CSLR 52
Networks Lab
Assignment - 2

106119029

Dipesh Kafle

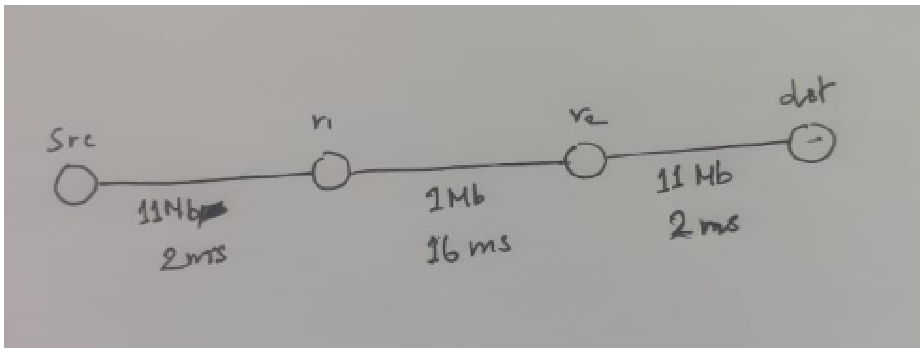
Assignment - 2

1.

Code for Question 1 (Given in question)

```
# ns-tcp-wired.tcl
# Author: Jeff Pang <jeffpang+744@cs.cmu.edu>
#
# 15-744 ps2

# whether to create the nam log for visualization {yes,no}
set namlog      "yes"

set tf [open "out.tr" w]
# when to start the sender
set starttime    1.0
# when to stop the simulation
set stoptime     10.0
# where to log the sender's cwnd trace
set logfilename  "cwnd-tcp-wired.out"

set ns [new Simulator]
$ns trace-all $tf
set cwndf [open $logfilename w]

if {$namlog == "yes"} {
   set namf [open tcp-wired.nam w]
   $ns namtrace-all $namf
}

proc finish {} {
   global ns namlog cwndf namf tf
   $ns flush-trace
   close $cwndf
   close $tf
   if {$namlog == "yes"} {
   close $namf
   }
   exit 0
}

# source node
```

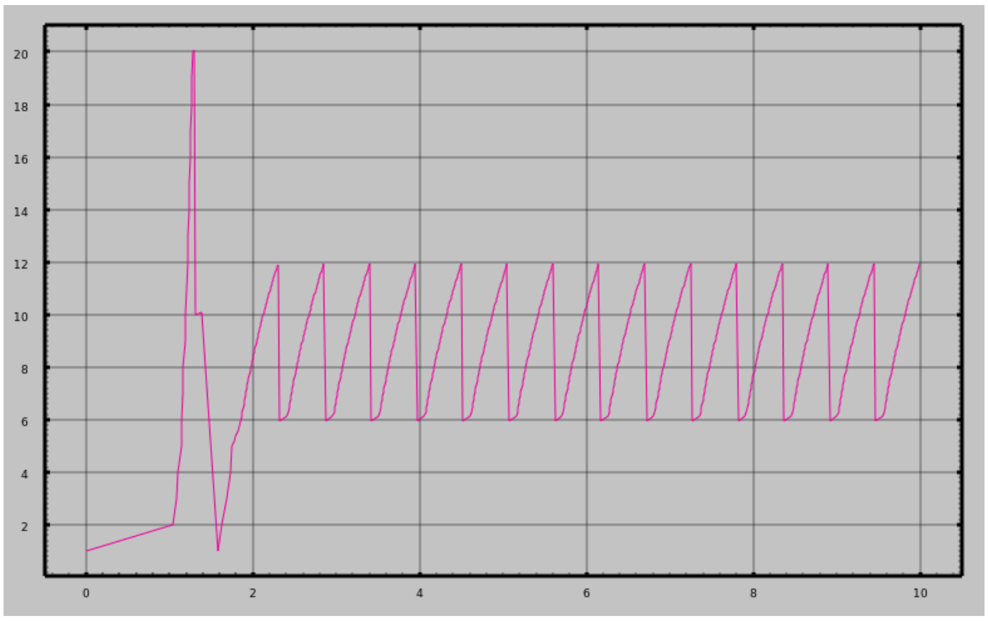
```
set src [$ns node]
# desination node
set dst [$ns node]
# routers
set r1 [$ns node]
set r2 [$ns node]

# bottleneck link
$ns duplex-link $r1 $r2 1Mb 16ms DropTail
# router queue
$ns queue-limit $r1 $r2 5
# access links
$ns duplex-link $src $r1 11Mb 2ms DropTail
$ns duplex-link $dst $r2 11Mb 2ms DropTail

# attach tcp agents
set tcpSender [new Agent/TCP/Reno]
# configure tcp agent
$ns attach-agent $src $tcpSender
set tcpReceiver [new Agent/TCPSink]
$ns attach-agent $dst $tcpReceiver

# have the sender run an ftp app
set ftp [new Application/FTP]
$ftp attach-agent $tcpSender

# trace the sender's cwnd
$tcpSender attach $cwndf
$tcpSender tracevar cwnd_

$ns connect $tcpSender $tcpReceiver

$ns at $starttime "$ftp start"
$ns at $stoptime "$ftp stop"

$ns at $stoptime "finish"

# do the simulation
$ns run
```

a. 

This is a simple network of a source, destination and two routers, with the bandwidth and latency as mentioned.

b. The queueing discipline used is a Droptail queue. A Droptail queue is a basic queue implementation where after reaching the queue capacity any additional packets are simply dropped until there is enough room to accept packets.

c. By simply making a graph from the cwnd trace file generated by the program we get

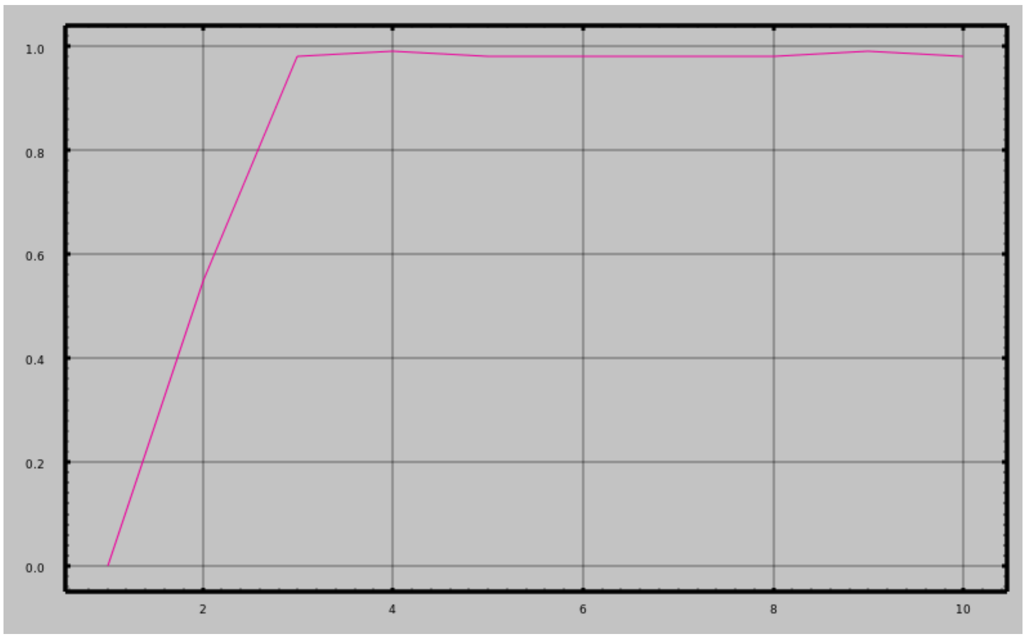


When a new connection is made, cwnd is initialized to one TCP data or acknowledgment packet, and waits for an acknowledgement, or ACK. When that ACK is received, the congestion window is incremented until the cwnd is less than ssthresh. Slow start also terminates when congestion is experienced. And here we can see that the congestion is achieved at the first peak. And hence that is where slow start ends.

d. The throughput is calculated with a simple python script, the core function is as follows:

```
for line in self.get_lines():
           if(line.event == "r" and line.to_node == dest):
               recv_size += line.pkt_size
           if(line.time-last_time >= self.interval):
               last_time += self.interval
               fp.write("{} {}\n".format(
                   last_time, (8*recv_size)/(10**6 * self.interval)))
               avg += (8*recv_size)/(10**6 * self.interval);
               leng += 1
               recv_size = 0
```

Where if in the trace file line that we input,
If a packet is received, we increase the recv_size
If we have exceeded the interval we are accumulating the size, we write the size and go to the next interval.
And the contents of this are printed onto a file so we can see a graph if necessary, and from this file we can also calculate average throughput which was asked in the question.
The average throughput is 0.8420159999999999. And the graph is as follows



2.

CODE

q2.tcl

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

$ns color 1 Blue
$ns color 2 Red

# setting files for trace and nam data
set namtracefile [open q2.nam w]
set tracefile [open q2.tr w]
$ns namtrace-all $namtracefile
$ns trace-all $tracefile

# defining finish procedure to close files
proc finish {} {
   global ns namtracefile tracefile
   $ns flush-trace
   close $namtracefile
   close $tracefile
   exec nam q2.nam &
   exit 0
}

set r1 [$ns node]
set r2 [$ns node]
$ns duplex-link $r1 $r2 2Mb 10ms DropTail

for {set i 0} {$i < 3} {incr i} {

   # create nodes
   set a($i) [$ns node]
   set b($i) [$ns node]

   # make links
   $ns duplex-link $r1 $a($i) 2Mb 10ms DropTail
   $ns duplex-link $r2 $b($i) 2Mb 10ms DropTail

   # create agents
```

```
    set tcp($i) [new Agent/TCP]
    set sink($i) [new Agent/TCPSink]

    # attach agents
    $ns attach-agent $a($i) $tcp($i)
    $ns attach-agent $b($i) $sink($i)
    $ns connect $tcp($i) $sink($i)

    # set packet information
    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp($i)
    $ftp($i) set type_ FTP

    #set color
    $tcp($i) set class_ 2
    $tcp($i) set fid_ 1

    # queue limit for loss
    $ns queue-limit $a($i) $r1 10
    $ns queue-limit $r1 $a($i) 10
    $ns queue-limit $b($i) $r2 10
    $ns queue-limit $r2 $b($i) 10

    # schedule events
    $ns at 0.1 "$ftp($i) start"
    $ns at 9.9 "$ftp($i) stop"
}

# schedule finish
$ns at 10 "finish"

# run
$ns run
```

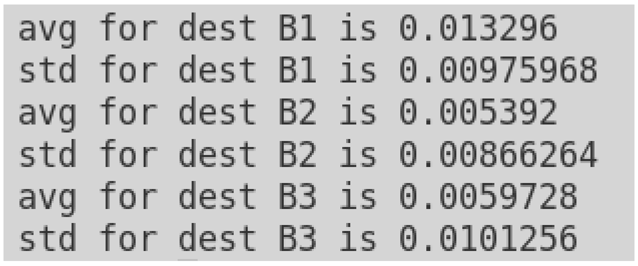
a.

```
   void processPacketData(char event, float timestamp, int toNode,int
seqNo){
        if(seqNo > 100) return;
        if(event == 'r' && toNode == destnode){
               times[toNode].push_back(timestamp);
       }
   }
```

This is the core logic for the program, we just save the arrival times of packets mapped by their sequence number, later those values can be printed onto a file while the average is calculated along the way.

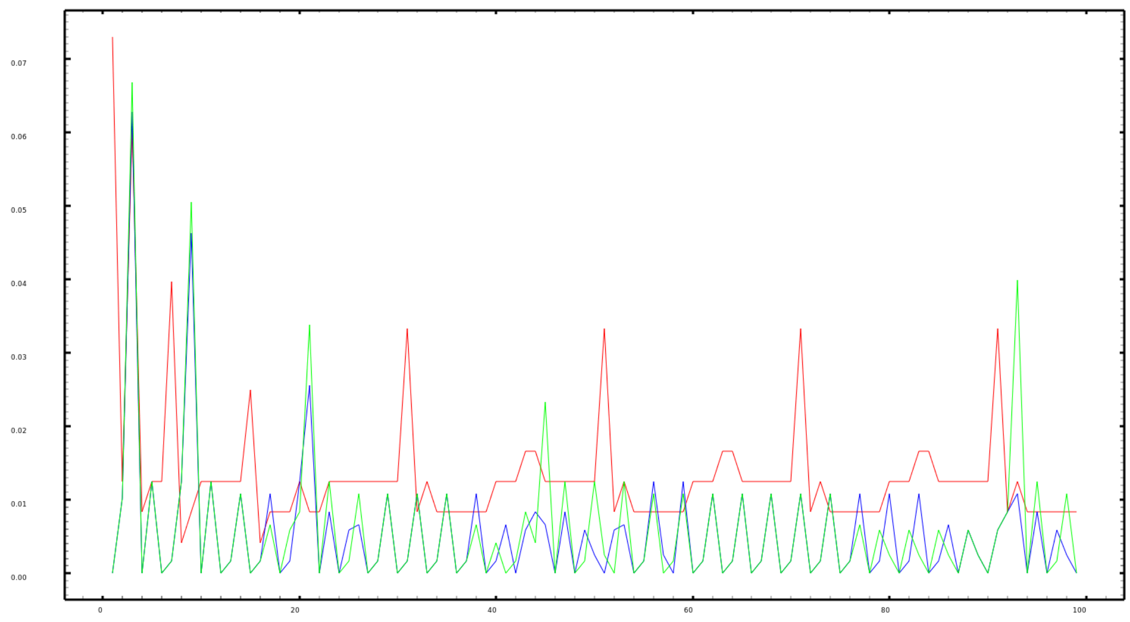
```
for(int i=1;i<100;i++){
           diff = times[3][i]-times[3][i-1];
           avg += diff/100;
           sd += diff*diff/100;
           f<<i<<" "<<diff<<endl;
       }
       cout<<"avg for dest 1 is "<< avg << endl;
       cout<<"std for dest 1 is "<< sqrt(abs(sd - avg*avg))<<endl;
```

The above code shows the core logic of how average, standard deviations are calculated and printed on to a file to graph.
The output is as follows:



```
avg for dest B1 is 0.013296
std for dest B1 is 0.00975968
avg for dest B2 is 0.005392
std for dest B2 is 0.00866264
avg for dest B3 is 0.0059728
std for dest B3 is 0.0101256
```

b. The output file from the above section is used to make the following graph.



c. The middle link limits the bandwidth and hence on reduction of number of nodes on either side, the traffic reduces and hence increases the inter - arrival times. The interarrival times at the beginning are high because tcp sends singular packets at the beginning and then sends more packets at once. If n = 2,1 then the overall traffic decreases and the bandwidth is utilized more effectively leading to lesser interarrival times between packets.