CSC339 – Data Communication & Computer Networks

Name: Mr. Aoun-Haider

ID: FA21-BSE-133

Section: A

Lab Assignment: 3 & 4

CLO:03 Bloom Taxonomy Level: <*Applying*>

Question no. 01 **[Assignment: 03]**

You need to do the following in ns2:

1. As in the paper, start with a network of 10 nodes (0 through 9) placed in a straight line (string topology). The distance between any two neighbouring nodes is 150m, which means that only nearest neighbour nodes can send data to each other. Each node has a queue for packets awaiting transmission, which holds up to 50 packets and is managed in a drop tail fashion. 2. Set up a single TCP connection between source node 0 and destination node 9. TCP packet size is 1460 bytes, 4 packets/sec and the duration of the simulation is 60s. Plot the results of your experiment as shown in Figure 2 of the paper (with window sizes of 4, 8 and 32)

Code Snippet:

```
#Create a ns simulator
set ns [new Simulator]


#Open the NS trace file
set tracefile [open out.tr w]
$ns trace-all $tracefile


#Open the NAM trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile
proc finish {} {
   global ns tracefile namfile
```

```
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam out.nam &
    exit 0
}
```

#Create 10 nodes

#create nodes,set distance and queue limit

```
for {set i 0} {$i < 10} {incr i} {
   set n($i) [$ns node]
   $n($i) set X [expr $i * 150]
}

#create links
for {set i 0} {$i < 9} {incr i} {
 $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail
 $ns duplex-link-op $n($i) $n([expr $i+1]) orient right
 $ns queue-limit $n($i) $n([expr $i+1]) 50
}
```

# Setup a CBR Application over TCP connection

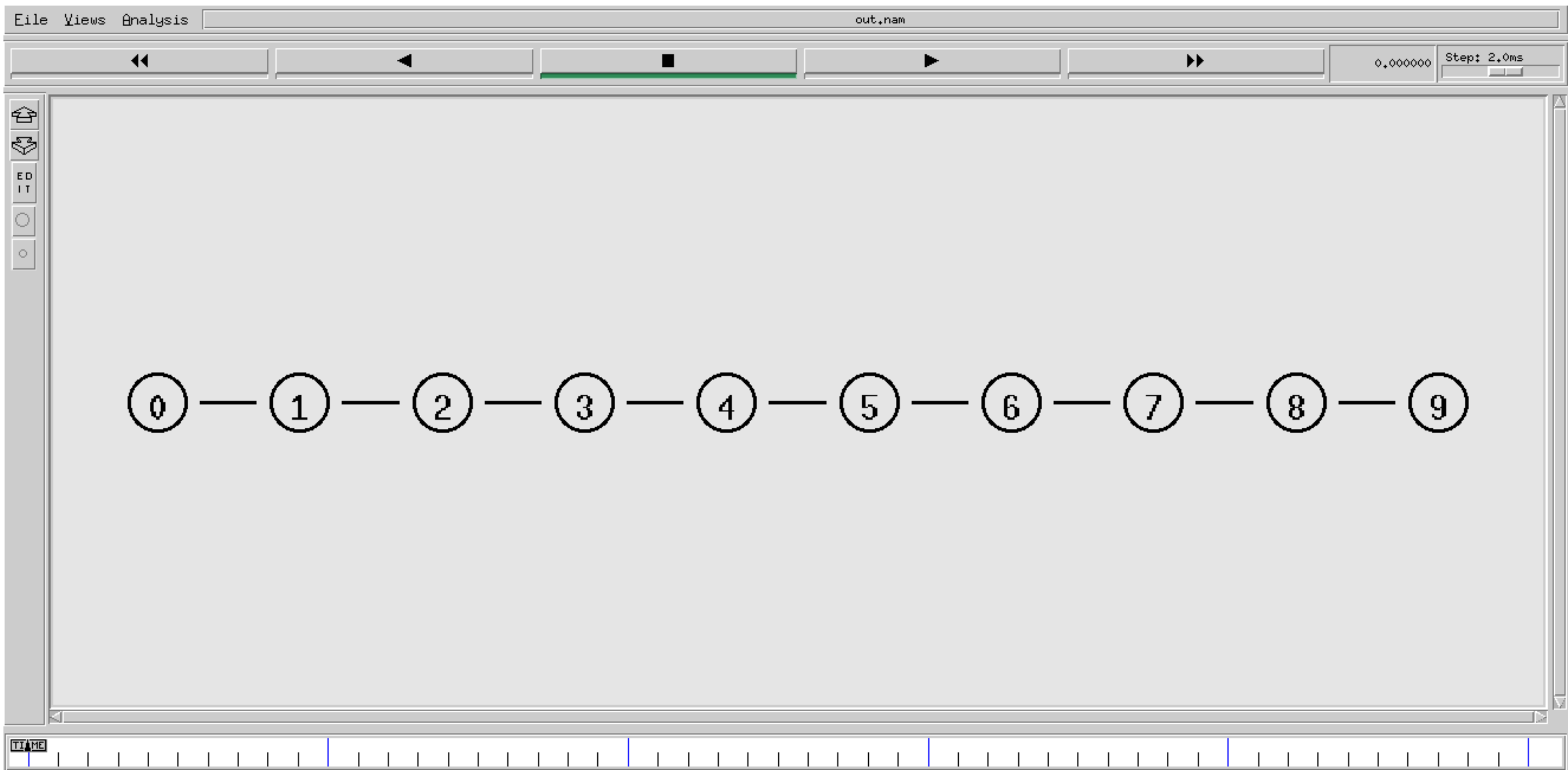
```
set window_size 4
set tcp [new Agent/TCP]
$tcp set window_ $window_size
$ns attach-agent $n(0) $tcp
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1460
$cbr set rate_ 4.0Mb
$cbr attach-agent $tcp
```

```
set null [new Agent/TCPSink]
$ns attach-agent $n(9) $null
$ns connect $tcp $null
```

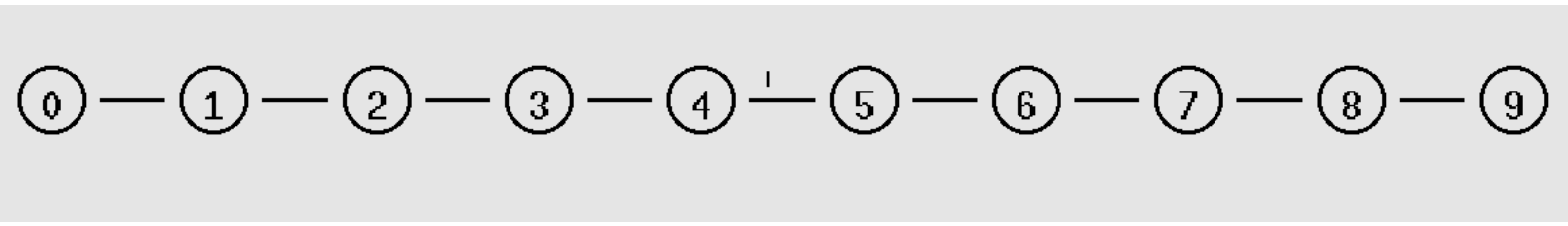
# Start and stop the application

```
$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "finish"
$ns run
```

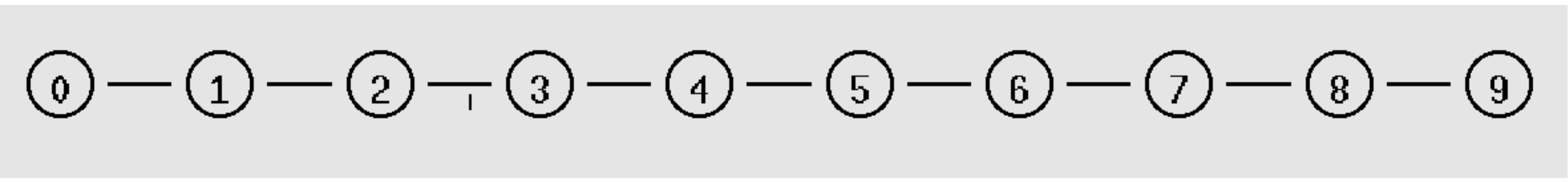
<u>Sample run:</u>



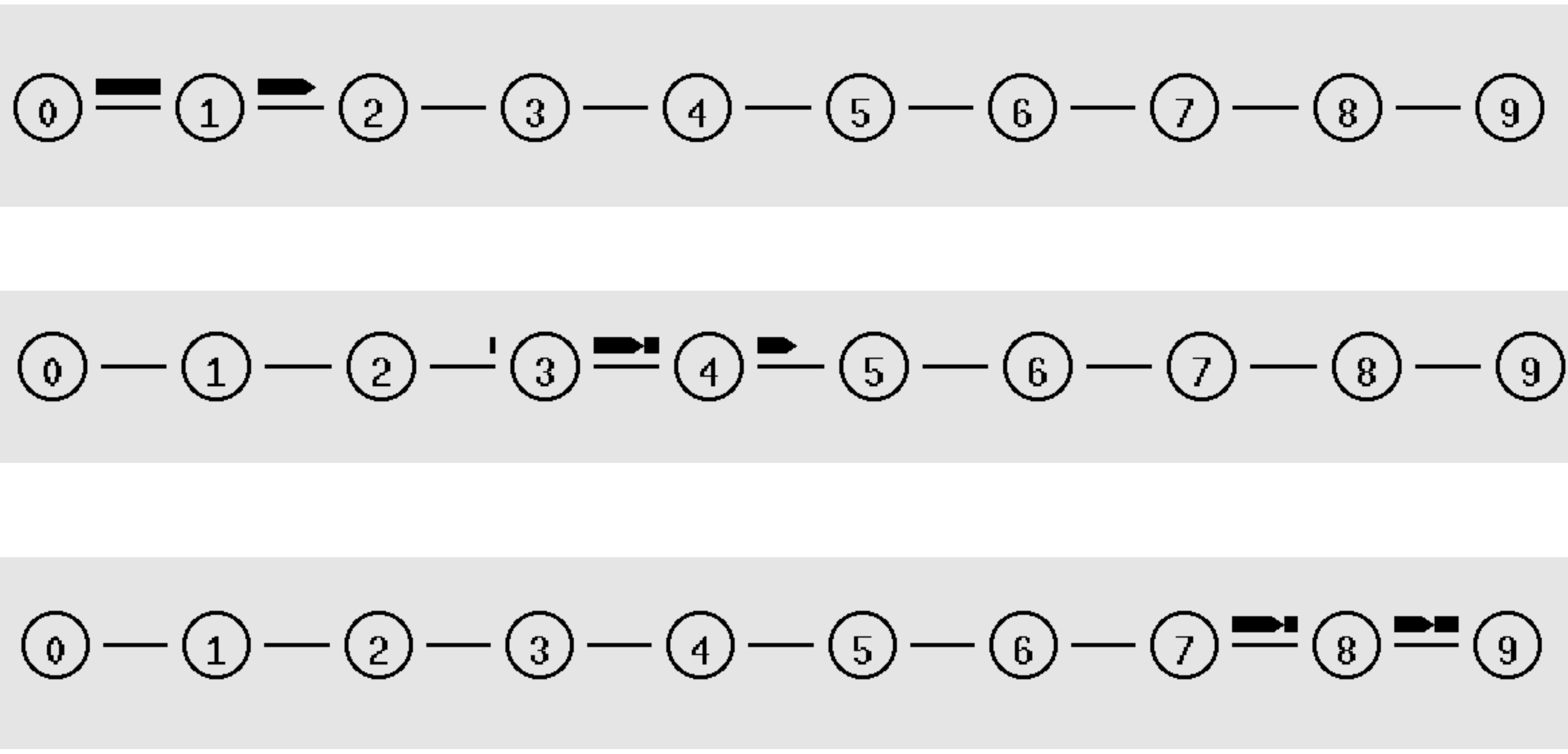
[Window size = 4]



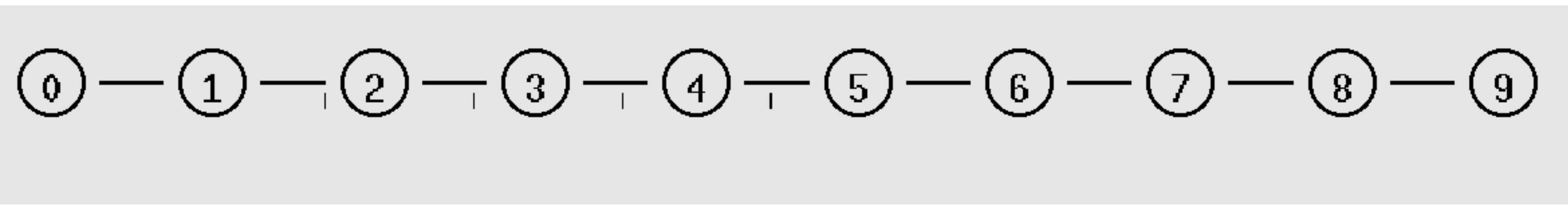
1. Sending SYN segment from n0 to n9



2. Receiving ACK segment from n9 to n0

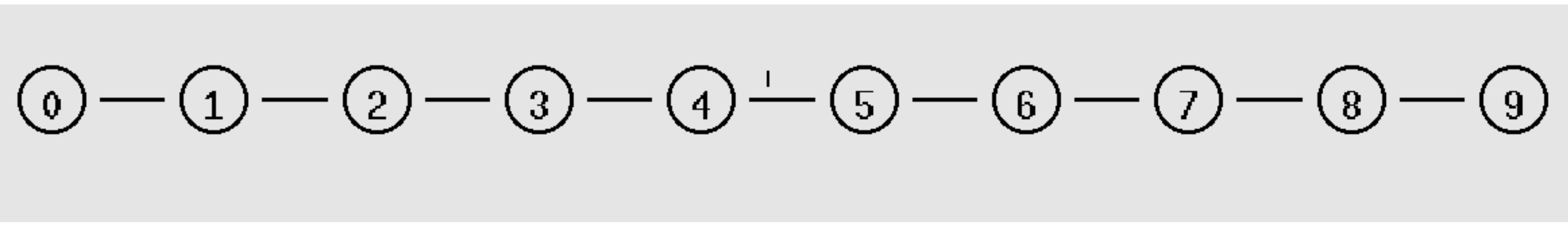


3. Sending Packets of 1460 bytes

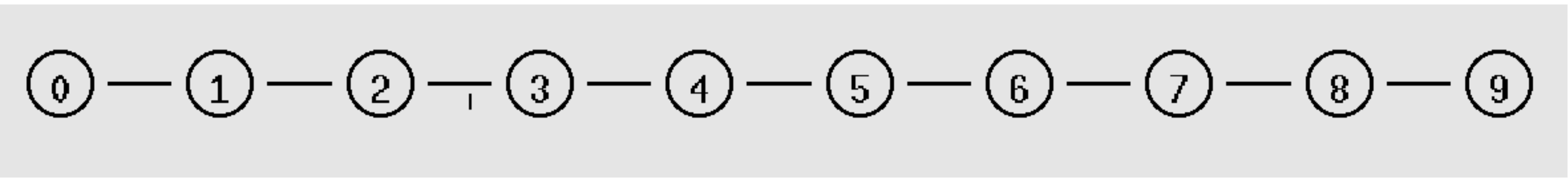


4. Receiving ACK segment for packets

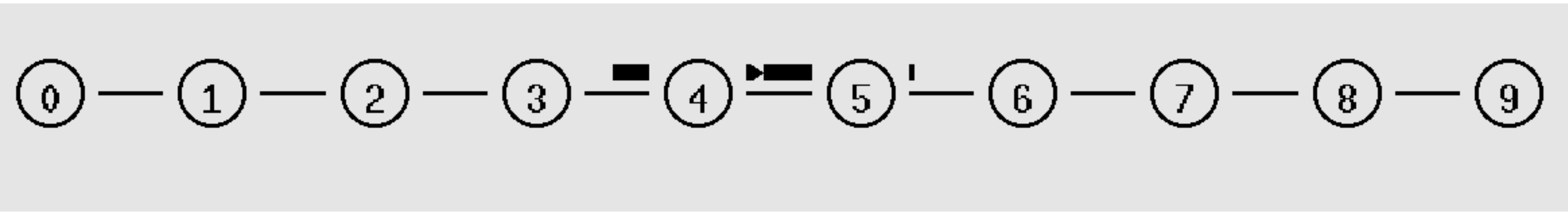
[Window size = 6]



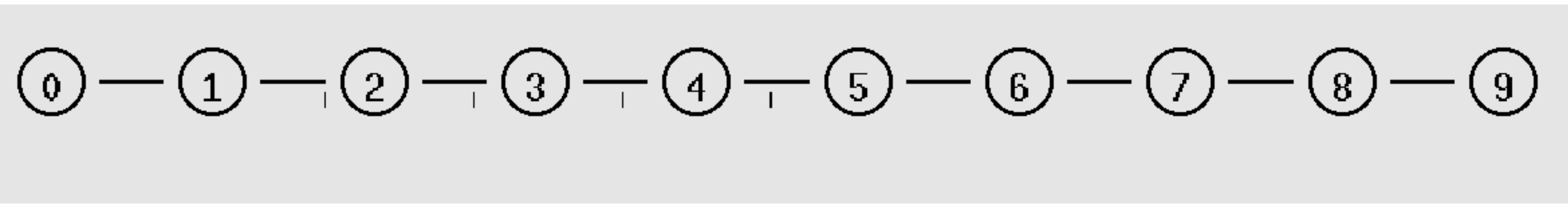
1. Sending SYN segment from n0 to n9



2. Receiving ACK segment from n9 to n0

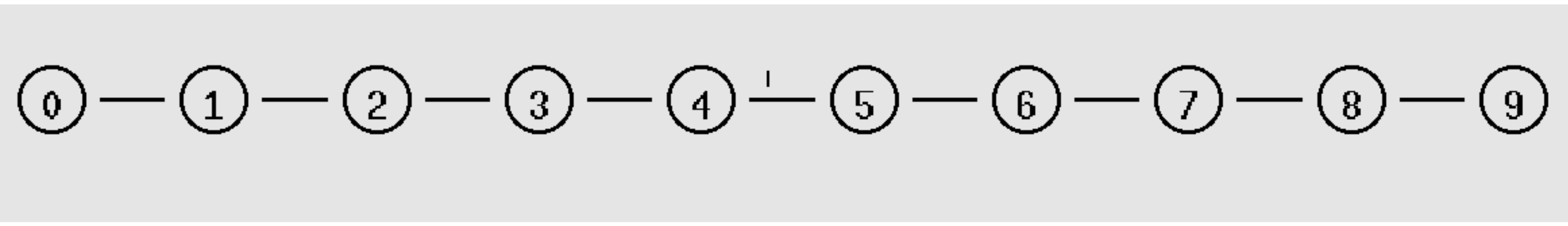


5. Sending packet of 1460 bytes

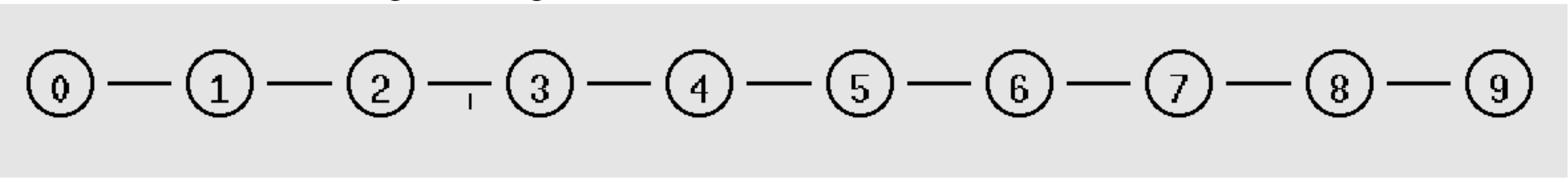


6. Receiving ACK segment for packets

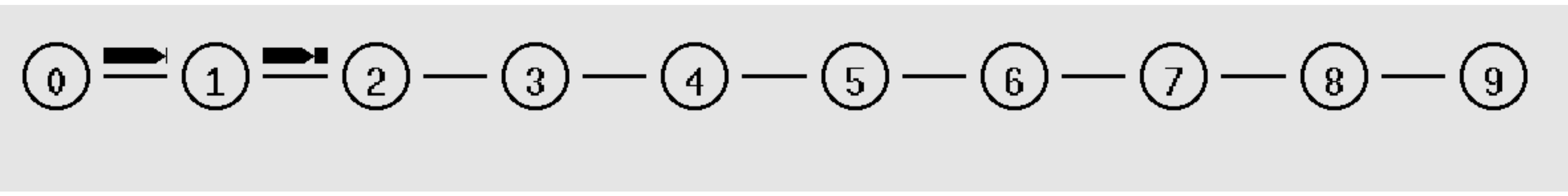
[Window size = 8]



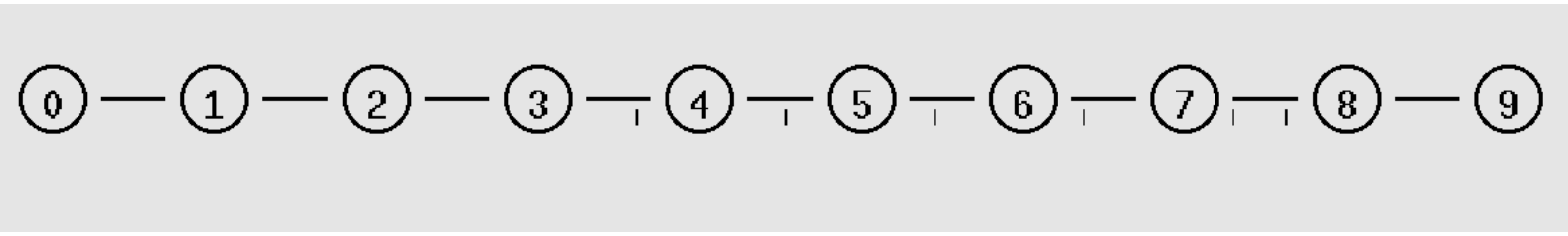
1. Sending SYN segment from n0 to n9



2. Receiving ACK segment from n9 to n0



3. Sending packet of 1460 bytes



4. Receiving ACK segment for packets

Connection b/w n3 and n4

Window size = 4

Code:

#Create a ns simulator

set ns [new Simulator]

#Open the NS trace file

```
set tracefile [open out.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open out.nam w]
$ns namtrace-all $namfile



proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam out.nam &
    exit 0
}
```

#Create 10 nodes

```
#create nodes,set distance and queue limit
for {set i 0} {$i < 10} {incr i} {
   set n($i) [$ns node]
   $n($i) set X [expr $i * 150]
}
```

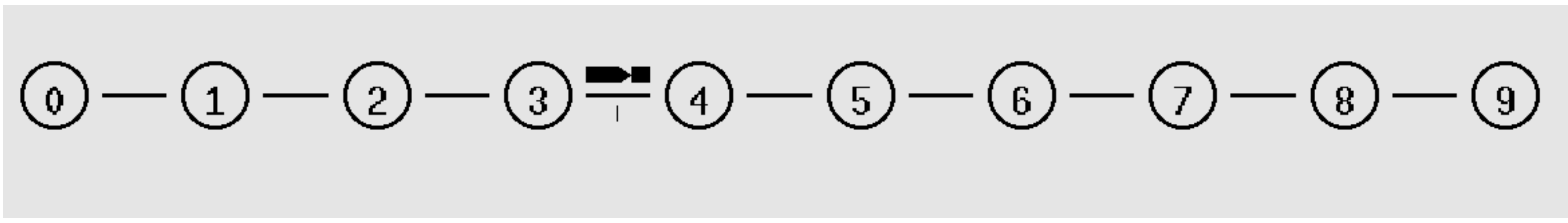
#create links

```
for {set i 0} {$i < 9} {incr i} {
 $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail
 $ns duplex-link-op $n($i) $n([expr $i+1]) orient right
 $ns queue-limit $n($i) $n([expr $i+1]) 50
}
```

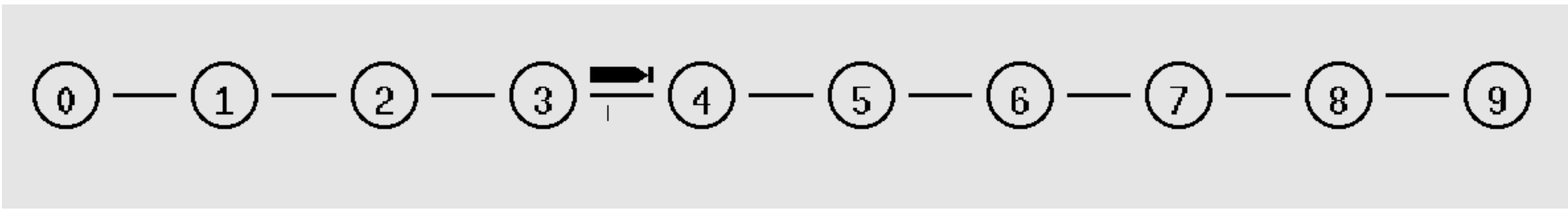
```
# Setup a CBR Application over TCP connection
set window_size 4
set tcp [new Agent/TCP]
$tcp set window_ $window_size
$ns attach-agent $n(3) $tcp

set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 1460
$cbr set rate_ 4.0Mb
$cbr attach-agent $tcp

set null [new Agent/TCPSink]
$ns attach-agent $n(4) $null

$ns connect $tcp $null

# Start and stop the application
$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop"

$ns at 5.0 "finish"
$ns run
```

Sample run:

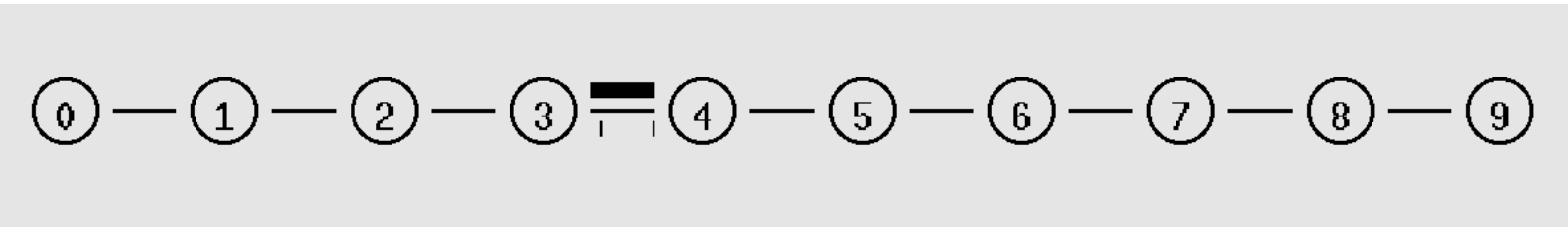
[Window size = 4]



[Window size = 6]



[Window size = 8]



5. Keep window size fixed at 8. Start Two TCP connections. One from node 0 to node 3 and the other from node 6 to 9. Plot the throughput (of both streams) as a function of time. 5. Keep window size fixed at 8. Next, start two TCP connections one from 6 to 4 and the other from 2 to 3. Conduct two experiments. i) let both TCP connections start at the same time. ii) Let the second session start 10 seconds later than the first TCP session. Plot the results as in Figure 4 of the paper.

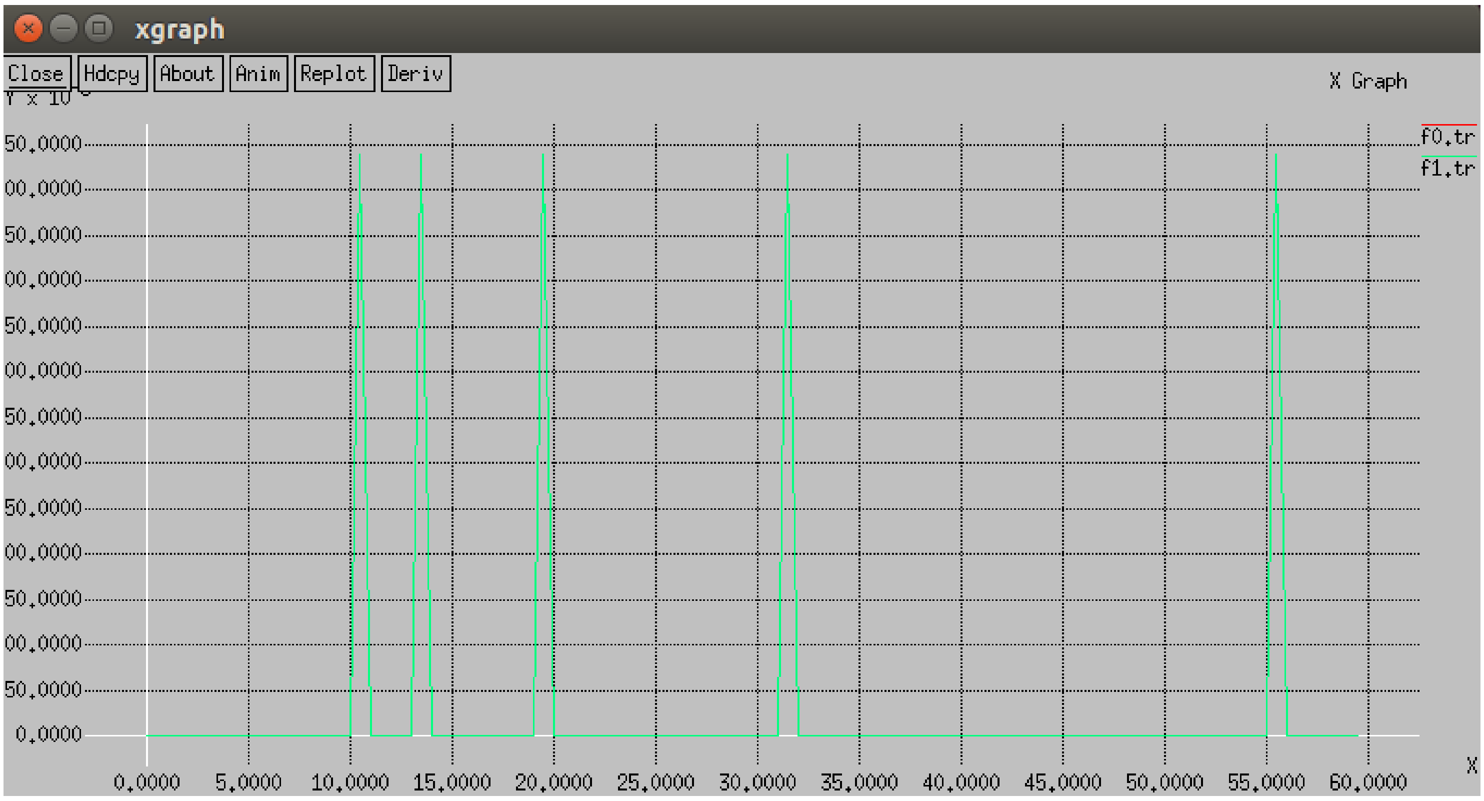
Code Snippet:

```
#Create a simulator object
set ns [new Simulator]


#Open the output files
set f0 [open f0.tr w]
set f1 [open f1.tr w]

#Create 10 nodes
for {set i 0} {$i < 10} {incr i} {
  set n($i) [$ns node]
  $n($i) set X [expr $i * 150]
}

#Connect the nodes
for {set i 0} {$i < 9} {incr i} {
 $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail
 $ns duplex-link-op $n($i) $n([expr $i+1]) orient right
 $ns queue-limit $n($i) $n([expr $i+1]) 50
}

#Define a 'finish' procedure
proc finish {} {
    global f0 f1
    #Close the output files
    close $f0
    close $f1
    #Call xgraph to display the results
    exec xgraph f0.tr f1.tr -geometry 800x400 &
      exit 0
}

proc attach-expoo-traffic { node sink window_size size burst idle rate } {
    #Get an instance of the simulator
    set ns [Simulator instance]
    #Create a UDP agent and attach it to the node
    set source [new Agent/TCP]
    $source set window_ window_size
    $ns attach-agent $node $source

    #Create an Expoo traffic agent and set its configuration parameters
    set traffic [new Application/Traffic/Exponential]
    $traffic set packetSize_ $size
    $traffic set burst_time_ $burst
    $traffic set idle_time_ $idle
    $traffic set rate_ $rate

     # Attach traffic source to the traffic generator
     $traffic attach-agent $source
    #Connect the source and the sink
    $ns connect $source $sink
    return $traffic
}


#Define a procedure which periodically records the bandwidth received by the
#three traffic sinks sink0/1/2 and writes it to the three files f0/1/2.
proc record {} {
     global sink0 sink1 sink2 f0 f1
    #Get an instance of the simulator
    set ns [Simulator instance]
    #Set the time after which the procedure should be called again
     set time 0.5
    #How many bytes have been received by the traffic sinks?
     set bw0 [$sink0 set bytes_]
     set bw1 [$sink1 set bytes_]

    #Get the current time
     set now [$ns now]
    #Calculate the bandwidth (in MBit/s) and write it to the files
     puts $f0 "$now [expr $bw0/$time*8/1000000]"
     puts $f1 "$now [expr $bw1/$time*8/1000000]"

    #Reset the bytes_ values on the traffic sinks
     $sink0 set bytes_ 0
     $sink1 set bytes_ 0

    #Re-schedule the procedure
     $ns at [expr $now+$time] "record"
}

#Create three traffic sinks and attach them to the node n4
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
$ns attach-agent $n(3) $sink0
$ns attach-agent $n(9) $sink1
#Create three traffic sources
set source0 [attach-expoo-traffic $n(0) $sink0 8 1460 2s 1s 100k]
set source1 [attach-expoo-traffic $n(6) $sink1 8 1460 2s 1s 300k]
#Start logging the received bandwidth
$ns at 0.0 "record"
#Start the traffic sources
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
#Stop the traffic sources
```

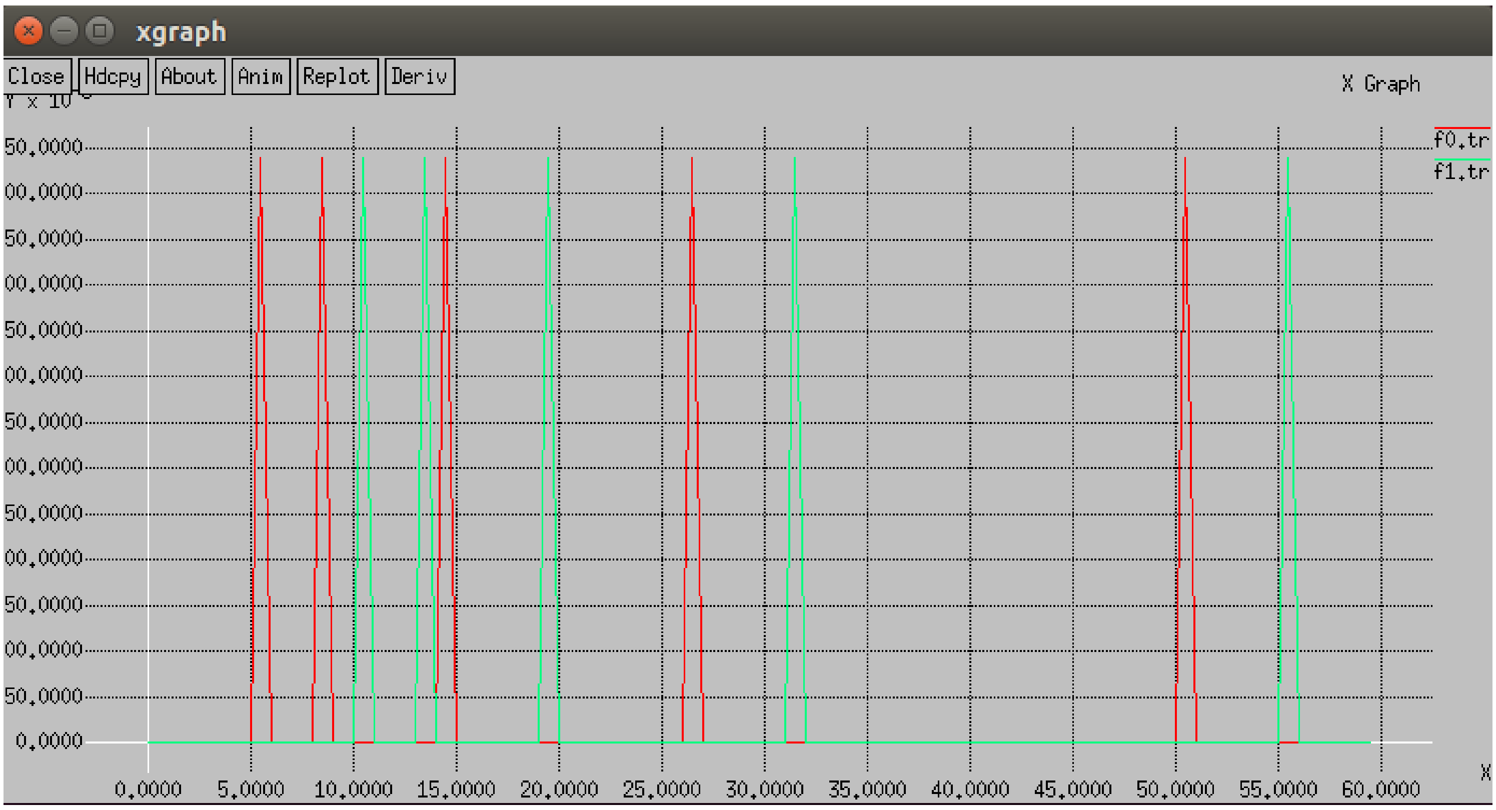
```
$ns at 50.0 "$source0 stop"

$ns at 50.0 "$source1 stop"

#Call the finish procedure after 60 seconds simulation time

$ns at 60.0 "finish"

#Run the simulation

$ns run
```

<u>Sample Run:</u>

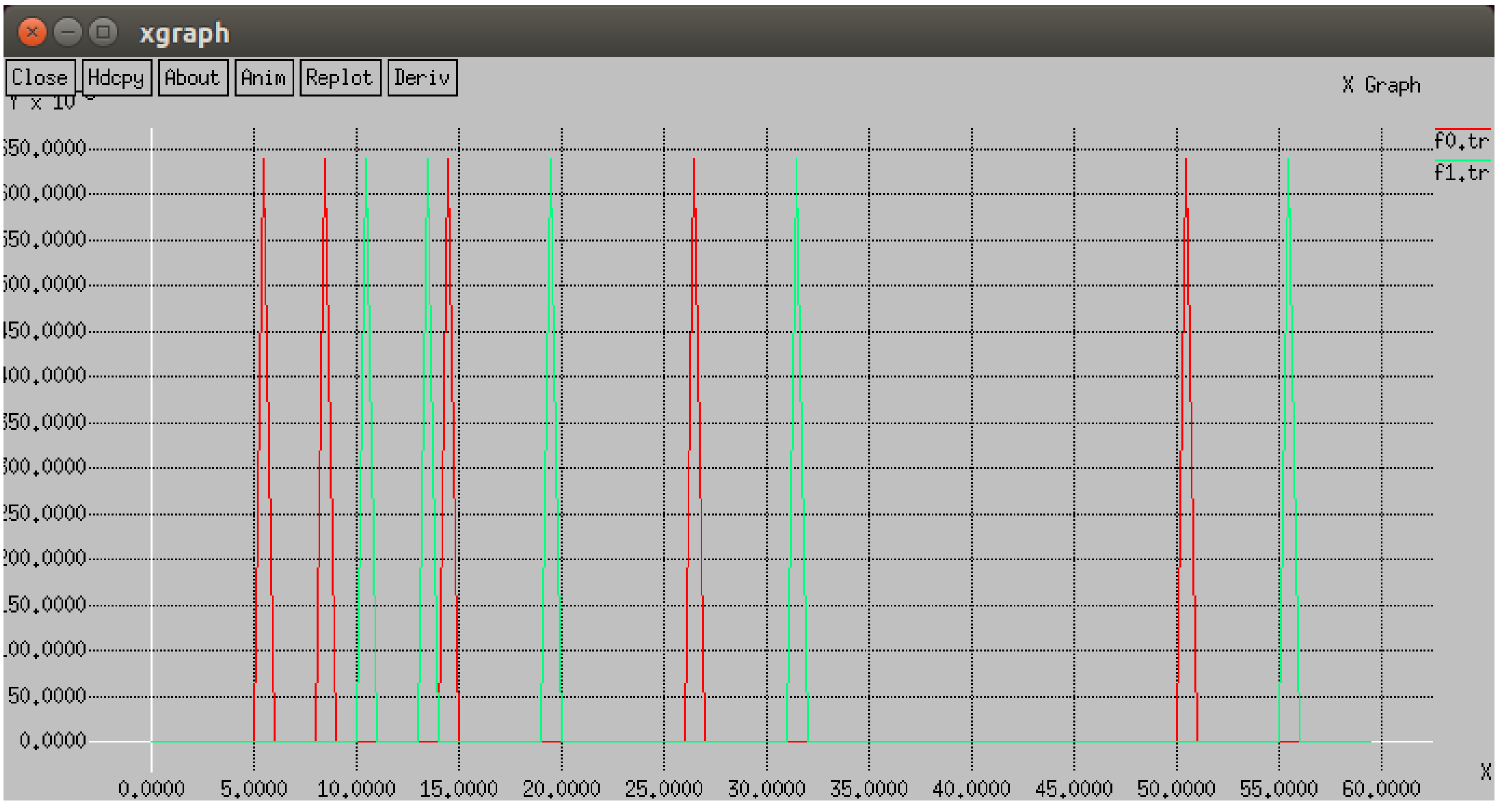
Packet transmission at same time:



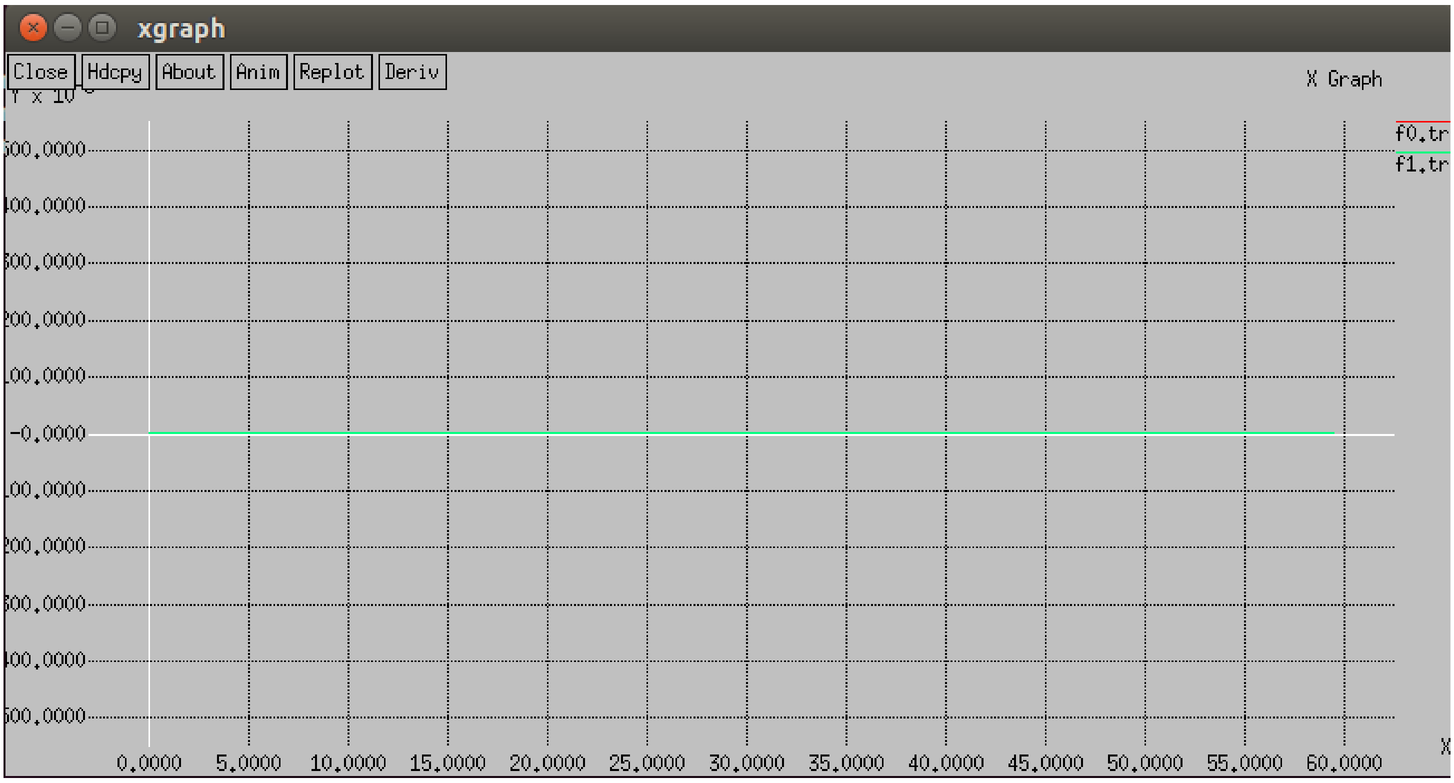
Packet transmission at different time:



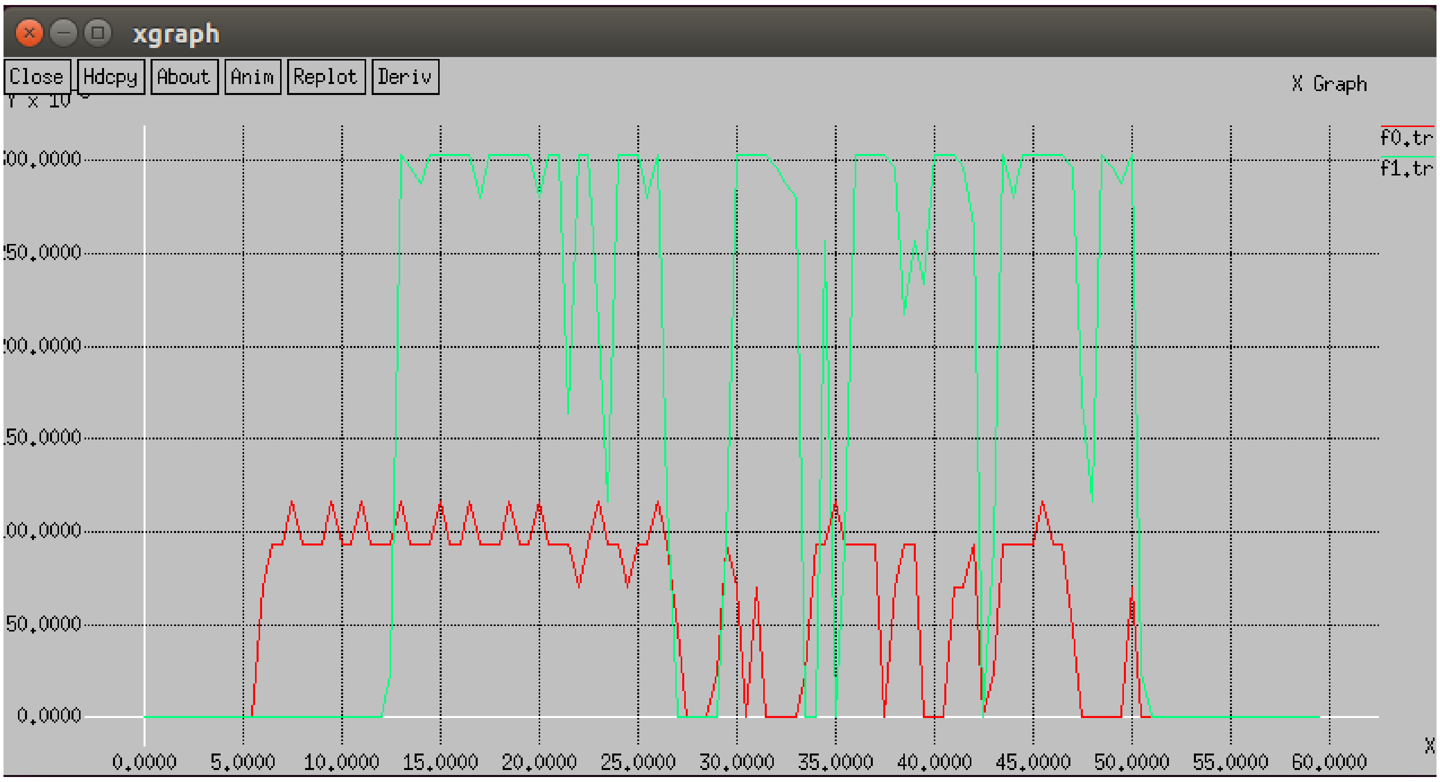
Connection b/w n6 and n4, n2 and n3



Window size = 8



UDP Transmission: *



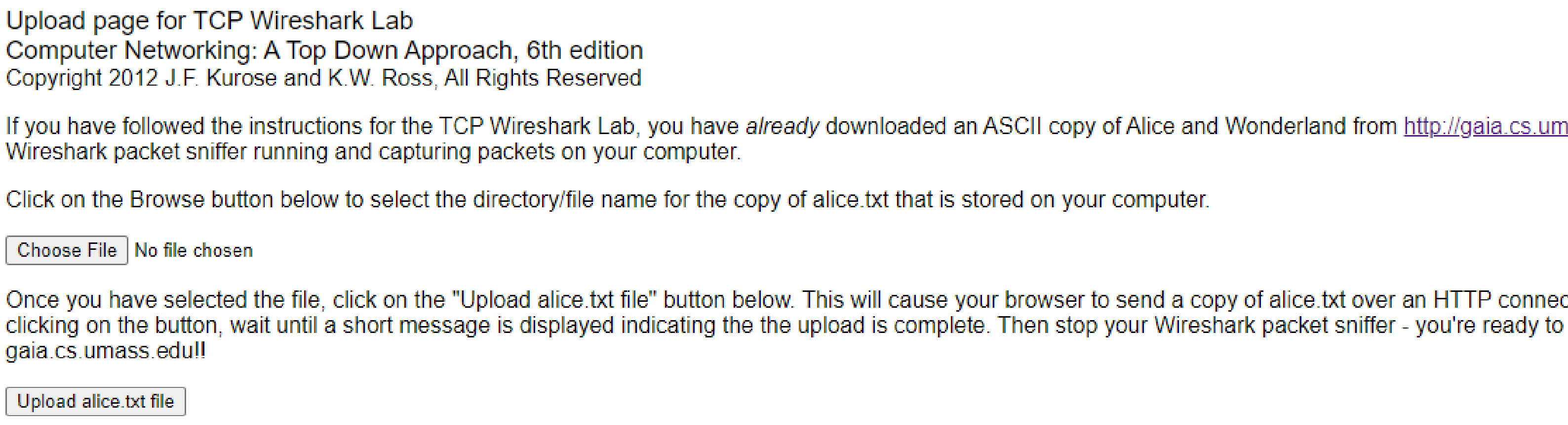
<u>Question no. 02</u> [Assignment: 04]

In this lab, you'll use your Web browser to access a file from a Web server. As in earlier Wireshark labs, you'll use Wireshark to capture the packets arriving at your computer. Unlike earlier labs, you'll also be able to download a Wireshark-readable packet trace from the Web server from which you downloaded the file. In this server trace, you'll find the packets that were generated by your own access of the Web server. You'll analyse the client- and server-side traces to explore aspects of TCP. You'll evaluate the performance of the TCP connection between your computer and the Web server. You'll trace TCP's window behaviour, and infer packet loss, retransmission, flow control and congestion control behaviour, and estimated roundtrip time.

<u>Solution:</u>

<u>*I followed the following steps to create a trace file in Wireshark.*</u>

1. Go to web browser. Go the http://gaia.cs.umass.edu/wireshark-labs/alice.txt and retrieve an ASCII copy of Alice in Wonderland. Store this as a .txt file somewhere on your computer.
2. Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.
3. This is the web page where uploaded the alice.txt file and started Wireshark trace at the same time.



Upload page for TCP Wireshark Lab
Computer Networking: A Top Down Approach, 6th edition
Copyright 2012 J.F. Kurose and K.W. Ross, All Rights Reserved

If you have followed the instructions for the TCP Wireshark Lab, you have *already* downloaded an ASCII copy of Alice and Wonderland from http://gaia.cs.um
Wireshark packet sniffer running and capturing packets on your computer.

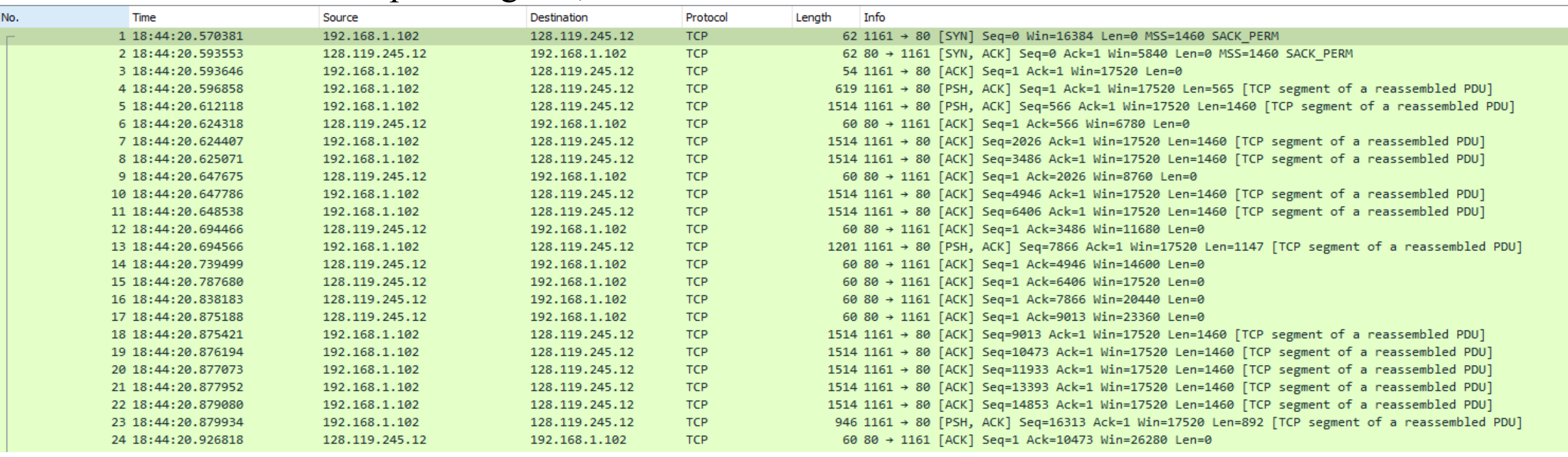
Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

Choose File No file chosen

Once you have selected the file, click on the "Upload alice.txt file" button below. This will cause your browser to send a copy of alice.txt over an HTTP connec
clicking on the button, wait until a short message is displayed indicating the the upload is complete. Then stop your Wireshark packet sniffer - you're ready to
gaia.cs.umass.edu!!

Upload alice.txt file

4. After uploading file, trace file look like this



```
No.  Time             Source           Destination      Protocol  Length  Info
   1 18:44:20.570381  192.168.1.102    128.119.245.12   TCP           62 1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_PERM
   2 18:44:20.593553  128.119.245.12   192.168.1.102    TCP           62 80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
   3 18:44:20.593646  192.168.1.102    128.119.245.12   TCP           54 1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
   4 18:44:20.596858  192.168.1.102    128.119.245.12   TCP          619 1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=565 [TCP segment of a reassembled PDU]
   5 18:44:20.612118  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
   6 18:44:20.624318  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0
   7 18:44:20.624407  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=2026 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
   8 18:44:20.625071  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
   9 18:44:20.647675  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0
  10 18:44:20.647786  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=4946 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  11 18:44:20.648538  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  12 18:44:20.694466  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0
  13 18:44:20.694566  192.168.1.102    128.119.245.12   TCP         1201 1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Win=17520 Len=1147 [TCP segment of a reassembled PDU]
  14 18:44:20.739499  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=4946 Win=14600 Len=0
  15 18:44:20.787680  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=6406 Win=17520 Len=0
  16 18:44:20.838183  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=7866 Win=20440 Len=0
  17 18:44:20.875188  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=9013 Win=23360 Len=0
  18 18:44:20.875421  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=9013 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  19 18:44:20.876194  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=10473 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  20 18:44:20.877073  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=11933 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  21 18:44:20.877952  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=13393 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  22 18:44:20.879080  192.168.1.102    128.119.245.12   TCP         1514 1161 → 80 [ACK] Seq=14853 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
  23 18:44:20.879934  192.168.1.102    128.119.245.12   TCP          946 1161 → 80 [PSH, ACK] Seq=16313 Ack=1 Win=17520 Len=892 [TCP segment of a reassembled PDU]
  24 18:44:20.926818  128.119.245.12   192.168.1.102    TCP           60 80 → 1161 [ACK] Seq=1 Ack=10473 Win=26280 Len=0
```

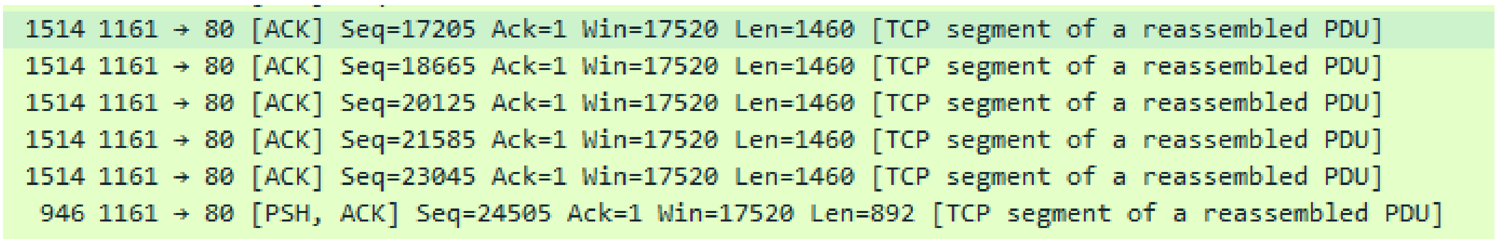
Figure 1

TCP Performance:

As TCP connection follows 3-way handshake, my pc sent a SYN segment to tell the server that I want to establish the connection. Server responded with SYN-ACK segment to allow the communication and then my pc sent the segment again to tell that connection is from legitimate client and alive. As we can see that there is small amount of loss of packets and data reliability is also ensured by addition checksum field in the header.

TCP ensures that all the packets are transferred without any interference.

1. **Throughput:** is almost constant in the given scenario and not decreasing too much due to successful transmission of almost all segments.
2. **Round-Trip Time (RTT):** is not much affected because initial time slice of transmission is 18.44.20.570381 and 18.44.20.596858 which is 0.0.0.26477.
3. **Packet Loss:** is very rare in this scenario of trace file as shown in Figure 1
4. **Retransmission Rate:** As packet loss is very rare that's why retransmission is not required.
5. **TCP window size** changes most of the time in sequence 16384, 5840, 17520, 17520, 17520, 6780 …
6. **Congestion window size** is handled by TCP internally.
7. **Throughput delay product** has not very large factor of increase.
8. **Connection establishment time** is almost similar as initial time to create connection was 18.44.570381
9. **Connection termination time** takes quite a long time than establishment time. This can be due to increasing data in the receiver buffer.
10. **Window scaling** is performed which shows that the receiver is processing data at a higher speed than the sender's sending rate.
11. **Buffer bloat** refers to excessively large buffers causing increased latency.
12. **TCP fast open (TFO)** a mechanism which allows sending data with SYN segment which is not used in the given scenario.
13. **Path MTU** maximum size of an IP packet that can be transmitted without any fragmentation. Fragmentation is also used for fast transmission and due to the very large size of packet.



```
1514 1161 → 80 [ACK] Seq=17205 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
1514 1161 → 80 [ACK] Seq=18665 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
1514 1161 → 80 [ACK] Seq=20125 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
1514 1161 → 80 [ACK] Seq=21585 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
1514 1161 → 80 [ACK] Seq=23045 Ack=1 Win=17520 Len=1460 [TCP segment of a reassembled PDU]
 946 1161 → 80 [PSH, ACK] Seq=24505 Ack=1 Win=17520 Len=892 [TCP segment of a reassembled PDU]
```

14. **Segment Size** is mostly 1514 bytes and often fragmentated.
15. **Flow control** is managed by performing fragmentation and advertising window size.
16. **Congestion control** is also managed but, in this case, receiver is advertising window size of increasing order which means congestion is not occurred yet.

Additional Comments:

1. Ensure that the measurements are taken under representative conditions, considering network characteristics and load.
2. Multiple measurements should be taken to account for variability.
3. Use both active measurements (tools generating traffic) and passive measurements (capturing existing traffic) for a comprehensive analysis.

References:

1. Ns2 tutorial: https://www.isi.edu/nsnam/ns/tutorial/index.html
2. Wireshark Lab helping material: https://gaia.cs.umass.edu/kurose_ross/wireshark.php