	H4	2	S764_H3	T001_H6
H4	H4	1	S820_H4	T002_H6
H5	H5	1	S502_H5	T812_H6
H6	H6	0	S204_H6	T001_H6
H7	H7	1	S238_H7	T002_H6
H8	H6	1	S160_H8	T811_H6

c) H6 Updated Routing Table

Figure 3 illustrates an example of link broken. We assume the link between the node H1 and H7 is broken . ode H7 detects the link broken and broadcasts an update packet (Figure 4a) to node H6. Node H6 updates its routing table with the newly received routing information (odd

sequence number – S517_H1 and ∞ metric) of entry H1. It means that the link to node H1 is broken.

2.2 Evaluation Features / Parameters

TCL Script have been written in order to show the DSDV protocol simulation. By running the nam file the following results have been obtained.

```
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(ant) Antenna/OmniAntenna ;# Antenna type
set val(ll) LL ;# Link layer type
set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type
set val(ifqlen) 50 ;# max packet in ifq
set val(netif) Phy/WirelessPhy ;# network interface type
set val(rp) DSDV ;# ad-hoc routing protocol
set val(nn) 5 ;# number of mobilenodes
set val(mac) Mac/802_11 ;# MAC type
set val(x) 500;
set val(y) 500;
set val(stop) 150;
set ns [new Simulator]
set tracefd       [open dsdv.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open dsdv.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)
```

```
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
        $ns node-config -adhocRouting $val(rp) \
                        -llType $val(ll) \
                        -macType $val(mac) \
                        -ifqType $val(ifq) \
                        -ifqLen $val(ifqlen) \
                        -antType $val(ant) \
                        -propType $val(prop) \
                        -phyType $val(netif) \
                        -channelType $val(chan) \
                        -topoInstance $topo \
                        -agentTrace ON \
                        -routerTrace ON \
                        -macTrace OFF \
                        -movementTrace ON
      for {set i 0} {$i < $val(nn) } { incr i } {
            set node_($i) [$ns node]
      }
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0


$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0
```

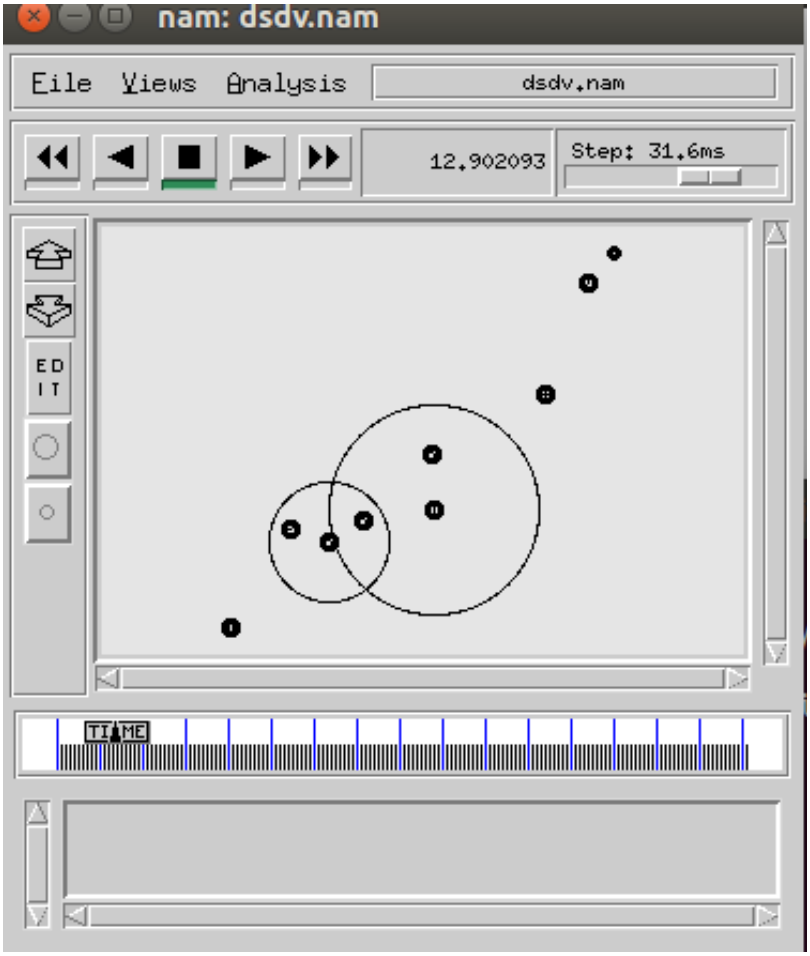
```
$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 320.0
$node_(3) set Y_ 260.0
$node_(3) set Z_ 0.0

$node_(4) set X_ 750.0
$node_(4) set Y_ 560.0
$node_(4) set Z_ 0.0
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"
proc plotWindow {tcpSource file} {
global ns
set time 0.01
```

```
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
exec nam dsdv.nam &
exit 0
}
$ns run
```

FIG 4: RESULTS OF DSDV.tcl SCRIPT



SIMULATION RESULTS

By making changes in the dsdv.cc file in all in one folder, the following results have been evaluated by executing the above tcl script and the results are as follow-

1) Total number of periodic update = 60

2) Printing the updated routing table. The following are the updated roots with source, destination, metrics, next hop, sequence number. The following is the output-

Route table updated..

frm 3 to 1 nxthp 1 [of 1] of sequence number[4]

frm 3 to 2 nxthp 2 [of 1] of sequence number[4]

frm 2 to 3 nxthp 3 [of 1] of sequence number[4]

frm 1 to 3 nxthp 3 [of 1] of sequence number[4]

frm 2 to 1 nxthp 3 [of 2] of sequence number[4]

frm 2 to 3 nxthp 3 [of 1] of sequence number[6]

frm 1 to 2 nxthp 3 [of 2] of sequence number[4]

frm 1 to 3 nxthp 3 [of 1] of sequence number[6]

frm 3 to 1 nxthp 1 [of 1] of sequence number[6]

frm 3 to 2 nxthp 2 [of 1] of sequence number[6]

frm 0 to 1 nxthp 2 [of 3] of sequence number[4]

frm 0 to 3 nxthp 2 [of 2] of sequence number[6]

frm 0 to 2 nxthp 2 [of 1] of sequence number[0]

frm 1 to 3 nxthp 3 [of 1] of sequence number[8]

frm 2 to 3 nxthp 3 [of 1] of sequence number[8]

frm 3 to 1 nxthp 1 [of 1] of sequence number[8]

frm 3 to 2 nxthp 2 [of 1] of sequence number[8]

frm 0 to 2 nxthp 2 [of 1] of sequence number[8]

frm 2 to 0 nxthp 0 [of 1] of sequence number[8]

frm 1 to 2 nxthp 3 [of 2] of sequence number[8]

frm 1 to 3 nxthp 3 [of 1] of sequence number[10]

frm 2 to 1 nxthp 3 [of 2] of sequence number[8]

frm 2 to 3 nxthp 3 [of 1] of sequence number[10]

frm 3 to 0 nxthp 2 [of 2] of sequence number[8]

frm 1 to 0 nxthp 2 [of 2] of sequence number[8]

frm 2 to 0 nxthp 0 [of 1] of sequence number[10]

frm 3 to 2 nxthp 2 [of 1] of sequence number[10]

frm 0 to 2 nxthp 2 [of 1] of sequence number[10]

frm 2 to 0 nxthp 0 [of 1] of sequence number[12]

frm 1 to 3 nxthp 3 [of 1] of sequence number[12]

3) The following given below gives the simulation results such as the node from which the update need to take place, the number of entries in the routing table, the number of entries which is to be updated in the routing table and the number of entries which remains unchanged.

For Periodic Update of the packet from 0

Total number of entries in the table=4

Number of entries which is is not to be updated=0

Number of Entries to be updated = 4

For Periodic Update of the packet from 4

Total number of entries in the table=4

Number of entries which is is not to be updated=0

Number of Entries to be updated = 4

For Periodic Update of the packet from 3

Total number of entries in the table=1

Number of entries which is is not to be updated=0

Number of Entries to be updated = 1

For Periodic Update of the packet from 1

Total number of entries in the table=4

Number of entries which is is not to be updated=0

Number of Entries to be updated = 4

TABLE 2: SCENARIO 1 RESULTS OF DIFFERENT PARAMETER BY VARYING THE NUMBER OF NODES

S NO.	NUMBER OF NODES	SEND PACKET	RECEIVE PACKET	PACKET DROPPED	PDR	THROUGH PUT
1	3	10429	10399	30	0.9971	82.86
2	5	11635	11635	29	0.9975	92.48
3	7	11419	11371	48	0.9958	90.61
4	9	9778	9741	37	0.9962	77.62

TABLE 3: SCENARIO 2 RESULTS OF DIFFERENT PARAMETER BY VARYING THE SIMULATION TIME

S NO.	START TIME SIMULATION	END TIME SIMULATION	SEND PACKET	RECEIVE PACKET	PACKET DROPPED	PDR
1	0	150	10429	10399	30	0.9971
2	0	350	10434	10399	35	0.9966
3	0	550	10437	10399	38	0.9964
4	0	750	10440	10399	41	0.9961

AWK Script is generated in order to calculate the number of packets send, number of packets received ,number of packet dropped, actual start and end time of packet delivery, throughput and packet delivery ratio. The following results are generated. Two scenarios are considered in order to obtain the output. The table 2 gives the results of the various parameters by varying the number of nodes of the tcl script for dsdv protocol. The table 2 gives the results of the various parameter by varying the simulation time in the tcl script of the dsdv protocol.

The formula for the parameters are-

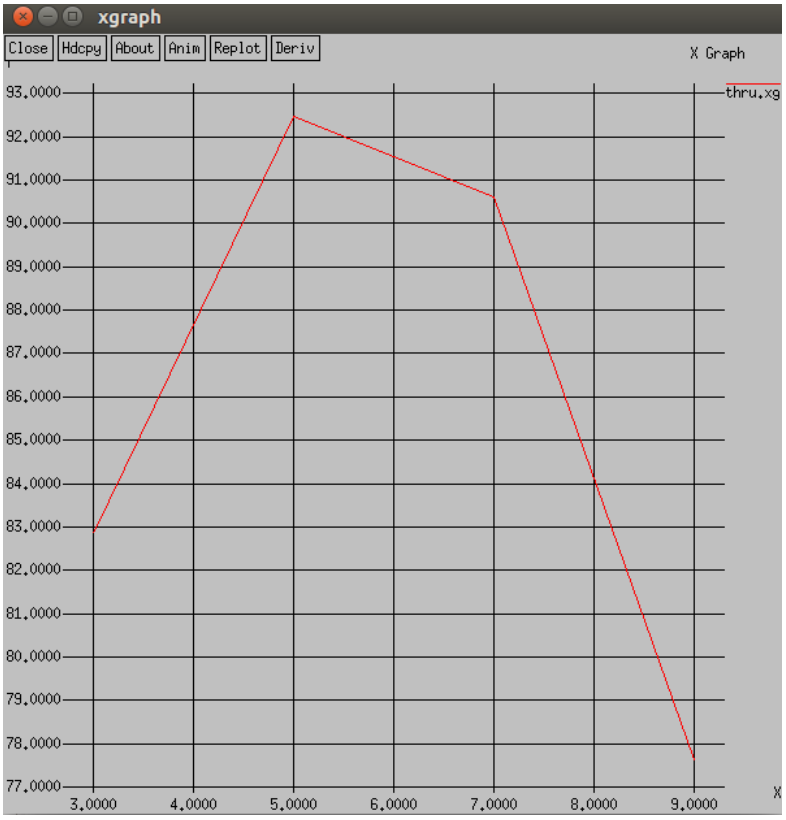
1) Number of packet dropped = Packets Send – Packet Receive

2)Packet delivery Ratio (PDR) =

3) Throughput= recievedsize / simulation time

2.3 Graphical Analysis of DSDV Protocol

FIG 5: NUMBER OF NODES VS AVERAGE THROUGHPUT



By considering the results of the simulation given in section 2.2 the following x-graph are generated. Figure 5 display the x-graph in which the results of the change in average throughout is shown with respect to the increasing number of nodes. As a result we can see that the PDR increases with increase in number of nodes up-to the certain level then decreases gradually.

FIG 6: NUMBER OF NODES VS PDR

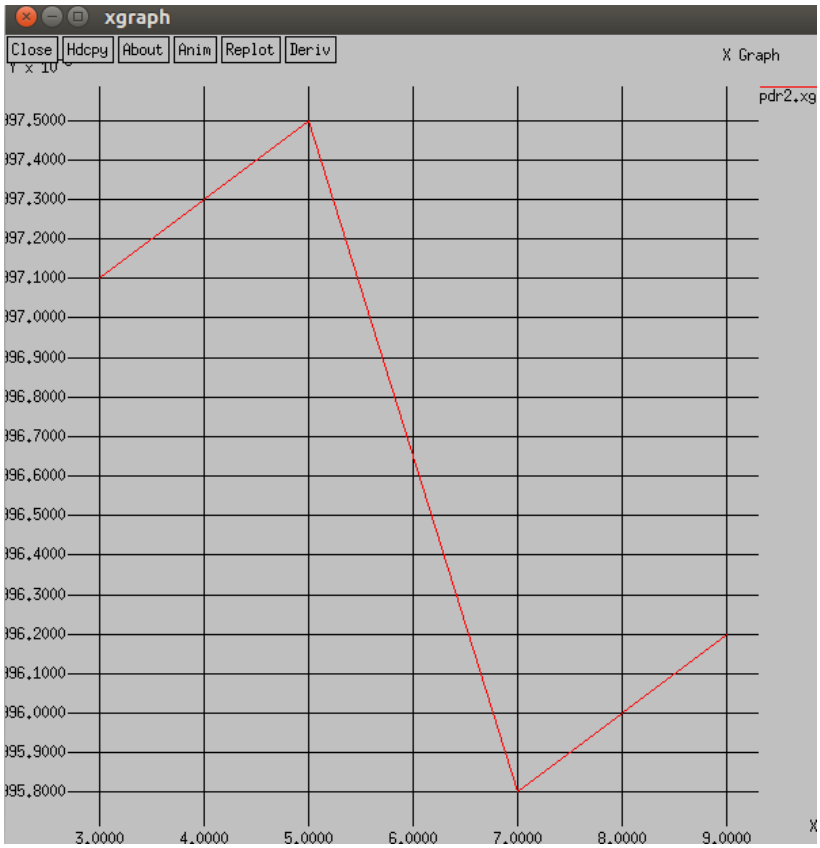


Figure 6 display the x-graph in which the results of the change in PDR is shown with respect to the increasing number of nodes. As a result we can see that the PDR increases with increase in number of nodes up-to the certain level then decreases gradually and then increase.

FIG 7: NUMBER OF NODES VS PACKET DROPPED

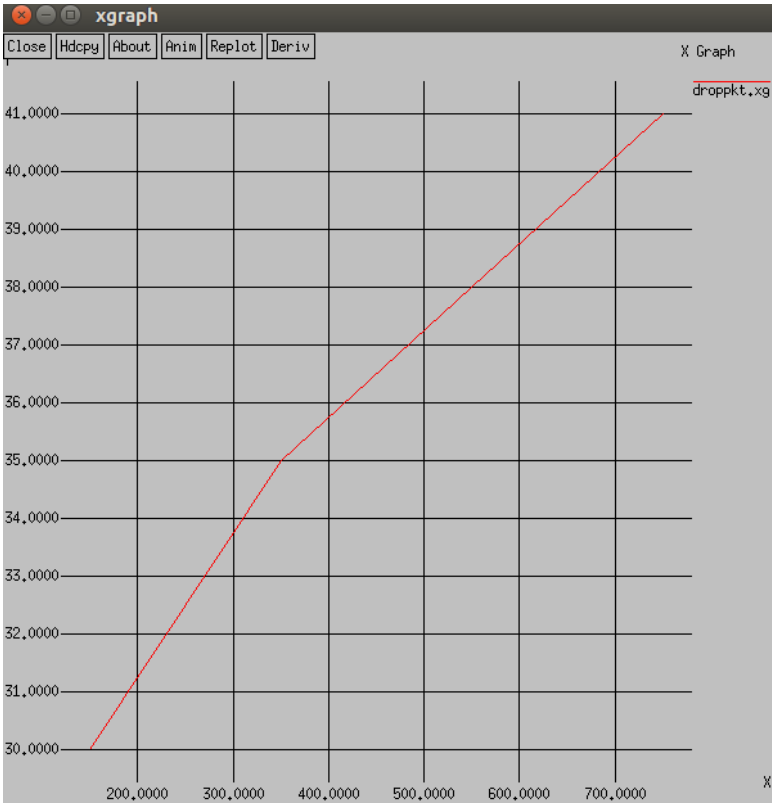


FIG 8: SIMULATION TIME VS NUMBER OF PACKET DROPPED

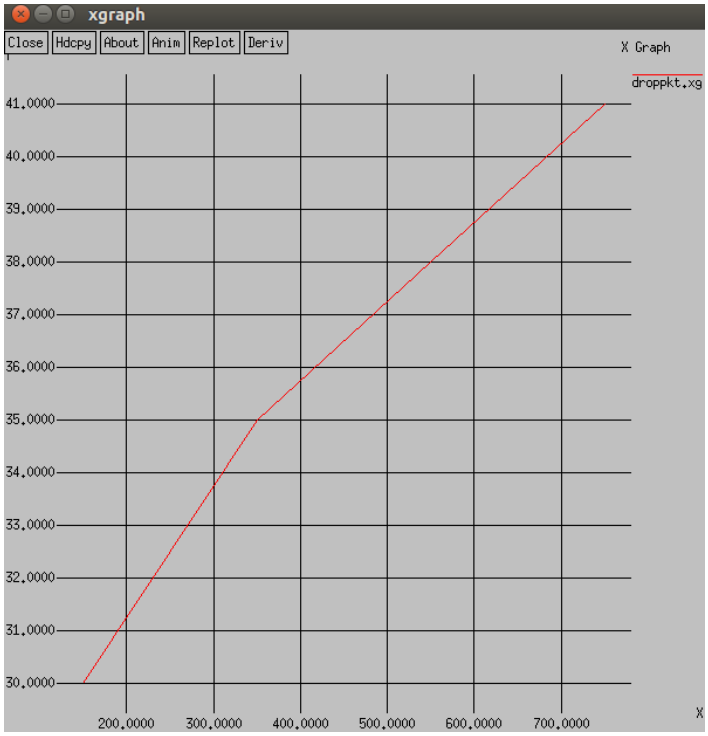


Figure 7 gives the information about the number of packets dropped with increase in the number of nodes. By considering the graph we can say that as the number of nodes increases, more number of packets are dropped.

Figure 8 is the result of scenario 2 in which the simulation time is varied and the behavior of the various parameter have been shown. As we can see, with increase in the simulation time, the number of packets which are dropped have been increasing gradually.

FIG 9: SIMULATION TIME VS PDR

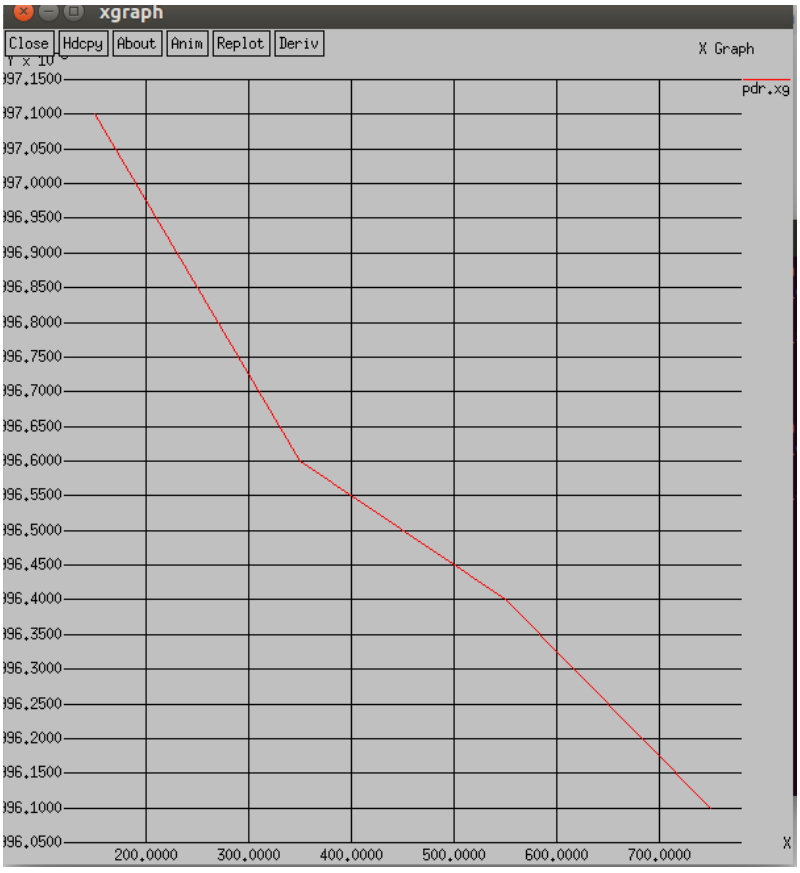


Figure 9 gives the information about the change in PDR value with increase in the simulation time. The results shows that the PDR value gradually decreases with increase in the simulation time.

ADVANTAGES OF DSDV PROTOCOL

- DSDV protocol guarantees loop free paths.
- In DSDV count to infinity problem is reduced which was a major problem in Distance vector protocol.
- Extra traffic can be avoided with incremental updates.
- In routing table, DSDV not maintain multiple paths to destination. A good practice in DSDV is to maintain best paths to a destination only. Because of this space consumed by routing table is reduced.

DRAWBACKS OF DSDV PROTOCOL

DSDV assumes that all wireless links in an ad hoc network are bi-directional. However, this is not true in reality. Wireless media is different from wired media due to its asymmetric connection. Unidirectional links are prevalent in wireless networks.

- The presence of unidirectional links creates the following problems for DSDV protocol-
- Knowledge Asymmetry: Over the unidirectional links, the sink nodes know the existence of the source nodes, but the source nodes cannot assume the existence of the sink nodes.

  Sink Unreachability: In DSDV, the destination node initiates the path updates. Over a

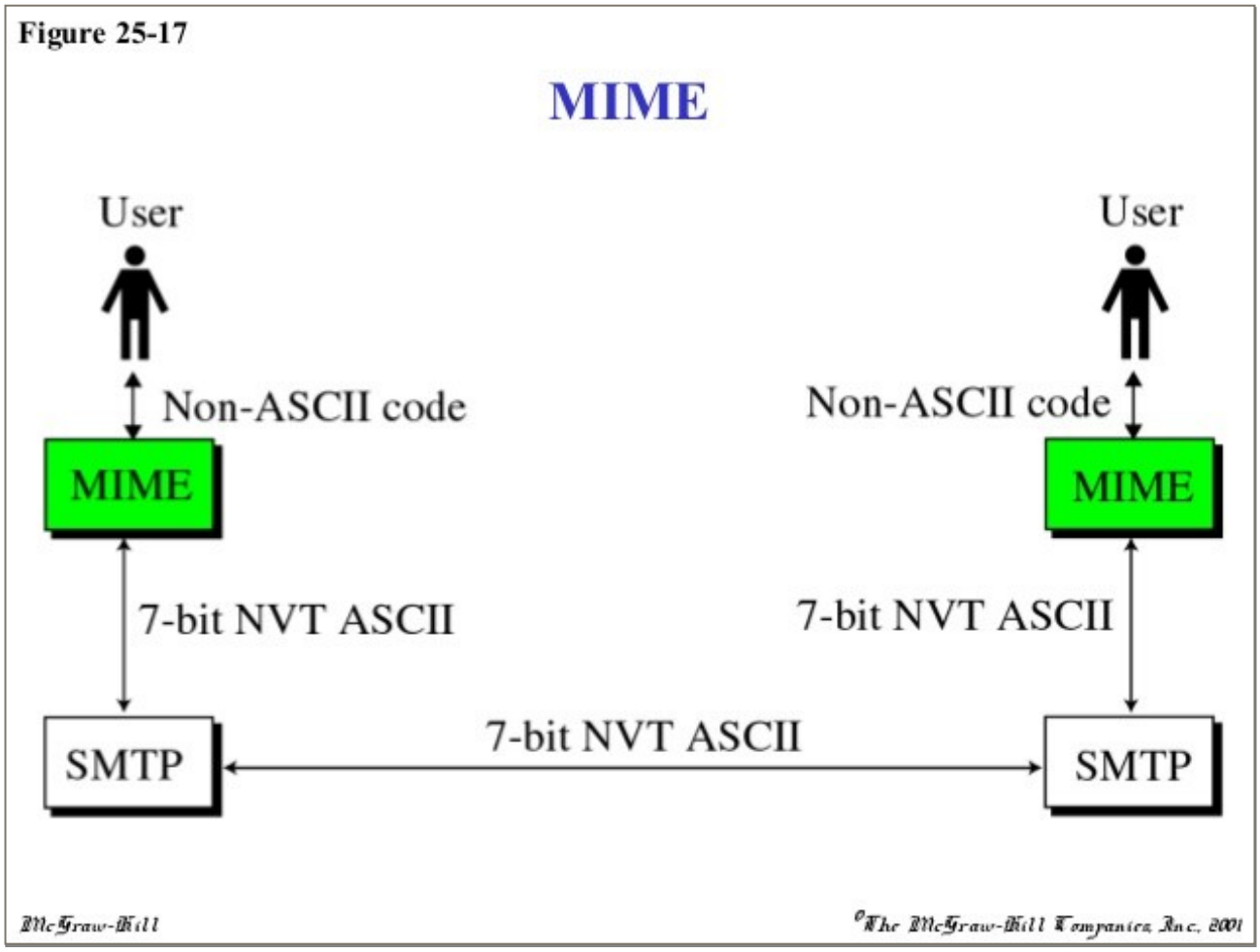
  unidirectional link, there might be no way that a sink node can broadcast its existence.
- It is difficult to determine the maximum setting time.
- DSDV does not support multi-path routing.
- The destination central synchronization suffers from latency problem.
- It has excessive communication overhead due to periodic and triggered updates. Each node must have a complete routing table.

PART C

TOPIC: TCL SCRIPT TO CREATE MIME TRAFFIC AND ANALYSIS THE SAME

2.1 Introduction to Protocol

MIME (Multi-Purpose Internet Mail Extensions) is an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data files on the



Internet: audio, video, images, application programs, and other kinds, as well as the ASCII text handled in the original protocol, the Simple Mail Transport Protocol (SMTP). In 1991, Nathan Borenstein of Bellcore proposed to the IETF that SMTP be extended so that Internet (but mainly Web) clients and servers could recognize and handle other kinds of data than ASCII text. As a result, new file types were added to "mail" as a supported Internet Protocol file type.

The PackMime Internet traffic model was developed by researchers in the Internet Traffic Research group at Bell Labs, based on recent Internet traffic traces. PackMime includes a model of HTTP traffic, called PackMime-HTTP. The traffic intensity generated by PackMime-HTTP is controlled by the rate parameter, which is the average number of new HTTP connections tarted each second. The PackMime-HTTP implementation in ns-2, developed at UNC-Chapel Hill, is capable of generating HTTP/1.0 and HTTP/1.1 (persistent, non-pipelined) connections. The goal of PackMime-HTTP is not to simulate the interaction between a single web client and web server, but to simulate the TCP-level traffic generated on a link shared by many web clients and servers. A typical PackMime-HTTP instance consists of two ns nodes: a server node and a client node. It is important to note that these nodes do not correspond to a single web server or web client. A single PackMime-HTTP client node generates HTTP connections coming from a "cloud" of web clients. Likewise, a single PackMime-HTTP server node accepts and serves HTTP connections destined for a "cloud" of web servers. A single web client is represented by a single PackMime-HTTP client application, and a single web server is represented by a single PackMime-HTTP server application. There are many client applications assigned to a single client ns node, and many server applications assigned to a single server ns node. PackMimeHTTP is an ns object that drives the generation of HTTP traffic. Each PackMimeHTTP object controls the operation of two types of Applications, a PackMimeHTTP server Application and a PackMimeHTTP client Application. Each of these Applications is connected to a TCP Agent (Full-TCP). Note: PackMime-HTTP only supports Full-TCP agents. Each web server or web client cloud is represented by a single ns node that can produce and consume multiple HTTP connections at a time. For each HTTP connection, PackMimeHTTP creates (or allocates from the inactive pool, as described below) server and client Applications and their associated TCP Agents.

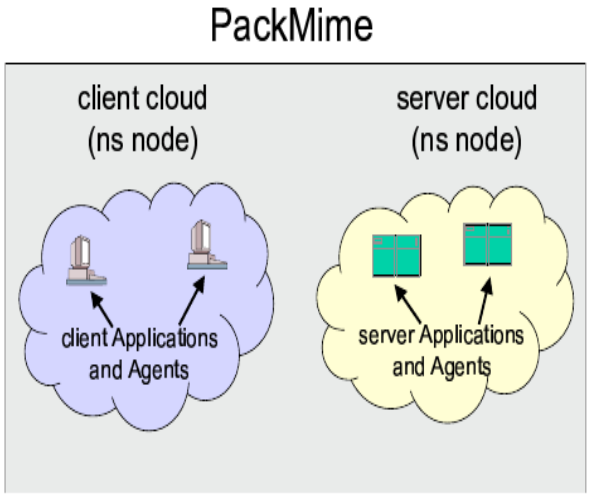
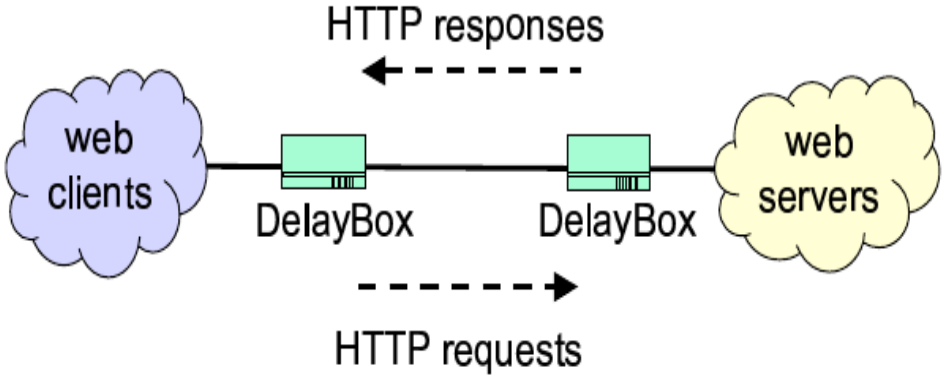


Figure 42.1: PackMimeHTTP Architecture. Each PackMimeHTTP object controls a server and a client cloud. Each cloud can represent multiple client or server Applications. Each Application represents either a single web server or a single web client.

PackMimeHTTP sets a timer to expire when the next new connection should begin. The time between new connections is governed by the connection rate parameter supplied by the user. New connections are started according to the connection arrival times without regard to the completion of previous requests, but a new request between the same client and server pair (as with HTTP 1.1)
begins only after the previous request-response pair has been completed. PackMimeHTTP handles the re-use of Applications and Agents that have completed their data transfer. There are 5 pools used to maintain Applications and Agents – one pool for inactive TCP Agents and one pool each for active and inactive client and server Applications. The pools for active Applications ensure that all active Applications are destroyed when the simulation is finished. Active TCP Agents do not need to be placed in a pool because each active Application contains a pointer to its associated TCP Agent. New objects are only created when there are no Agents or Applications available in the inactive pools.



Example Topology Using PackMimeHTTP and DelayBox. The cloud of web clients is a single ns node, and the cloud of web servers is a single ns node. Each of the DelayBox nodes is a single ns node.

2.2 Evaluation Features / Parameters

Test-packmime-.tcl

```
set rate 5;                        # number of new connections/s
set length 30;                     # length of traced simulation (s)
set window 16;                     # max TCP window size in KB
set bw 100;                        # link speed (Mbps)
set warmup 30;                     # warmup interval (s)
set duration [expr $warmup + $length]; # total simulation time (s)

# useful constants
set CLIENT 0
set SERVER 1

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup Simulator
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

remove-all-packet-headers;          # removes all packet headers
add-packet-header IP TCP;           # adds TCP/IP headers
set ns [new Simulator];             # instantiate the Simulator
$ns use-scheduler Heap;             # use the Heap scheduler

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup Topology
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

#          $bw Mb
#  client  --------  server
```

```
#  cloud     0ms     cloud
#
#   n(0)              n(1)

# create nodes
set n(0) [$ns node]
set n(1) [$ns node]

# create link
$ns duplex-link $n(0) $n(1) ${bw}Mb 0ms DropTail

# set queue buffer sizes (in packets)  (default is 20 packets)
$ns queue-limit $n(0) $n(1) 200
$ns queue-limit $n(1) $n(0) 200

# setup TCP
Agent/TCP/FullTcp set segsize_ 1460;          # set MSS to 1460 bytes
Agent/TCP set window [expr round ($window * 1024.0 / 1500.0)]

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup PackMime
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

set pm [new PackMimeHTTP]
$pm set-client $n(0);                # name $n(0) as client
$pm set-server $n(1);                # name $n(1) as server
$pm set-rate $rate;                  # new connections per second

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup PackMime Random Variables
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
global defaultRNG

# create RNGs (appropriate RNG seeds are assigned automatically)
set flowRNG [new RNG]
set reqsizeRNG [new RNG]
set rspsizeRNG [new RNG]

# create RandomVariables
set flow_arrive [new RandomVariable/PackMimeHTTPFlowArrive $rate]
set req_size [new RandomVariable/PackMimeHTTPFileSize $rate $CLIENT]
set rsp_size [new RandomVariable/PackMimeHTTPFileSize $rate $SERVER]

# assign RNGs to RandomVariables
$flow_arrive use-rng $flowRNG
$req_size use-rng $reqsizeRNG
$rsp_size use-rng $rspsizeRNG

# set PackMime variables
$pm set-flow_arrive $flow_arrive
$pm set-req_size $req_size
$pm set-rsp_size $rsp_size

# record HTTP statistics
$pm set-outfile "data-test-packmime.dat"

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Packet Tracing
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

proc trace {} {
```

```
    global ns n

    # setup packet tracing
    Trace set show_tcphdr_ 1
    set qmonf [open "|gzip > data-test-packmime.trq.gz" w]
    $ns trace-queue $n(0) $n(1) $qmonf
    $ns trace-queue $n(1) $n(0) $qmonf
}

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Cleanup
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

proc finish {} {
    global ns pm reqsizeRNG rspsizeRNG flowRNG req_size rsp_size flow_arrive

    $ns flush-trace

    # delete all of the RNGs and RanVars we created
    delete $reqsizeRNG
    delete $rspsizeRNG
    delete $flowRNG
    delete $req_size
    delete $rsp_size
    delete $flow_arrive

    # delete PackMime
    delete $pm

    # delete Simulator
    delete $ns
```

```
    exit 0
}

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Simulation Schedule
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

$ns at 0.0 "$pm start"
$ns at $warmup "trace"
$ns at $duration "$pm stop"
$ns at [expr $duration + 1] "finish"

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Start the Simulation
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

$ns run
```

Evaluation of Mime is done here on the basis of PDR and Information transfer in different scenarios.

PDR = received information(packets)/time;

A. Minimum header 1 client 1server

A.1) 1Mb max limit

Information sent - 2925

Information Received - 715

PDR -24

A.2) 10Mb max limit

Information sent - 2925

Information Received - 715

PDR - 24

A.3) 100Mb max limit

Information sent - 2933

Information Received - 724

PDR -24

B. MIME without header

B.1) 10 Mb

Send packets = 2925

Received packets=715

PDR= 24

B.2) 100Mb

Send packets=2925

Received packets=715

PDR= 24

C. Analysis with delay box (for testing limited connections)

Send packets=8

Received packets=8

PDR= 48

D. Analysis for 1 clients and 1 server over http 1.1 system

Send packets=4671

Received packets=1315,

PDR= 193

E. Analysis for 2 clients 1server over http 1.1 system

Send packets=4671

Received packets=1431

PDR= 211

F. Analysis for 1 clients and 1 server over http 1.0 system

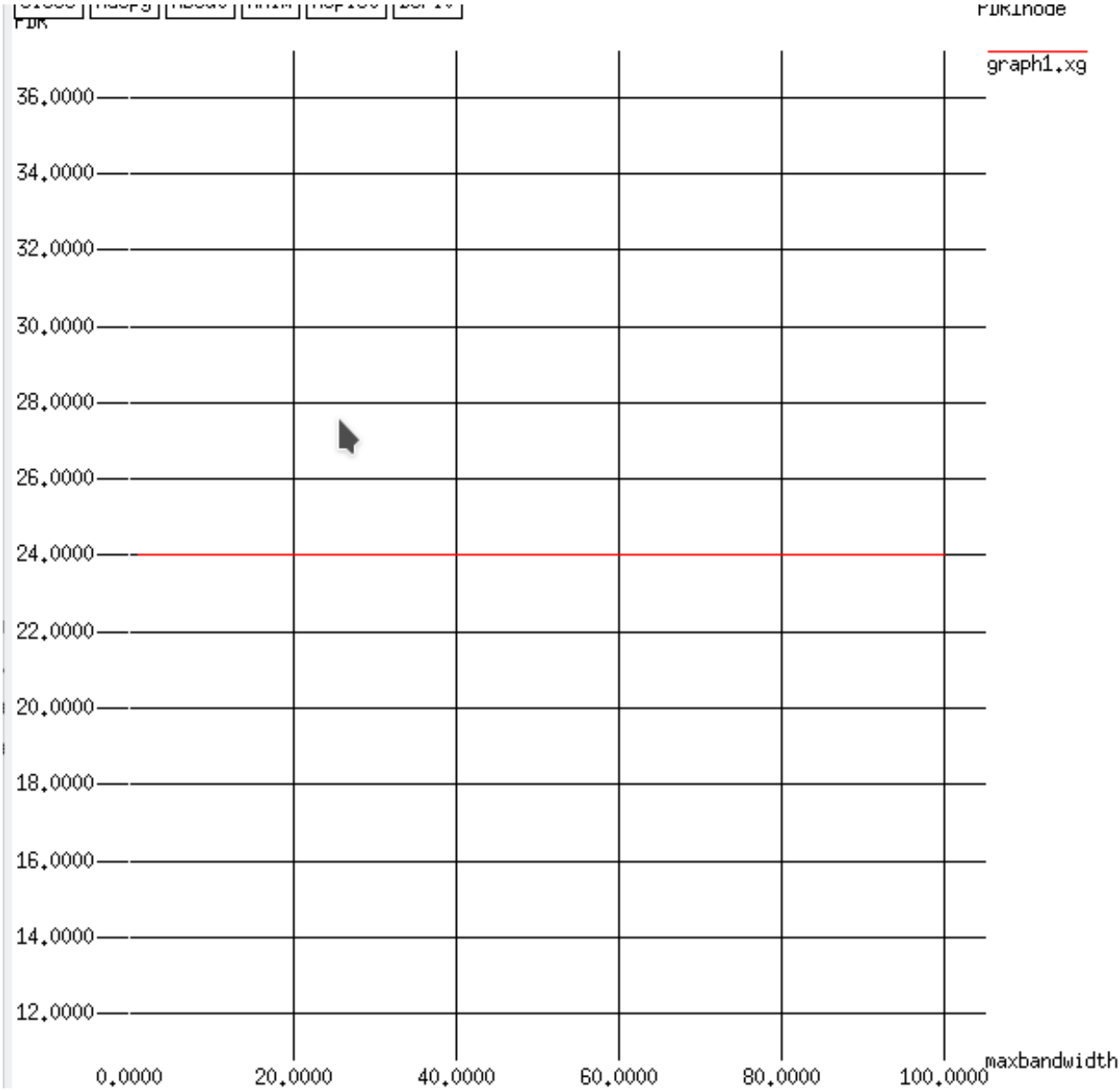
Send packets=30186

Received packets=8441

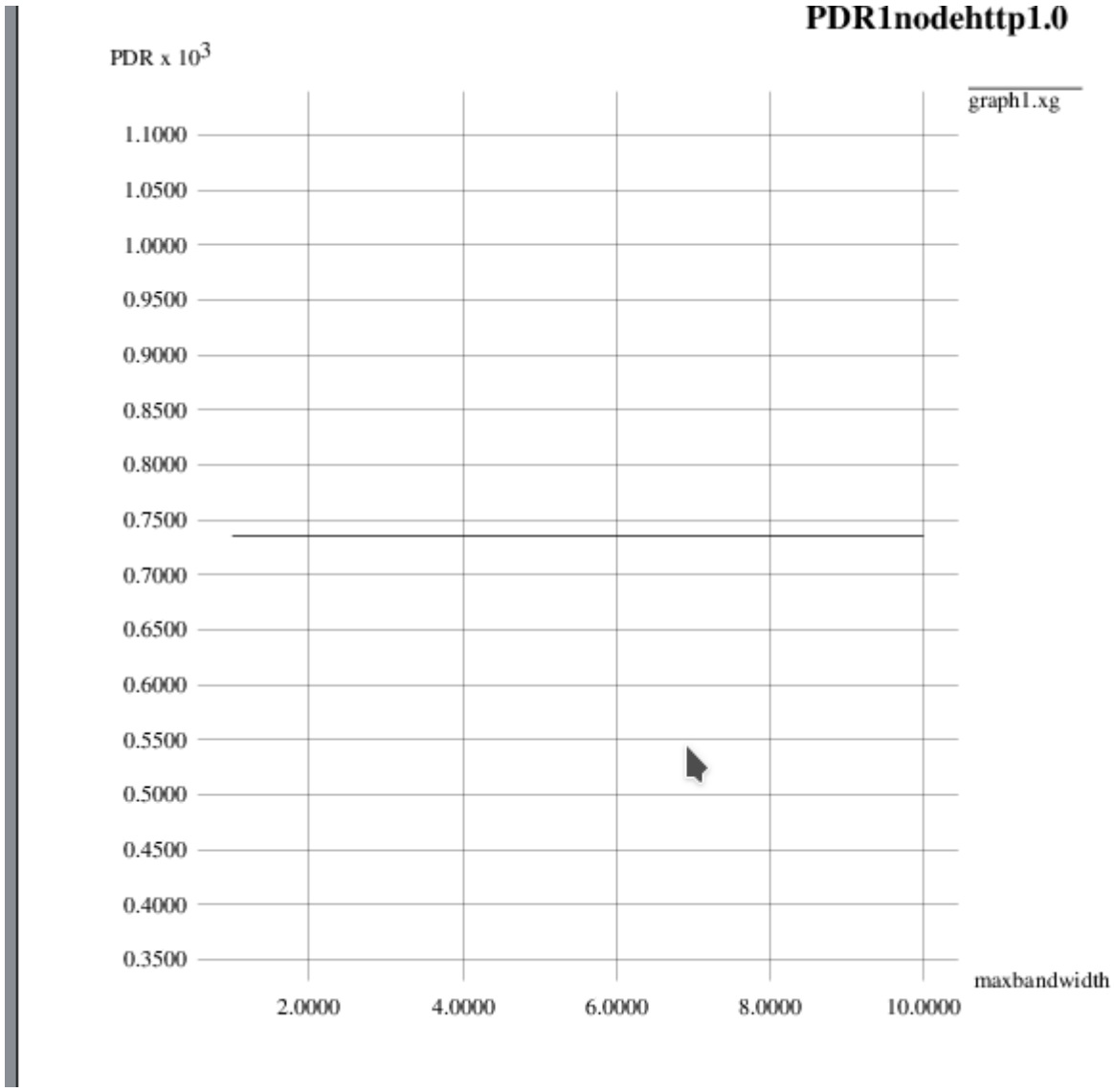
PDR= 735

2.3 Graphical Analysis of Protocol

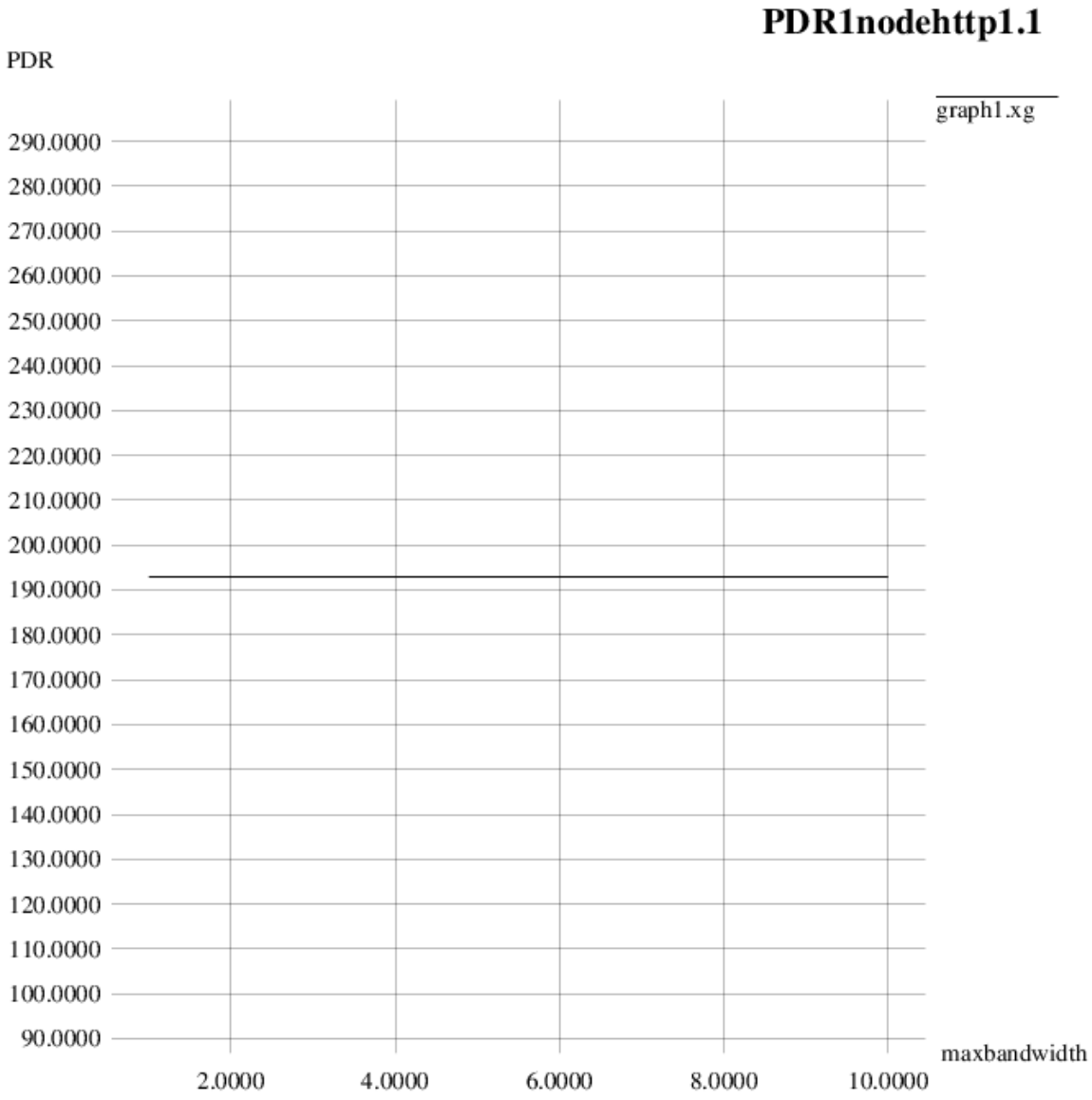
PDR vs max bandwidth for 1 node following tcp.



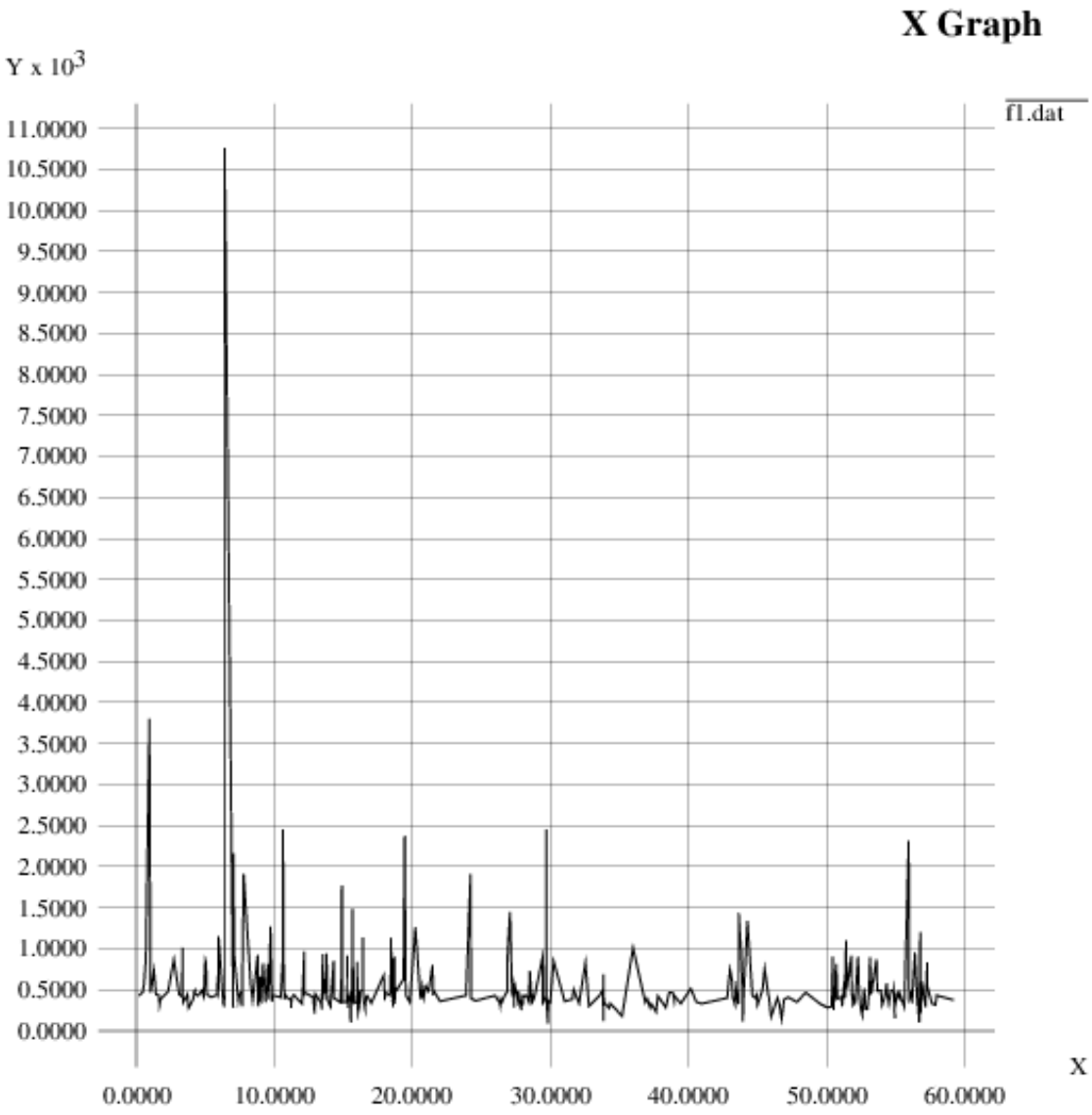
PDR vs max bandwidth for 1 node with http1.0



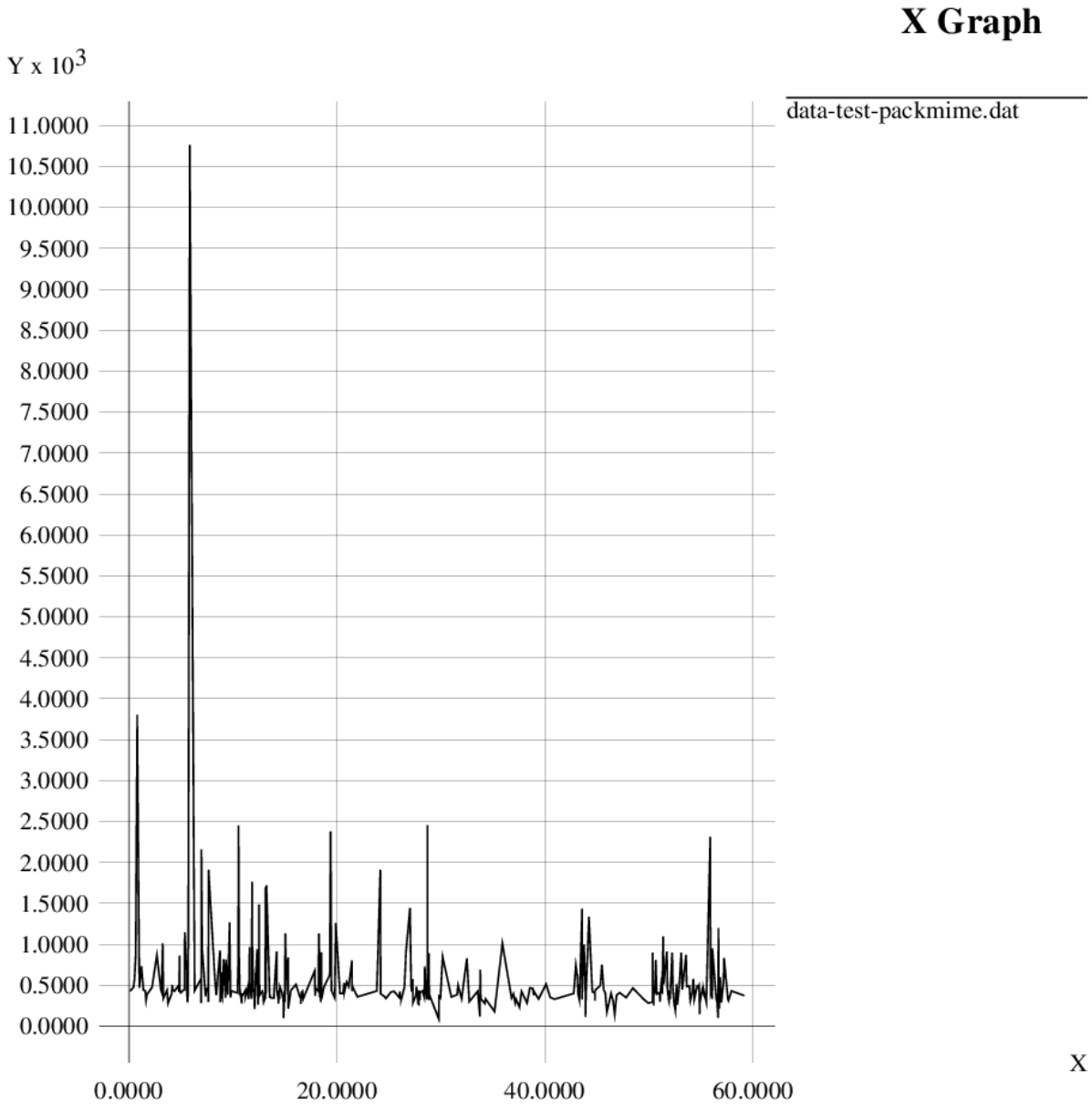
PDR vs max bandwidth for 1 node with http1.1



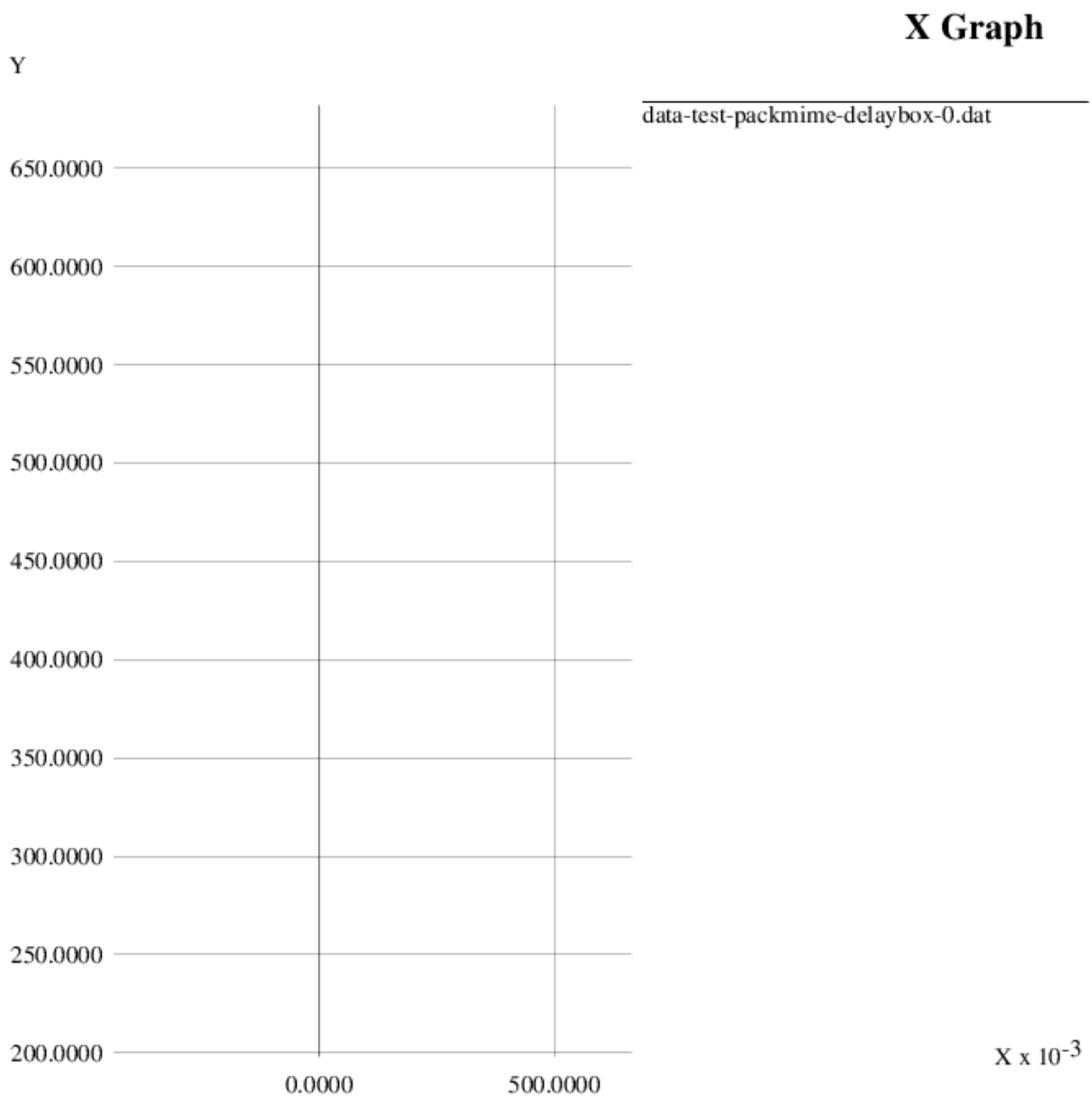
Information flow in 1server 1 client with header



Information flow in MIME without header1 server 1 client (info vs time)



F. Single information flow detected by delay box 1 node 1server tcp protocol. At 0 sec.



AWK CODES

PART A: Implement Congestion control in TCP

1. Throughput.awk

```
BEGIN {
#Initialisation part
}
{
if (seq_no == 0)
		startTime = time;

	if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))   #since
seq_no = -1 for ping
	{
		endTime = time;
		TotalPktRec += pktSize;
	}

	if (time < 35)
	{
		if (seq_no == 0)
			startTime1 = time;

		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
		{
			endTime1 = time;
			TotalPktRec1 += pktSize;
		}

	}
	else if (time >=35 && time < 40)
	{
		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
```

```
		{
			endTime2 = time;
			TotalPktRec2 += pktSize;
		}
	}
	else if (time >=40 && time < 45)
	{
		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
		{
			endTime3 = time;
			TotalPktRec3 += pktSize;
		}
	}
	else if (time >=45 && time < 50)
	{
		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
		{
			endTime4 = time;
			TotalPktRec4 += pktSize;
		}
	}
	else if (time >=50 && time < 55)
	{
		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
		{
			endTime5 = time;
			TotalPktRec5 += pktSize;
		}
	}
	else if (time >=55 && time < 60)
	{
		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
		{
			endTime6 = time;
			TotalPktRec6 += pktSize;
		}
	}
	else if (time >=60 && time < 65)
	{
		if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
		{
```

```
                endTime7 = time;
                TotalPktRec7 += pktSize;
        }
    }
    else if (time >=65 && time <67)
    {
        if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
        {
                endTime8 = time;
                TotalPktRec8 += pktSize;
        }
    }
    else if (time >=67 && time <69)
    {
        if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
        {
                endTime9 = time;
                TotalPktRec9 += pktSize;
        }
    }
    else
    {
        if ((event = "r") && ((to == 3) || (to == 4) || (to == 5)) && (seq_no > 0))
#since seq_no = -1 for ping
        {
                endTime10 = time;
                TotalPktRec10 += pktSize;
        }
    }
}

END {
    throughput = (TotalPktRec * 8)/(endTime - startTime);  #in bits/sec

    throughput1 = (TotalPktRec1 * 8)/(endTime1 - startTime1);  #in bits/sec
    throughput2 = ((TotalPktRec2) * 8)/(endTime2 - 35);  #in bits/sec
    throughput3 = ((TotalPktRec3) * 8)/(endTime3 - 40);  #in bits/sec
    throughput4 = ((TotalPktRec4) * 8)/(endTime4 - 45);  #in bits/sec
    throughput5 = ((TotalPktRec5) * 8)/(endTime5 - 50);  #in bits/sec
    throughput6 = ((TotalPktRec6) * 8)/(endTime6 - 55);  #in bits/sec
    throughput7 = ((TotalPktRec7) * 8)/(endTime7 - 60);  #in bits/sec
    throughput8 = ((TotalPktRec8) * 8)/(endTime8 - 65);  #in bits/sec
    throughput9 = ((TotalPktRec9) * 8)/(endTime9 - 67);  #in bits/sec
    throughput10 = ((TotalPktRec10) * 8)/(endTime10 - 69);  #in bits/sec
```

```
        printf ("\nOverall throughput: %d\n", throughput);

        printf ("\nThroughput by intervals:\n");

        printf ("start-35: %d\n",throughput1);
        printf ("35-40:    %d\n",throughput2);
        printf ("40-45:    %d\n",throughput3);
        printf ("45-50:    %d\n",throughput4);
        printf ("50-55:    %d\n",throughput5);
        printf ("55-60:    %d\n",throughput6);
        printf ("60-65:    %d\n",throughput7);
        printf ("65-67:    %d\n",throughput8);
        printf ("67-69:    %d\n",throughput9);
        printf ("69-end:   %d\n\n",throughput10);
}
```

DropRatio.awk

```
BEGIN {
#Initialisation part
}
{
#
if (event == "d" && from == 2)
                TotalPktDropped ++ ;

        if (time < 20)
        {
                if (event == "d" && from == 2)
                        PktDropped1 ++ ;
        }
        else if (time >=20 && time < 40)
        {
                if (event == "d" && from == 2)
                        PktDropped2 ++ ;
        }
        else if (time >=40 && time < 60)
        {
                if (event == "d" && from == 2)
                        PktDropped3 ++ ;
        }
        else
        {
```

```
                if (event == "d" && from == 2)
                        PktDropped4 ++ ;
        }
}

END {
        printf ("\nOverall Packets dropped: %d\n", TotalPktDropped);

        printf ("\nPackets dropped by intervals:\n");

        printf ("start-20: %d\n",PktDropped1);
        printf ("20-40:    %d\n",PktDropped2);
        printf ("40-60:    %d\n",PktDropped3);
        printf ("60-end:   %d\n",PktDropped4);
}
```

PART B: TCP over a n nodes Ad-hoc network with DSDV routing protocol

PDR.awk File

```
BEGIN {


      droppacket=0;

      sendLine = 0;

      recvLine = 0;

      fowardLine = 0;

      if(mseq==0)

 mseq=10000;

for(i=0;i<mseq;i++){

 rseq[i]=-1;

 sseq[i]=-1;

 }
```

```
}
# Applications received packet
$0 ~/^s.* AGT/ {
# if(sseq[$6]==-1){
        sendLine ++ ;
#        sseq[$6]=$6;
# }
}

# Applications to send packets
$0 ~/^r.* AGT/{
# if(rreq[$6]==-1){
        recvLine ++ ;
#         sseq[$6]=$6;
#        }

}


# Routing procedures to forward the packet
$0 ~/^f.* RTR/ {

       fowardLine ++ ;

}
```

```
# Final output

END {


droppacket=sendLine-recvLine

printf "cbr Send Packets:%d\nrecieve Packets:%d\nPacket Delivery Ratio :%.4f\nNumber of packet forwarded:%d\nNumber of Packet dropped=%d\n", sendLine, recvLine, (recvLine/sendLine),fowardLine,droppacket;


}
```

2) AWK Script to calculate the average throughput-

```
BEGIN {

recvdSize = 0

txsize=0

drpSize=0

startTime = 0

stopTime = 0

thru=0


}


{

event = $1

time = $2

node_id = $3
```

```
pkt_size = $8
level = $4

# Store start time
if (level == "AGT" && event == "s" ) {
if (time < startTime) {
startTime = time
}
# hdr_size = pkt_size % 400
#      pkt_size -= hdr_size
# Store transmitted packet’s size
txsize++;

}

# Update total received packets’ size and store packets arrival time
if (level == "AGT" && event == "r" ) {
if (time > stopTime) {
stopTime = time
}
# Rip off the header
#  hdr_size = pkt_size % 400
# pkt_size -= hdr_size
# Store received packet’s size
recvdSize++
```

```
# thru=(recvdSize/txsize)

# printf(” %.2f %.2f \n” ,time,thru)>”tru2.tr”


}

if (level == "AGT" && event == "D" ) {

# hdr_size = pkt_size % 400

#      pkt_size -= hdr_size

# Store received packet’s size

drpSize++


}

}

END {

printf("Average Throughput[kbps] = %.2f\ns=%.2f\nd=%.2f\nr=%.2f\nStartTime=
%.2f\nStopTime=%.2f\n",(recvdSize/(stopTime-
startTime)),txsize,drpSize,recvdSize,startTime,stopTime)

}
```

PART C:TCL script to create MIME traffic and analysis the same

For HTTP protocols

```
BEGIN {

sendpkt=0;

rcvpkt=0;
```

```
sttime=0;
flag=0;
timee=0;

}
{
event=$1;
time=$2;
from=$3;
to=$4;
flowid=$8;
if( event="+")
{sendpkt++;
if(flag==0)
{sttime=time;
flag=1;

}
}
if( event=="r")
rcvpkt++;

}
END {
timee=time-sttime;
pdr=rcvpkt/timee;
        printf("Send packets=%d\n recieved packets=%d\n , PDR= %d \n",sendpkt,rcvpkt,pdr);
```

```
}
```

For tcp protocols

```
BEGIN {
sendpkt=0;
rcvpkt=0;
sttime=0;
flag=0;
timee=0;

}
{
event=$1;
time=$2;
from=$3;
to=$4;
flowid=$8;
if(from==0 && event="+")
{sendpkt++;
if(flag==0)
{sttime=time;
flag=1;

}
```

```
}
if( event=="r")
rcvpkt++;


}
END {
timee=time-sttime;
pdr=rcvpkt/timee;
        printf("Send packets=%d\n recieved packets=%d\n , PDR= %d \n",sendpkt,rcvpkt,pdr);
}
```

REFERENCE

1. “.cc” files in the ns2 package.

2. Fall, K. and Floyd, S. 1996. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. ACM SIGCOMM Computer Communication Review, 26(3):5-21.

3. Destination-Sequenced Distance Vector (DSDV) Protocol Guoyou He Networking Laboratory Helsinki University of Technology ghe@cc.hut.f