**Laboratory 2**

**Advanced Emulab tutorial on DETER**

Esha Desai

USC ID: 6993245898

**Contents**

• Dynamic Scheduling of Events………………………………………………………………………………………..3

• Supported Events……………………………………………………………………………………………………………6

• Program Objects…………………………………………………………………………………………………………….7

• Link tracing and Monitoring……………………………………………………………………………………………9

• EndNode Tracing and Monitoring…………………………………………………………………………………11

• Creating Event Groups………………………………………………………………………………………………….11

• Conclusion……………………………………………………………………………………………………………………14

**Advanced Emulab tutorial on DETER**

In the advanced tutorial various new terms were made familiar to me. I learnt what “router queues” are. *Every router interface has a queue* to hold packets awaiting transmission. I learnt about the two types of queues other than the usual Droptail queue which are called: RED (random early detection) and GRED (Gentle random early detection).The difference between the two queues is that RED uses a steep dropping (packets) function to maintain an average queue size while the GRED uses a smooth dropping function to maintain an average queue size. I learnt about a new terminology called CBR (Constant bit rate) generator which generates packet traffic at a constant rate. TCP and UDP agent/s are used for that. I learnt about how the packets on links and lans can be observed from the Link tracing and monitoring section.

**1.Dynamic scheduling of events:**

Dynamic scheduling of events allows us to schedule events on the fly. The *tevc* command allows dynamic injection of events.

NS Script:

*source tb\_compat.tcl*

*set ns [new Simulator]*

*# Create four nodes*

*set nodeA [$ns node]*

*set nodeB [$ns node]*

*# Create a RED duplex link*

*set link0 [$ns duplex-link $nodeA $nodeB 100Mb 0ms RED]*

*# Get the queue object for the nodeA/nodeb link and modify its RED params.*

*set queue0 [[$ns link $nodeA $nodeB] queue]*

*$queue0 set gentle\_ 1*

*$queue0 set queue-in-bytes\_ 0*

*$queue0 set limit\_ 50*

*$queue0 set maxthresh\_ 20*

*$queue0 set thresh\_ 7*

*$queue0 set linterm\_ 11*

*$queue0 set q\_weight\_ 0.004*

*# Create a UDP agent and attach it to nodeA*

*set udp0 [new Agent/UDP]*

*$ns attach-agent $nodeA $udp0*

*# Create a CBR traffic source and attach it to udp0*

*set cbr0 [new Application/Traffic/CBR]*

*$cbr0 set packetSize\_ 500*

*$cbr0 set interval\_ 0.005*

*$cbr0 attach-agent $udp0*

*# Create a TCP agent and attach it to nodeA*

*set tcp0 [new Agent/TCP]*

*$ns attach-agent $nodeA $tcp0*

*# Create a CBR traffic source and attach it to tcp0*

*set cbr1 [new Application/Traffic/CBR]*

*$cbr1 set packetSize\_ 500*

*$cbr1 set interval\_ 0.005*

*$cbr1 attach-agent $tcp0*

*# Create a Null agent (a UDP traffic sink) and attach it to node nodeB*

*set null0 [new Agent/Null]*

*$ns attach-agent $nodeB $null0*

*# Create a TCPSink agent (a TCP traffic sink) and attach it to node nodeB*

*set null1 [new Agent/TCPSink]*

*$ns attach-agent $nodeB $null1*

*# Connect the traffic sources with the traffic sinks*

*$ns connect $udp0 $null0*

*$ns connect $tcp0 $null1*

*# And some events.*

*$ns at 60.0 "$cbr0 start"*

*$ns at 70.0 "$link0 bandwidth 10Mb duplex"*

*$ns at 80.0 "$link0 delay 10ms"*

*$ns at 90.0 "$link0 plr 0.05"*

*$ns at 100.0 "$link0 down"*

*$ns at 110.0 "$link0 up"*

*$ns at 115.0 "$cbr0 stop"*

*$ns at 120.0 "$cbr1 start"*

*$ns at 130.0 "$cbr1 set packetSize\_ 512"*

*$ns at 130.0 "$cbr1 set interval\_ 0.01"*

*$ns at 140.0 "$link0 down"*

*#Run the simulation*

*$ns run*

After swapping in the above ns file, I got the following details as the listing in the ”Show events” tab under the Details of the experiment:

*Experiment: USC558L/tevc*

*Event List:*

*Time Node Agent Type Event Parent Arguments*

*------------ ------------ ---------- ---------- ---------- ---------- ------------*

*0.000 nodeA cbr0 TRAFGEN MODIFY \_\_ns\_sequence PACKETSIZE=500*

*RATE=100000*

*INTERVAL=0.005*

*IPTOS=-1*

*0.000 nodeA cbr1 TRAFGEN MODIFY \_\_ns\_sequence PACKETSIZE=500*

*RATE=100000*

*INTERVAL=0.005*

*IPTOS=-1*

*60.000 nodeA cbr0 TRAFGEN START*

*70.000 link0 LINK MODIFY BANDWIDTH=10000*

*80.000 link0 LINK MODIFY DELAY=10ms*

*90.000 link0 LINK MODIFY PLR=0.05*

*100.000 link0 LINK DOWN*

*110.000 link0 LINK UP*

*115.000 nodeA cbr0 TRAFGEN STOP*

*120.000 nodeA cbr1 TRAFGEN START*

*130.000 nodeA cbr1 TRAFGEN MODIFY PACKETSIZE=512*

*130.000 nodeA cbr1 TRAFGEN MODIFY INTERVAL=0.01*

*140.000 link0 LINK DOWN*

*Event Summary:*

*--------------*

*Event count: 13*

*First event: 0.000 seconds*

*Last event: 140.000 seconds*

The above details only display the events in the ns script but the dynamic events can be seen using the “*tail –f*” command to watch the change in the event scheduler log file on the fly.

I opened two terminals and from one I used the tevc command to stop cbr0 at real time by using the command *tevc -e USC558L/tevc now cbr0 stop.*

And from the second terminal ,I invoked the command:

*At Node A:*

*sc558ag@nodea:/proj/USC558L/exp/tevc/logs$ tail -f event-sched.log*

After a few seconds after the invoking of tevc command, the last two lines show that the real time fire of the log information of stopping of cbr0. The few seconds take place because it takes time for the script to go through the boss and event scheduler. The following is the snapshot of the output:

After stopping cbr0, I again “dynamically” started cbr0 and then stopped it.

**2. Supported Events:**

We can also modify the link parameters by using the tevc commands(also shown in snapshot) :

*tevc –e USC/558L/dynsched now link) modify link0 bandwidth=20000*

*tevc –e USC/558L/dynsched now link) modify link0 delay=5ms*

*tevc –e USC/558L/dynsched now link) modify link0 up*

We can see that the link0 gets modified thrice when the above 3 tevc commands are invoked.

At node A:

Next we can see that the queue parameters can also be modified using the tevc command by modifying the link which is related to that queue. Below can be seen the modification of queue parameters related to link0. Following are the commands for modifying various queue parameters like limit\_ and q\_weight\_ and link parameters like bandwidth:

*tevc –e USC558L/dynsched now link0 modify q\_weight\_=0.003*

*tevc –e USC558L/dynsched now link0 modify bandwidth=20000*

*tevc –e USC558L/dynsched now link0 modify limit\_=75*

At node A:

**3.Program Objects:**

NS Script:

*# This is a simple ns script. Comments start with #.*

*set ns [new Simulator]*

*source tb\_compat.tcl*

*set nodeA [$ns node]*

*set nodeB [$ns node]*

*set nodeC [$ns node]*

*set nodeD [$ns node]*

*set prog0 [$nodeA program-agent -command "echo HELLO &> tmp/foo"]*

*ser prog1 [$nodeA program-agent –command “/bin/ls –lt”]*

*set link0 [$ns duplex-link $nodeB $nodeA 30Mb 50ms DropTail]*

*tb-set-link-loss $link0 0.01*

*set lan0 [$ns make-lan "$nodeD $nodeC $nodeB " 100Mb 0ms]*

*# Set the OS on a couple.*

*tb-set-node-os $nodeA FBSD-STD*

*tb-set-node-os $nodeC RHL-STD*

*$ns rtproto Static*

*# Go!*

*$ns run*

The above ns scriptallows us to make a program object that prints “HELLO” and puts the output in the file foo under tmp directory.We can see that before the invoking of the tevc command there was no foo file created under the tmp directory but after the tevc command is invoked, the foo file is created and we can see that HELLO is printed in the foo file. Below is the snapshot of the result.

*At node A:*

Also below is the snap shot of tevc command used to start program objects prog0, start prog1 and stop prog1 on the left terminal. While the right terminal shows the real time change in the start / stop of the program objects prog0 and prog1.

**4.Link Tracing and Monitoring:**

NS Script:

*# This is a simple ns script. Comments start with #.*

*set ns [new Simulator]*

*source tb\_compat.tcl*

*set nodeA [$ns node]*

*set nodeB [$ns node]*

*set nodeC [$ns node]*

*set nodeD [$ns node]*

*set link0 [$ns duplex-link $nodeB $nodeA 30Mb 50ms DropTail]*

*$link0 trace*

*$link0 trace packet*

*$link0 trace monitor "icmp or tcp"*

*tb-set-link-loss $link0 0.01*

*set lan0 [$ns make-lan "$nodeD $nodeC $nodeB " 100Mb 0ms]*

*# Set the OS on a couple.*

*tb-set-node-os $nodeA FBSD-STD*

*tb-set-node-os $nodeC RHL-STD*

*$ns rtproto Static*

*$ns run*

The ns script ‘$link0 trace’ captures just the packet headers. In case we want to capture all the data in the packets then ‘$link0 trace packet’ is helpful. If we want to filter our packet capturing to just icmp or tcp packets then ‘$link0 “icmp or tcp”’ script is used.

We can monitor the packet capturing real time at node A and node B by using interactive web-interface. We can pause packet capturing, restart it or kill it using the web interface. Below is the output from interactive 'Link Tracing/Monitoring':

At node A:

At node B:

The first thing on all the lines above denotes the timestamp. The number of the brackets show the number of interfaces to that node. Icmp:0,0 indicates that there were 0 bytes of icmp traffic sent in 0 packets.

Other than the interactive web-interface, we can locally monitor/capture the packets too at the delay node.

The tcpdump in the /local/logs file at the delaynode tb0 after executing the following commands looks like:

*[sc558ag@users ~]$ ssh pc030* (pc030 being the unqualified name of the delay node)

*-bash-2.05b$ cd /local/logs*

*-bash-2.05b$ ls -a*

*. trace\_nodeA-link0.recv trace\_nodeB-link0.recv* (4 CAPTURE FILES ARE CREATED)

*.. trace\_nodeA-link0.xmit trace\_nodeB-link0.xmit*

*-bash-2.05b$ sudo cat trace\_nodeA-link0.recv*

(The .recv files hold the packets that were sent by the node[A/B] and *received* by the delay node tb0. The .xmit files hold those packets that were *transmitted* by the delay node tb0 and received by the node[A/B] on the other side of the link.)

**5.EndNode Tracing/Monitoring:**

I also did end node tracing and monitoring by inserting *$link0 trace\_endnode* in the ns script, where instead of 4 capture files only one file was created that is only *trace\_nodeA-link0.xmit* file was created. In this one the end nodes are used to capture packets instead of the delay node.

**6.Creating Event Groups:**

NS Script:

*source tb\_compat.tcl*

*set ns [new Simulator]*

*# Create two nodes*

*set nodeA [$ns node]*

*set nodeB [$ns node]*

*set nodeC [$ns node]*

*# Create a RED duplex link*

*set link0 [$ns duplex-link $nodeA $nodeB 100Mb 0ms RED]*

*set link1 [$ns duplex-link $nodeB $nodeC 200Mb 0ms RED]*

*# cREATE AN EVENT GROUP*

*set mylinks [new EventGroup $ns]*

*$mylinks add $link0 $link1*

*$ns at 140.0 "$mylinks down"*

*# Get the queue object for the nodeA/nodeb link and modify its RED params.*

*set queue0 [[$ns link $nodeA $nodeB] queue]*

*$queue0 set gentle\_ 1*

*$queue0 set queue-in-bytes\_ 0*

*$queue0 set limit\_ 50*

*$queue0 set maxthresh\_ 20*

*$queue0 set thresh\_ 7*

*$queue0 set linterm\_ 11*

*$queue0 set q\_weight\_ 0.00*

*# Create a UDP agent and attach it to nodeA*

*set udp0 [new Agent/UDP]*

*$ns attach-agent $nodeA $udp0*

*# Create a CBR traffic source and attach it to udp0*

*set cbr0 [new Application/Traffic/CBR]*

*$cbr0 set packetSize\_ 500*

*$cbr0 set interval\_ 0.005*

*$cbr0 attach-agent $udp0*

*# Create a TCP agent and attach it to nodeA*

*set tcp0 [new Agent/TCP]*

*$ns attach-agent $nodeA $tcp0*

*# Create a CBR traffic source and attach it to tcp0*

*set cbr1 [new Application/Traffic/CBR]*

*$cbr1 set packetSize\_ 500*

*$cbr1 set interval\_ 0.005*

*$cbr1 attach-agent $tcp0*

*# Create a UDP agent and attach it to nodeC*

*set udp1 [new Agent/UDP]*

*$ns attach-agent $nodeC $udp1*

*# Create a CBR traffic source and attach it to udp1*

*set cbr2 [new Application/Traffic/CBR]*

*$cbr2 set packetSize\_ 500*

*$cbr2 set interval\_ 0.005*

*$cbr2 attach-agent $udp1*

*# Create a TCP agent and attach it to nodeC*

*set tcp1 [new Agent/TCP]*

*$ns attach-agent $nodeC $tcp1*

*# Create a CBR traffic source and attach it to tcp1*

*set cbr3 [new Application/Traffic/CBR]*

*$cbr3 set packetSize\_ 500*

*$cbr3 set interval\_ 0.005*

*$cbr3 attach-agent $tcp1*

*# Create a Null agent (a UDP traffic sink) and attach it to node nodeB*

*set null0 [new Agent/Null]*

*$ns attach-agent $nodeB $null0*

*# Create a TCPSink agent (a TCP traffic sink) and attach it to node nodeB*

*set null1 [new Agent/TCPSink]*

*$ns attach-agent $nodeB $null1*

*# Connect the traffic sources with the traffic sinks*

*$ns connect $udp0 $null0*

*$ns connect $tcp0 $null1*

*# And some events.*

*$ns at 60.0 "$cbr0 start"*

*$ns at 70.0 "$link0 bandwidth 10Mb duplex"*

*$ns at 80.0 "$link0 delay 10ms"*

*$ns at 90.0 "$link0 plr 0.05"*

*$ns at 100.0 "$link0 down"*

*$ns at 110.0 "$link0 up"*

*$ns at 115.0 "$cbr0 stop"*

*$ns at 120.0 "$cbr1 start"*

*$ns at 130.0 "$cbr1 set packetSize\_ 512"*

*$ns at 130.0 "$cbr1 set interval\_ 0.01"*

*#$ns at 140.0 "$link0 down"*

*$ns at 150.0 "$cbr1 stop"*

*# And some events.*

*$ns at 60.0 "$cbr2 start"*

*$ns at 70.0 "$link1 bandwidth 10Mb duplex"*

*$ns at 80.0 "$link1 delay 10ms"*

*$ns at 90.0 "$link1 plr 0.05"*

*$ns at 100.0 "$link1 down"*

*$ns at 110.0 "$link1 up"*

*$ns at 115.0 "$cbr2 stop"*

*$ns at 120.0 "$cbr3 start"*

*$ns at 130.0 "$cbr3 set packetSize\_ 512"*

*$ns at 130.0 "$cbr3 set interval\_ 0.01"*

*#$ns at 140.0 "$link1 down"*

*$ns at 150.0 "$cbr3 stop"*

*#Run the simulation*

*$ns run*

But on running this script ,I got the following error.

On analyzing this, I tried instead for the dynamic event scheduling method instead of including it in the ns script. Following snapshot shows that at nodeA the group MyLinks which includes both link0 and link1 goes down on invoking the *tevc –e USC558L/evntgrps now mylinks down.*

**Conclusion:**

The advanced tutorial aimed at getting familiar with the ns scripting in detail. I learnt how the monitoring of the packet traffic could be done. I learnt about two types of Queues – RED and GRED and various queue parameters and link parameters. I learnt how the various parameters could be changed on the file using the “tevc” command on the fly. We can monitor link or queue parameters if they are modified on the fly in the log files. We can create groups of links or groups of same objects which have same events at the same time to avoid unnecessary repetition of scripts.