COMPUTER NETWORKS (CS F303)
LAB-SHEET – 8
TOPIC: Queuing and Routing Protocol Simulation using ns-2

Learning Objectives:

1. **To understand different queuing mechanism (i.e., DropTail and RED)**
2. **To simulate link failure in a larger topology.**
3. **To understand the Routing protocol used in ns-2.**

Prerequisites: Lab-6 and Lab-7

Quick Recap:-

Lab 6 introduced the concept of Network Simulation using ns2 in which we learned to use UDP and TCP agents, analyzing the trace file and using nam for visualizing the simulation. In Lab-7 our focus was specifically on TCP and to observe the variation in throughput and congestion w.r.t time under different scenarios. In this lab session we will start with different queuing techniques, then we will simulate link failure and finally we will see different routing protocols used in ns-2 and simulate one of them.

Module 1: Queue Management and Packet Scheduling:

Queues represent locations where packets may be held (or dropped). Packet scheduling refers to the decision process used to choose which packets should be serviced or dropped. Buffer management refers to any particular discipline used to regulate the occupancy of a particular queue. NS2 supports different techniques like drop-tail (FIFO) queuing, RED buffer management, variants of Fair Queuing and Deficit Round-Robin (DRR) etc.
Till now we have used only DropTail queuing which we specified during defining the link:

```
$ns duplex-link $n0 $n1 3Mb 10ms DropTail
```

In ns-2 a queue object is a general class of object capable of holding and possibly marking or discarding packets as they travel through the simulated topology. It has various parameters one of the parameters that we will use is limit_ i.e. the queue size in packets it can be specified as:

```
$ns queue-limit $n0 $n1 20
```

Objects derived from the base class Queue are DropTail, FQ, SFQ, DRR, RED etc. which we can use in place of DropTail. Today, we will compare DropTail with RED rest of the mechanism can be explored as an exercise.

DropTail queue object: It implements simple FIFO queue. There are no configuration parameter that are specific to drop-tail objects.

RED object: It implement random early-detection gateways. It has a number of parameters that you can set a few are discussed below:

- q_weight_: weight factor wq used to compute the average queue size

- thresh_: The minimum threshold for the average queue size in packets (queue threshold at which RED begins to mark packets probabilistically).
- maxthresh_: The maximum threshold for the average queue size in packets (queue threshold at which RED begins to mark every arriving packets).

These can be set by writing:

Queue/RED set thresh_ 15
Queue/RED set maxthresh_ 18

Monitoring Queues:

Queue statistics can be recorded by queue monitors in a queue monitor file.

set qmon [$ns monitor-queue $n1 $n2 [open qtrace.tr w] 0.03]

[$ns link $n1 $n2] queue-sample-timeout

Here, 0.03 is the recording interval and **qtrace.tr** is output file-name.
The format of the output monitor file is as:

Time	From Node	To Node	Current Queue Size in Bytes	Current Queue Size in Packets	#arr P	#dep P	#drop	#arr B	#dep B	#drop

From and To Node is the link that recorded the event.
#arr P – No. of arrival (packets) in monitor-interval

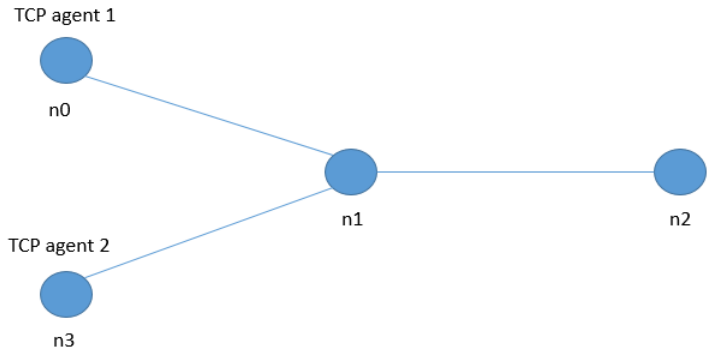
#dep P - No. of departures (packets) in monitor-interval

#drop - No. of drop (packets) in monitor-interval.

Last three column are same parameter in bytes.

Exercise 1: (Using DropTail and RED queue for plotting congestion window and queue size.)

We will use the same topology that we have used in **Exercise-2 of Lab 7**



Node n0 and n3 are the source of TCP traffic, and n2 is the receiver/sink. Links are full-duplex, n0-n1 and n3-n1 are 2Mbps with 10ms delay whereas n1-n2 is 1 Mbps with 10ms delay. Set the queue-limit of n1-n2 link as 20. Add the queue monitor for n1-n2 link. (Note: You don't need to specify any RED parameter for this task, they will get configured automatically).
In your simulation, start the TCP traffic at 1 second and run it till 60 seconds and end the simulation.
You can add the following line in the code to monitor queue in nam:

$ns duplex-link-op $n1 $n2 queuePos 0.5

Tasks:

1. Run the simulation twice, first with DropTail and then with RED queue on n1-n2 link.
2. Using Xgraph, plot the cwnd and throughput of the TCP flows w.r.t. time.
3. In every queue monitor object i.e qmon in our case, current queue size is available in the variable pkts_. Plot the queue size variation w.r.t time.

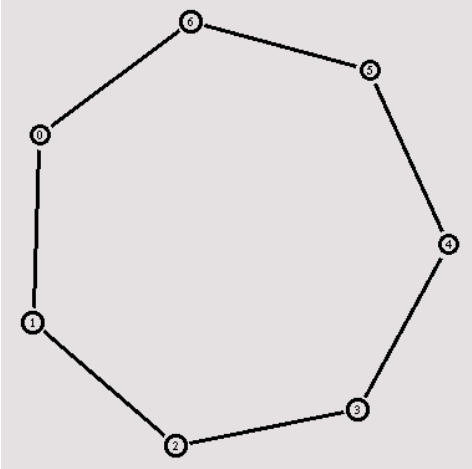
Try to answer following questions:

1. What is TCP synchronization problem? In which case it is more prominent.
2. Which queuing mechanism (Drop Tail or RED) is better for bursty traffic?
3. Observe the Queue size variation in both the cases. Compare both the methods w.r.t queuing delay.
4. How does changing the RED parameters affect the plots?
5. Add one more TCP flow on any of the node (n0/n3). Change queue-size to 30 on n1-n2 link and 3Mbps bandwidth instead of 2 Mbps on n0-n1 & n3-n1. Analyze whether the pattern continues.

Module 2: Link-Failure

Exercise 2: (Simulating dynamic link failure in a network topology)

In this case we will use a topology of seven nodes which can be extended. For building the topology we will use loop construct. Topology is shown in the figure below:



For building the topology required script is as follows:

```
#Create seven nodes
for {set i 0} {$i < 7} {incr i} {
    set n($i) [$ns node]
}
#Create links between the nodes
for {set i 0} {$i < 7} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
```

Note: The name of the node variable is n(0) which actually is an array instead of n0. So rename the variable in your code if you are using the latter. You can try increasing and decreasing the number of nodes by changing the loop limit i.e. 7.

Now add a CBR-UDP traffic sending data from n(0) to n(3). Run the simulation for 5s.

For Link-Failure we will need to add the following lines to the code:

```
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
```

In this case the link between $n(1) & $n(2) will go down after 1s and will recover at 2s. So it will simulate the link-failure for a duration of 1s.

Tasks:

1. Build the topology given above. Note: Click on re-layout button in nam to get the above structure.
2. First run the simulation and observe the flow of traffic without any link-failure.
3. Add link-failure in the topology as specified above.

Try to answer following questions:

1. Even after the link-failure does node $n(0) continues to send packets constantly. Why?
2. Try the same simulation with TCP traffic and answer Q1.
3. Why node $n(0) is not using the alternate path after the link-failure?

Module 3: Routing in ns-2:

The user level simulation script requires one command to specify the unicast routing strategy or protocols for the simulation. A routing strategy is a general mechanism by which ns-2 will compute routes for the simulation. There are four routing strategies specified in ns-2: Static, Session, Dynamic and Manual.
rtproto{} is the instance that specifies the unicast routing protocol to be used in the simulation. It takes multiple arguments, the first of which is mandatory; this first argument identifies the routing protocol to be used. Subsequent arguments specify the nodes that will run the instance of this protocol. The default is to run the same routing protocol on all the nodes in the topology.
Example:

```
$ns rtproto Static              ;# Enable static route strategy for the simulation
$ns rtproto Session             ;# Enable session routing for this simulation
$ns rtproto DV $n1 $n2 $n3      ; #Run DV agents on nodes $n1, $n2, and $n3
```

Link cost: It specifies the cost of the link to carry the traffic. The link cost can be calculated using dynamic parameters or static parameters. For example, if we specify the cost of a link such that its cost becomes inversely proportional to its total capacity (i.e. BW) then cost of low BW links becomes more than the cost of high BW links. Now routes calculated based on minimum cost prefers higher BW links over lower BW links. It is possible to change the link costs at each of the links.

```
$ns cost $n1 $n2 10
$ns cost $n2 $n1 5
```

Assign link cost based on the following formula: (Used in OSPF protocol)

C = 10^8/BW of link in bps (assume link BW is not more than 100 Mbps)

Notice that the procedure sets the cost along one direction only.

In this lab we will focus on Static and DV routing protocol.

Static Routing: The static route computation strategy is the default route computation mechanism in ns-2. This strategy uses the Dijkstra's all-pairs SPF algorithm. The route computation algorithm runs exactly once prior to the start of the simulation. And this is the reason why the alternate path was not used by the node $n(0) in the previous exercise.

DV Routing: DV routing is the implementation of Distributed Bellman-Ford (or Distance Vector) routing in ns-2. The implementation sends periodic route updates at every **advertInterval**.
We can specify this as:

```
Agent/rtProto/DV set advertInterval [interval in sec]
```

In addition to periodic updates, each agent also sends triggered updates; it does this whenever the forwarding tables in the node change.
Each agent employs the split horizon with poisoned reverse mechanisms to advertise its routes to adjacent peers. "Split horizon" is the mechanism by which an agent will not advertise the route to a destination out of the interface that it is using to reach that destination.

Now to get the routing table we will need a procedure to dump the routing information. You can use the following:

```
proc rtdump {} {
        global ns
        set now [$ns now]
        puts "Routing table at time $now"
        # Use any one of the following
        $ns dump-routelogic-nh                  #Table in terms of next hops
        $ns dump-routelogic-distance            #Table in terms of distance
}
```

Call this procedure at specified time to get the instantaneous routing table.
i.e. $ns at [time] rtdump

Exercise 3: (Using dynamic routing protocols in ns-2)

Tasks:

1. Run the simulation of Exercise 2 with DV routing protocol for all nodes and observe flow of traffic. Don't specify any advertInterval it will take the default value.
2. Get the routing table information before link is down.
3. Get the routing table information when the link is down.
4. Open the trace files and find the packets sent by routing protocol.
5. Now set the **advertInterval** to 4s and observe the flow of traffic.
6. After performing above tasks set the link cost to avoid the link-failure.

Try to answer following questions:

1. We specified the link-failure for a duration of 1s and **adverInterval** to 4s in task 5. How does node $n(0) detected link-failure without the periodic updates?
2. Can you guess the default **advertInterval** approximately using the trace files? (Hint: Run simulation without link-failure).
3. Why to use routing protocols if we can just avoid link-failures as you did in Task-6.
4. Is it possible to simulate the Count-to-Infinity problem using DV protocol in the given topology?

******