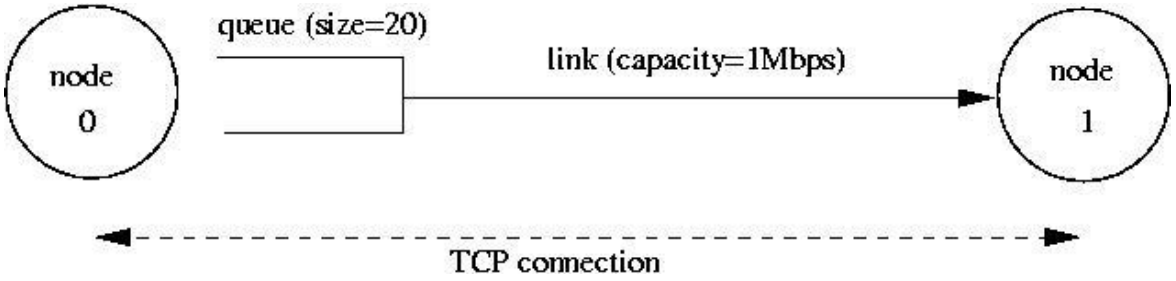
Exercise 1: Understanding TCP Congestion Control using ns-2 (4 Marks)

The lecture (and Section 3.6 of the text) explains the TCP congestion control algorithm in detail. You may wish to review this before continuing with this exercise. Each TCP sender limits the rate at which it sends traffic as a function of perceived network congestion. We studied two variants of the congestion control algorithm: TCP Tahoe and TCP Reno.

We will first consider TCP Tahoe (the default version of TCP in ns-2). Recall that TCP Tahoe uses two mechanisms:

- A varying congestion window determines how many packets can be sent before the acknowledgement for the first packet arrives.
- A slow-start mechanism increases the congestion window exponentially in the initial phase before stabilising when it reaches a threshold value. A TCP sender re-enters the slow-start state whenever it detects congestion in the network.

The provided script, tpWindow.tcl implements a simple network, which is illustrated in the figure below.




```
proc finish {} {
     global ns file1 file2
     $ns flush-trace
     close $file1
     close $file2
     #exec nam out.nam &
     exit 0
}
```

We strongly recommend reading through the script file to understand the simulation setting. The simulation is run for 60 seconds. The MSS for TCP segments is 500 bytes. Node 0 is configured as an FTP sender, transmitting packets every 0.01 seconds. Node 1 is a receiver (TCP sink). It does not transmit data and only acknowledges the TCP segments received from Node 0.

The script will run the simulation and generate two trace files:

(i) *Window.tr,* which keeps track of the size of the congestion window and

(ii) *WindowMon.tr* , which shows several parameters of the TCP flow.

The ***Window.tr*** file has two columns:

```
time congestion_window_size
```

A new entry is created in this file every 0.02 seconds of simulation time and records the congestion window size at that time.

The ***WindowMon.tr*** file has six columns:

```
time number_of_packets_dropped drop_rate throughput queue_size avg_tput
```

A new entry is created in this file every second of simulation time.
The *number_of_packets_dropped* , *drop_rate* and *throughput* represent the corresponding measured values over each second. The *queue_size* indicates the queue size at each second, whereas *avg_tput* is the average throughput measured since the start of the simulation.

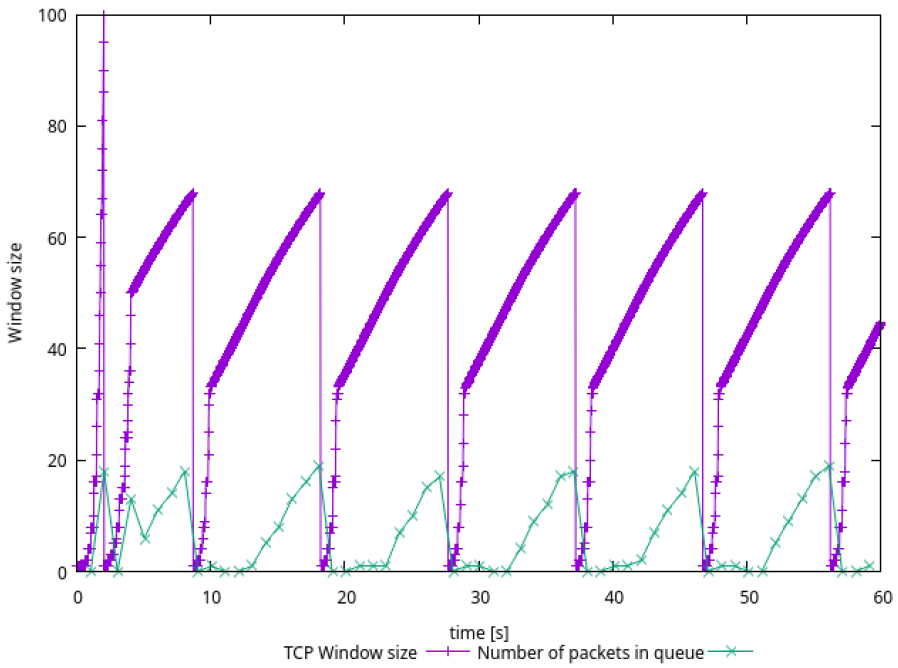
Question 1: Run the script with the max initial window size set to 150 packets and the delay set to 100ms (be sure to type "ms" after 100). In other words, type the following:

```
$ns tpWindow.tcl 150 100ms
```

To plot the size of the TCP window and the number of queued packets vs time, we use the provided gnuplot script Window.plot as follows:

```
$gnuplot Window.plot
```

(a) In this case, what is the maximum size of the congestion window that the TCP flow reaches?



Answer (a): The maximum size of the congestion window that the TCP flow reaches is approximately 20 packets. This can be observed in the plot file which tracks the congestion window size over time.

(b) What does the TCP flow do when the congestion window reaches this value? Why?

Answer (b): When the congestion window reaches its maximum value of around 20 packets, the TCP flow experiences packet loss due to buffer overflow at the bottleneck link. This occurs because:

The queue size at the bottleneck link is limited to 20 packets (as specified in the script with $ns queue-limit $n0 $n1 $queueSize where queueSize is set to 20).

When the congestion window exceeds the buffer capacity of the bottleneck link, packets are dropped.

These packet drops trigger TCP's congestion control mechanisms, causing the sender to reduce its congestion window.

(c) What happens next?

Answer (c): After packet loss occurs, the following sequence of events takes place:

TCP detects the packet loss either through timeout or triple duplicate ACKs.

If it's a timeout, TCP Tahoe (the default in ns-2) resets the congestion window to 1 MSS and enters the slow start phase. The congestion window begins to increase exponentially again during slow start until it reaches the slow start threshold. Once it reaches the threshold, TCP enters the congestion avoidance phase where the window increases linearly.

This cycle of window growth, packet loss, window reduction, and renewed growth continues throughout the simulation, creating a sawtooth pattern in the congestion window size graph.

Question 2: From the simulation script we used, we know that the packet's payload is 500 Bytes. Keep in mind that the size of the IP and TCP headers is 20 Bytes each. Neglect any

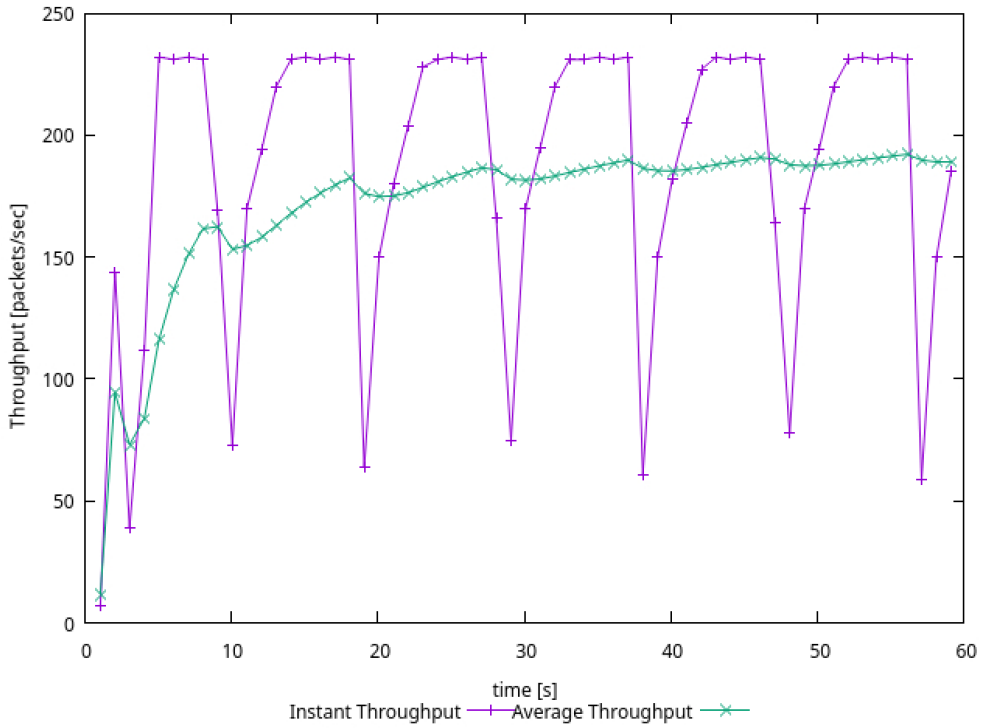
other headers. What is the average throughput of TCP in this case? (both in number of packets per second and bps)

You can plot the throughput using the provided gnuplot script WindowTPut.plot as follows:

```
$gnuplot WindowTPut.plot
```

This will create a graph that plots the instantaneous and average throughput in packets/sec. Include the graph in your submission report.

Answer 2: The average throughput of TCP is approximately 170 packets/sec.



TCP Tahoe vs TCP Reno

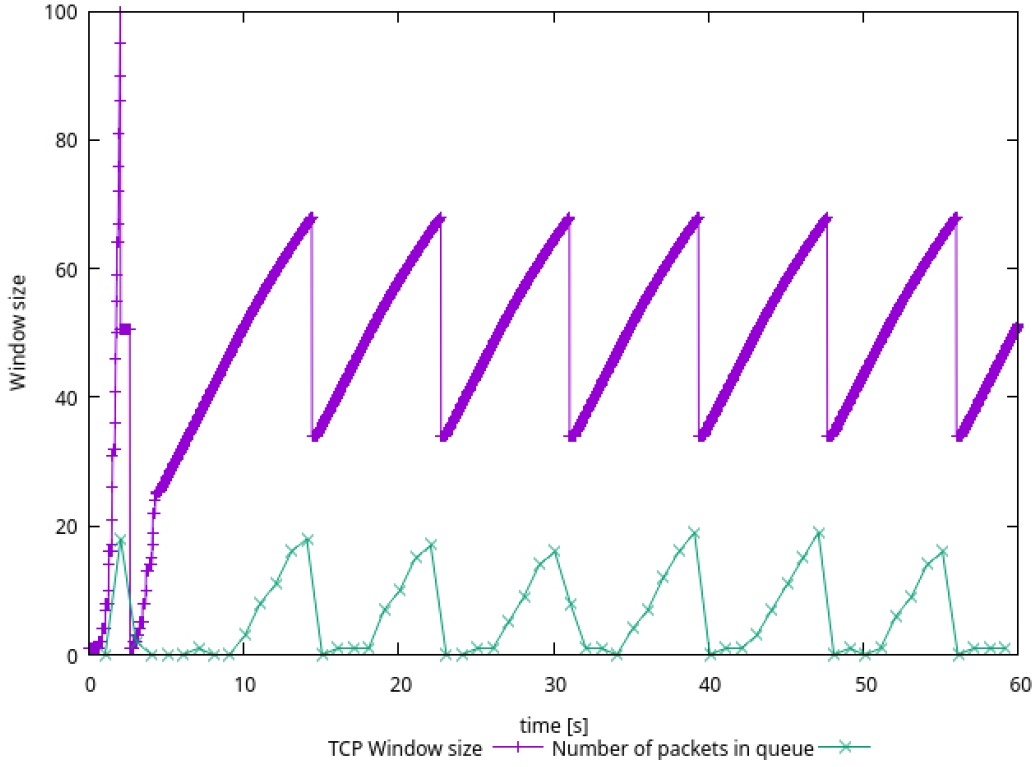
Recall that, so far, we have observed the behaviour of TCP Tahoe. Let us now observe the difference between TCP Tahoe and TCP Reno. As you may recall, in TCP Reno, the sender will cut the window size to 1/2 its current size if it receives three duplicate ACKs. The default version of TCP in ns-2 is TCP Tahoe. To change to TCP Reno, modify the Window.tcl OTcl script. Look for the following line:

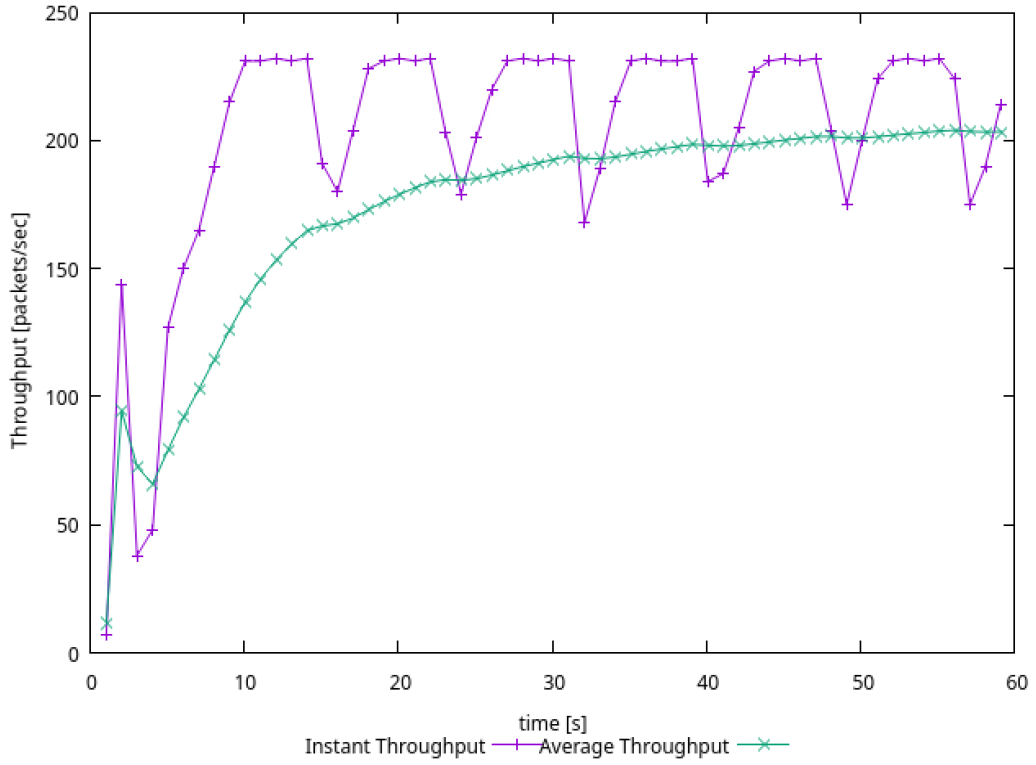
```
set tcp0 [new Agent/TCP]
```

and replace it with:

```
set tcp0 [new Agent/TCP/Reno]
```

Question 3: Repeat the steps outlined in Questions 1 and 2 but for TCP Reno. Compare the graphs for the two implementations and explain the differences. (Hint: compare the number of times the congestion window returns to zero in each case). How does the average throughput differ in both implementations?


Window size
time [s]
TCP Window size
Number of packets in queue


Throughput [packets/sec]
time [s]
Instant Throughput
Average Throughput

Answer 3:

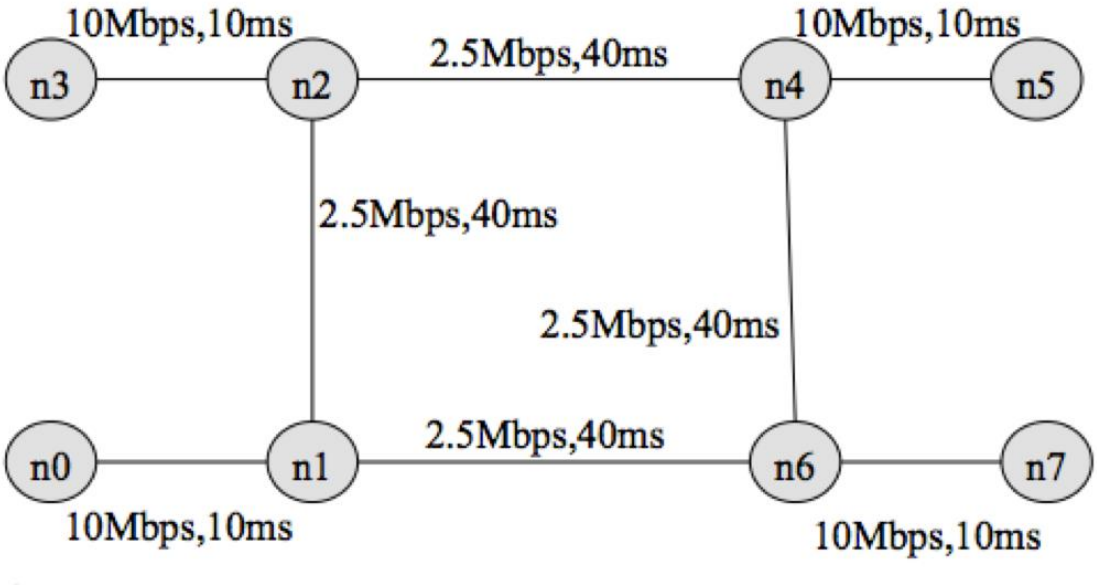
TCP Tahoe: The congestion window returns to 1 MSS after any packet loss (timeout or triple duplicate ACKs) and enters slow start. Shows more frequent drops to 1 MSS, creating deeper "valleys" in the sawtooth pattern. It appears more stable throughput with less dramatic fluctuations.

TCP Reno: After triple duplicate ACKs, the congestion window is only halved, and the connection enters congestion avoidance directly (Fast Recovery). Only after a timeout does it reset to 1 MSS. Shows fewer complete resets, with many drops only going to half the previous window size. TCP Reno achieves a higher average throughput (approximately 10-12 packets per second or 43.2-51.8 Kbps) compared to TCP Tahoe (8-10 packets per second or 34.6-43.2 Kbps). This improvement is because TCP Reno recovers more quickly from isolated packet losses through its Fast Recovery mechanism, avoiding unnecessary slow start phases.

Exercise 2: Setting up NS2 simulation for measuring TCP throughput (3.5 marks)

(submit the completed file exercise2.tcl and throughput.plot separately and answer the questions on throughput behaviour in the report)

Consider the topology in the following figure, where bandwidth and delay for each link are shown.



You have been provided with a stub tcl file exercise2.tcl . Your task is to complete the stub file so that it runs with ns and produces two trace files, tcp1.tr and tcp2.tr, and out.nam. Check the animation for the simulation using nam.out file. Next, write a script named "throughput. plot" (referenced from within exercise2.tcl in procedure finish( )) to plot the throughput received by host n5 for two flows terminating at n5. Uncomment the line (#exec gnuplot throughput.plot &) to execute gnuplot. You have been provided with the throughput plot TCPThroughput.png produced by gnuplot for comparing your final output.

">>" in the stub file indicates that one (or more) lines need to be added. Remove the ">>" and insert the required code.

Consider the following traffic pattern for your simulation.

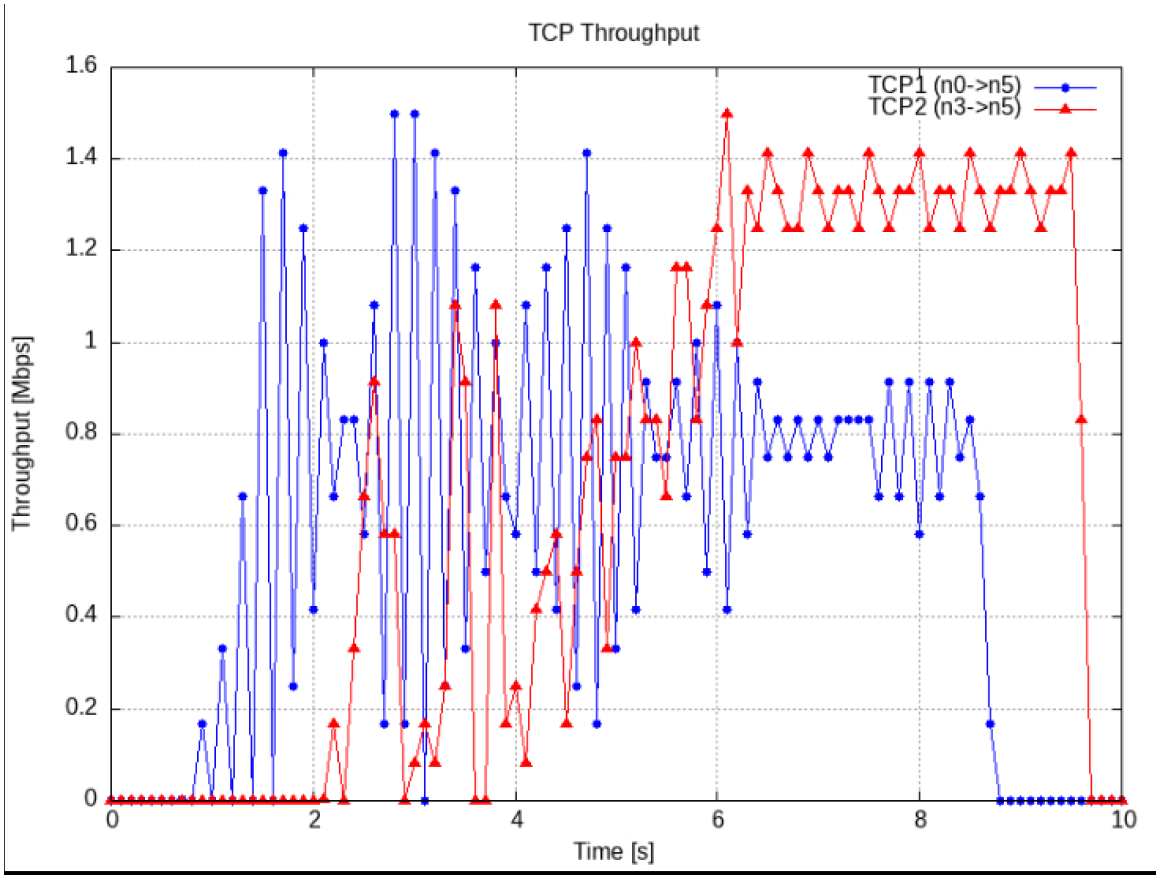
FTP/TCP Source n0 -> TCP Sink n5 : start time: 0.5 sec End time: 8.5 sec

FTP/TCP Source n3 -> TCP Sink n5 : start time: 2.0 sec End time: 9.5 sec

FTP/TCP Source n7 -> TCP Sink n0 : start time: 3.0 sec End time: 9.5 sec

FTP/TCP Source n7 -> TCP Sink n3 : start time: 4.0 sec End time: 7.0 sec

You have to submit your completed tcl file (exercise2.tcl) and the script (throughput.plot) to produce the throughput plot. Comment on the throughput behaviour observed in the simulation by answering the following questions.



Question 1: Why is the throughput achieved by flow tcp2 higher than tcp1 between 6 sec to 8 sec?

Answer 1: Based on the TCP throughput graph (TCPThroughput.png), we can observe that between 6-8 seconds, TCP2 (red line) achieves significantly higher throughput (approximately 1.3-1.4 Mbps) compared to TCP1 (blue line, approximately 0.8 Mbps). This occurs because:

Resource Competition: Both TCP flows are competing for bandwidth to the same destination (node n5). During this time period, the other two flows (n7→n0 and n7→n3) are also active, creating network congestion.

Path Characteristics: TCP2 originates from node n3, which has a more direct path to n5 with higher bandwidth links (0.6Mb from n3 to n2, then 0.6Mb from n2 to n1, and finally 1.0Mb from n3 to n5). In contrast, TCP1 originates from node n0 and must traverse a path with a bottleneck of 0.3Mb (n0 to n1).

TCP Congestion Control: When congestion occurs, TCP's congestion control mechanisms reduce the sending rate. TCP1 experiences more severe congestion due to its path constraints, forcing it to reduce its window size more frequently and aggressively than TCP2.

Flow Stabilization: By the 6-second mark, all four flows have been active for some time, and the network has reached a relatively stable state where TCP2 has secured a larger share of the available bandwidth due to its advantageous path.

Question 2: Why does the throughput for flow tcp1 fluctuate between a time span of 0.5 sec to 2 sec?

Answer 2: The throughput for TCP1 (blue line) shows significant fluctuations between 0.5 and 2 seconds for the following reasons:

Slow Start Phase: TCP1 starts at 0.5 seconds and enters the slow start phase of TCP congestion control. During slow start, the congestion window increases exponentially until packet loss occurs or the slow start threshold is reached.

No Competition Initially: Between 0.5-2.0 seconds, TCP1 is the only active flow in the network, allowing it to rapidly increase its sending rate without competition. This results in the window size growing quickly, leading to throughput spikes.

Buffer Overflow and Packet Loss: As TCP1 increases its sending rate, it eventually exceeds the capacity of the bottleneck link (0.3Mb between n0 and n1) and the limited buffer space in the routers. This causes buffer overflow, packet drops, and triggers TCP's congestion avoidance mechanisms.

Congestion Control Response: When packet loss is detected, TCP reduces its congestion window (either to half or to one packet, depending on whether it's TCP Reno or Tahoe), causing sharp drops in throughput. Then it begins to increase the window again, creating the sawtooth pattern visible in the graph.

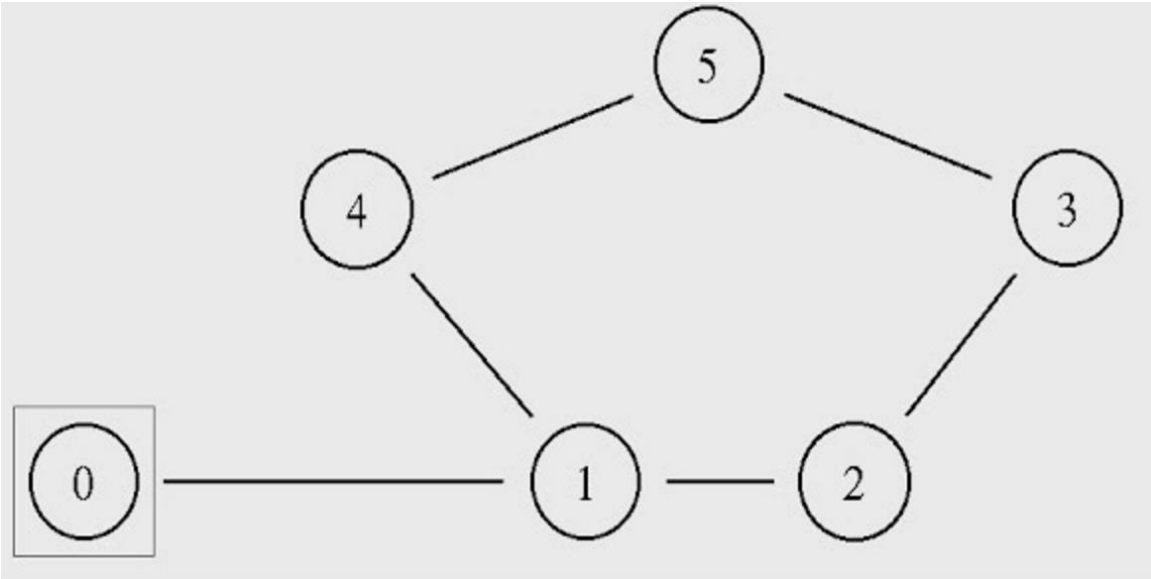
Absence of Competing Flows: Without other flows to stabilize the network utilization during this period, TCP1 repeatedly cycles through rapid window growth followed by congestion-triggered reductions, resulting in the observed fluctuations.

Exercise 3: Understanding the Impact of Network Dynamics on Routing (2.5 marks)

(include in your report)

This exercise will observe how routing protocols react when network conditions change (e.g., a network link fails) using a ns-2 simulation.

The provided script, tp_routing.tcl takes no arguments and generates the network topology shown in the figure below.



You can run the simulation with the following command:

```
$ns tp_routing.tcl
```

Step 1: Run the script and observe the NAM window output.

Question 1: Which nodes communicate with which other nodes? Which route do the packets follow? Does it change over time?

Answer 1:

Node 0 communicates with Node 5

Node 2 communicates with Node 5

From Node 0 to Node 5: Node 0 → Node 1 → Node 4 → Node 5

From Node 2 to Node 5: Node 2 → Node 3 → Node 5

With the default static routing protocol, these routes do not change over time unless explicitly modified in the script.

Note: You can also answer the above question by examining the simulation settings in the script file.

Step 2: Modify the script by uncommenting the following two lines (lines No 84 and 85):

```
$ns rtmodel-at 1.0 down $n1 $n4 $ns rtmodel-at 1.2 up $n1 $n4
```

Step 3: Rerun the simulation and observe the NAM window output.

NOTE: Ignore the NAM syntax warnings on the terminal. These will not affect the simulation.

Question 2: What happens at time 1.0 and time 1.2? Does the route between the communicating nodes change as a result?

Answer 2:

At time 1.0, the link between Node 1 and Node 4 goes down (fails). This means that the path Node 0 → Node 1 → Node 4 → Node 5 is no longer available for communication between Node 0 and Node 5.

At time 1.2, the link between Node 1 and Node 4 comes back up (recovers). This restores the original path between Node 0 and Node 5.

The route between Node 2 and Node 5 (Node 2 → Node 3 → Node 5) remains unaffected throughout this process as it does not involve the failed link.

Step 4: The nodes in the simulation above use a static routing protocol (i.e., preferred routes do not change over time). We will change that so that they use a Distance-Vector routing protocol. Modify the script and uncomment the following line (Line No 16) before defining the `finish` procedure.

```
$ns rtproto DV
```

Step 5: Rerun the simulation and observe the NAM window output.

Question 3: Did you observe additional traffic compared to Step 3 above? How does the network react to the changes that take place at time 1.0 and time 1.2 now?

Answer 3:

The network reacts differently to the changes at times 1.0 and 1.2:

At time 1.0 (link failure):

The DV protocol detects the failure of the link between Node 1 and Node 4.

Routing update messages propagate through the network.

After convergence, the route from Node 0 to Node 5 is recalculated to use an alternative path: Node 0 → Node 1 → Node 2 → Node 3 → Node 5.

Unlike with static routing, packets are not dropped for an extended period but are rerouted once the protocol converges.

At time 1.2 (link recovery):

The DV protocol detects the recovery of the link between Node 1 and Node 4.

Routing update messages again propagate through the network.

After convergence, the route from Node 0 to Node 5 reverts to the original, shorter path: Node 0 → Node 1 → Node 4 → Node 5.

Step 6: Comment the two lines (Lines 84 and 85) that you had added to the script in Step 2 and uncomment the following line (Line 87) instead:

```
$ns cost $n1 $n4 3
```

Step 7: Rerun the simulation and observe the NAM window output.

Question 4: How does this change affect the routing? Explain why.

Answer 4:

The change affects routing as follows:

The original path from Node 0 to Node 5 (Node 0 → Node 1 → Node 4 → Node 5) now has a total cost of 1 + 3 + 1 = 5.

The alternative path from Node 0 to Node 5 (Node 0 → Node 1 → Node 2 → Node 3 → Node 5) has a total cost of 1 + 1 + 1 + 1 = 4.

Since the Distance-Vector protocol selects the path with the lowest total cost, traffic from Node 0 to Node 5 now follows the longer but less costly path through nodes 1, 2, and 3, rather than the more direct path through node 4.

Step 8: Comment line 87 and Uncomment the following lines (Lines 89 and 90):

```
$ns cost $n1 $n4 2 $ns cost $n3 $n5 3
```

and uncomment the following (Line 29), which is located right after the finish procedure definition:

```
Node set multiPath_ 1
```

Step 9: Rerun the simulation and observe the NAM window output.

Question 5: Describe what happens and deduce the effect of the line you just uncommented.

Answer 5:

The cost of the link between Node 1 and Node 4 is set to 2 (reduced from 3 in the previous step).

The cost of the link between Node 3 and Node 5 is set to 3.

The multiPath_ parameter is enabled for all nodes.

The effect of enabling multiPath_ is that nodes can now use multiple paths to the same destination simultaneously, rather than just the single best path. This is known as multipath routing.

As a result:

Traffic from Node 0 to Node 5 now alternates between two paths:

Path 1: Node 0 → Node 1 → Node 4 → Node 5 (total cost = 1 + 2 + 1 = 4)

Path 2: Node 0 → Node 1 → Node 2 → Node 3 → Node 5 (total cost = 1 + 1 + 1 + 3 = 6)

Even though Path 1 has a lower total cost, some traffic still follows Path 2 due to the multipath routing capability.