Laboratory 2

Advanced Emulab tutorial on DETER

Esha Desai

USC ID: 6993245898

Contents

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
```

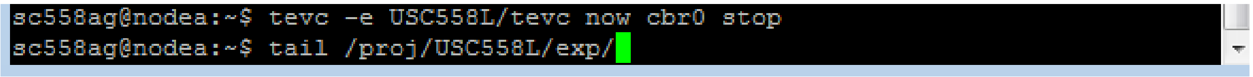
```
                                                                          RATE=100000
                                                                          INTERVAL=0.005
                                                                          IPTOS=-1
60.000       nodeA        cbr0       TRAFGEN    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
```

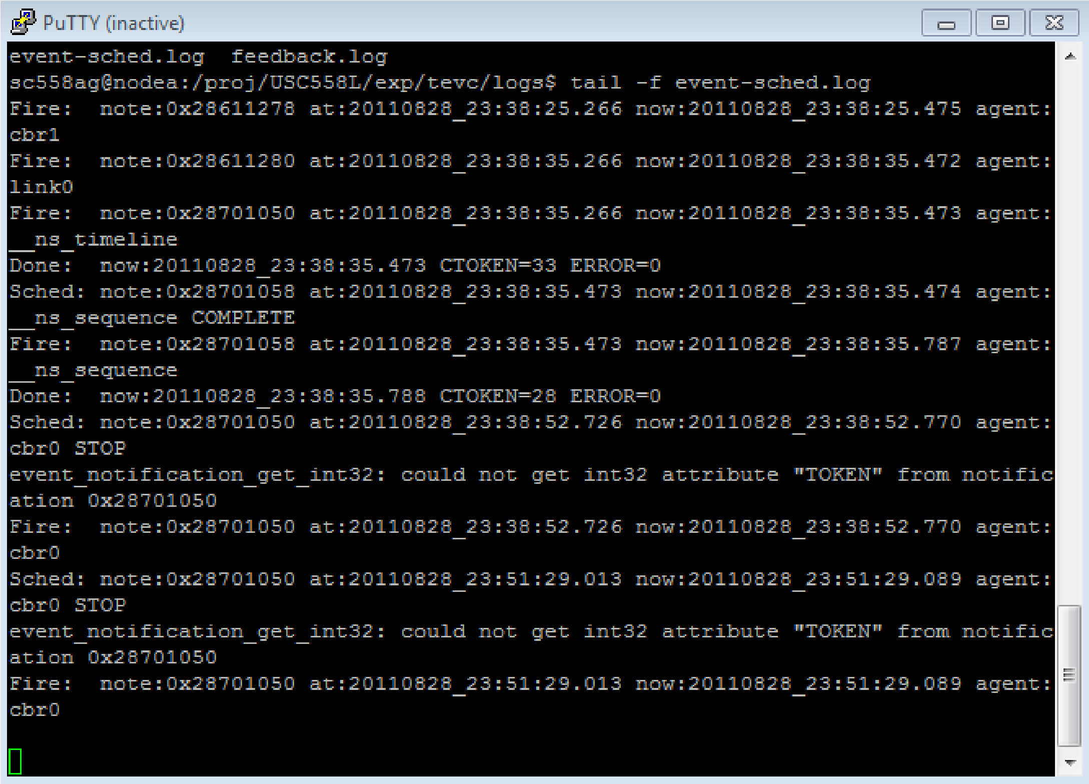
The above details only display the events in the ns script but the dynamic events can be seen using the “*tail –f*” command to watch the change in the event scheduler log file on the fly.

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:

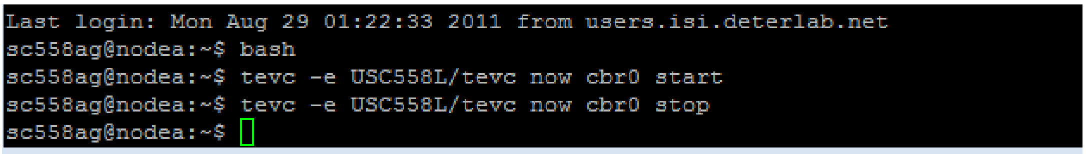


```
sc558ag@nodea:~$ tevc -e USC558L/tevc now cbr0 stop
sc558ag@nodea:~$ tail /proj/USC558L/exp/
```

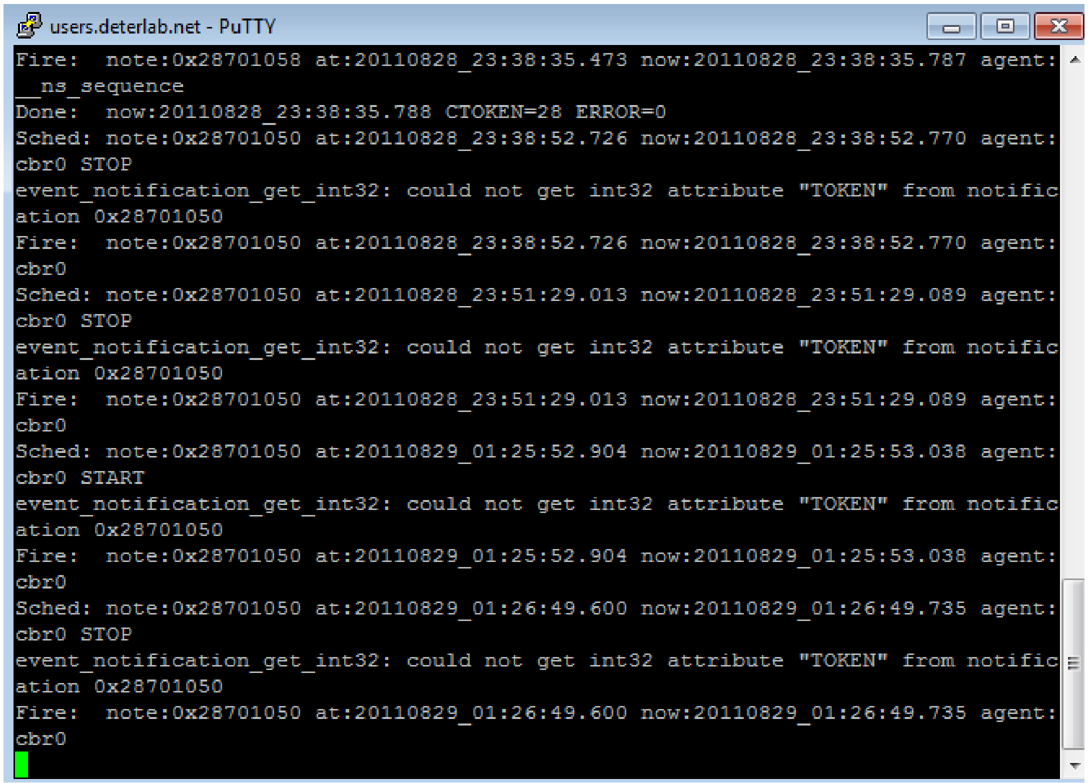


```
event-sched.log  feedback.log
sc558ag@nodea:/proj/USC558L/exp/tevc/logs$ tail -f event-sched.log
Fire:  note:0x28611278 at:20110828_23:38:25.266 now:20110828_23:38:25.475 agent:
cbr1
Fire:  note:0x28611280 at:20110828_23:38:35.266 now:20110828_23:38:35.472 agent:
link0
Fire:  note:0x28701050 at:20110828_23:38:35.266 now:20110828_23:38:35.473 agent:
__ns_timeline
Done:  now:20110828_23:38:35.473 CTOKEN=33 ERROR=0
Sched: note:0x28701058 at:20110828_23:38:35.473 now:20110828_23:38:35.474 agent:
__ns_sequence COMPLETE
Fire:  note:0x28701058 at:20110828_23:38:35.473 now:20110828_23:38:35.787 agent:
__ns_sequence
Done:  now:20110828_23:38:35.788 CTOKEN=28 ERROR=0
Sched: note:0x28701050 at:20110828_23:38:52.726 now:20110828_23:38:52.770 agent:
cbr0 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110828_23:38:52.726 now:20110828_23:38:52.770 agent:
cbr0
Sched: note:0x28701050 at:20110828_23:51:29.013 now:20110828_23:51:29.089 agent:
cbr0 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110828_23:51:29.013 now:20110828_23:51:29.089 agent:
cbr0
```

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



```
Last login: Mon Aug 29 01:22:33 2011 from users.isi.deterlab.net
sc558ag@nodea:~$ bash
sc558ag@nodea:~$ tevc -e USC558L/tevc now cbr0 start
sc558ag@nodea:~$ tevc -e USC558L/tevc now cbr0 stop
sc558ag@nodea:~$
```

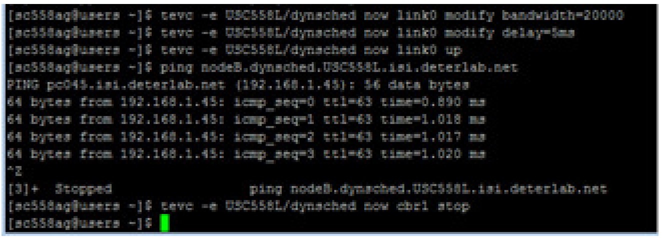


```
Fire:  note:0x28701058 at:20110828_23:38:35.473 now:20110828_23:38:35.787 agent:
__ns_sequence
Done:  now:20110828_23:38:35.788 CTOKEN=28 ERROR=0
Sched: note:0x28701050 at:20110828_23:38:52.726 now:20110828_23:38:52.770 agent:
cbr0 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110828_23:38:52.726 now:20110828_23:38:52.770 agent:
cbr0
Sched: note:0x28701050 at:20110828_23:51:29.013 now:20110828_23:51:29.089 agent:
cbr0 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110828_23:51:29.013 now:20110828_23:51:29.089 agent:
cbr0
Sched: note:0x28701050 at:20110829_01:25:52.904 now:20110829_01:25:53.038 agent:
cbr0 START
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_01:25:52.904 now:20110829_01:25:53.038 agent:
cbr0
Sched: note:0x28701050 at:20110829_01:26:49.600 now:20110829_01:26:49.735 agent:
cbr0 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_01:26:49.600 now:20110829_01:26:49.735 agent:
cbr0
```

2. Supported Events:

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

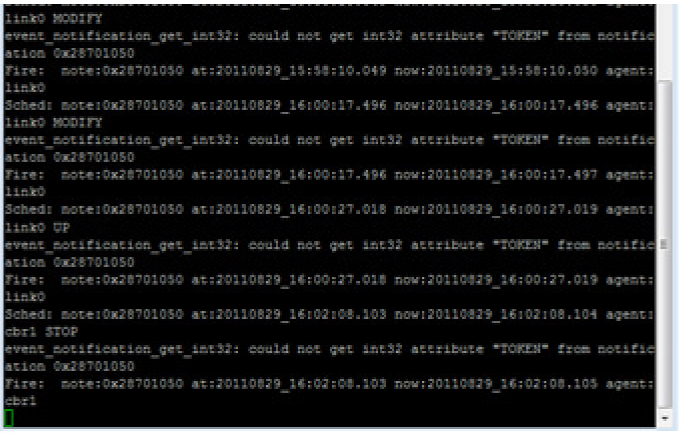
tevc –e USC/558L/dynsched now link) modify link0 bandwidth=20000
tevc –e USC/558L/dynsched now link) modify link0 delay=5ms
tevc –e USC/558L/dynsched now link) modify link0 up



```
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 modify bandwidth=20000
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 modify delay=5ms
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 up
[sc558ag@users ~]$ ping nodeB.dynsched.USC558L.isi.deterlab.net
PING pc045.isi.deterlab.net (192.168.1.45): 56 data bytes
64 bytes from 192.168.1.45: icmp_seq=0 ttl=63 time=0.890 ms
64 bytes from 192.168.1.45: icmp_seq=1 ttl=63 time=1.018 ms
64 bytes from 192.168.1.45: icmp_seq=2 ttl=63 time=1.017 ms
64 bytes from 192.168.1.45: icmp_seq=3 ttl=63 time=1.020 ms
^Z
[3]+  Stopped                 ping nodeB.dynsched.USC558L.isi.deterlab.net
[sc558ag@users ~]$ tevc -e USC558L/dynsched now cbr1 stop
[sc558ag@users ~]$
```

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

At node A:

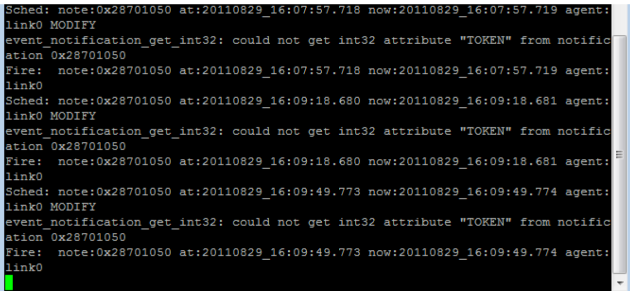


```
link0 MODIFY
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_15:58:10.049 now:20110829_15:58:10.050 agent:
link0
Sched: note:0x28701050 at:20110829_16:00:17.496 now:20110829_16:00:17.496 agent:
link0 MODIFY
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:00:17.496 now:20110829_16:00:17.497 agent:
link0
Sched: note:0x28701050 at:20110829_16:00:27.018 now:20110829_16:00:27.019 agent:
link0 UP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:00:27.018 now:20110829_16:00:27.019 agent:
link0
Sched: note:0x28701050 at:20110829_16:02:08.103 now:20110829_16:02:08.104 agent:
cbr1 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:02:08.103 now:20110829_16:02:08.105 agent:
cbr1
```

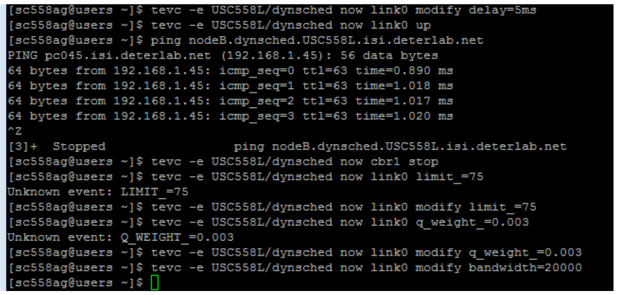
Next we can see that the queue parameters can also be modified using the tevc command by modifying the link which is related to that queue. Below can be seen the modification of queue parameters related to link0. Following are the commands for modifying various queue parameters like limit_ and q_weight_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



```
Sched: note:0x28701050 at:20110829_16:07:57.718 now:20110829_16:07:57.719 agent:
link0 MODIFY
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:07:57.718 now:20110829_16:07:57.719 agent:
link0
Sched: note:0x28701050 at:20110829_16:09:18.680 now:20110829_16:09:18.681 agent:
link0 MODIFY
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:09:18.680 now:20110829_16:09:18.681 agent:
link0
Sched: note:0x28701050 at:20110829_16:09:49.773 now:20110829_16:09:49.774 agent:
link0 MODIFY
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:09:49.773 now:20110829_16:09:49.774 agent:
link0
```



```
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 modify delay=5ms
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 up
[sc558ag@users ~]$ ping nodeB.dynsched.USC558L.isi.deterlab.net
PING pc045.isi.deterlab.net (192.168.1.45): 56 data bytes
64 bytes from 192.168.1.45: icmp_seq=0 ttl=63 time=0.890 ms
64 bytes from 192.168.1.45: icmp_seq=1 ttl=63 time=1.018 ms
64 bytes from 192.168.1.45: icmp_seq=2 ttl=63 time=1.017 ms
64 bytes from 192.168.1.45: icmp_seq=3 ttl=63 time=1.020 ms
^Z
[3]+  Stopped                 ping nodeB.dynsched.USC558L.isi.deterlab.net
[sc558ag@users ~]$ tevc -e USC558L/dynsched now cbr1 stop
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 limit_=75
Unknown event: LIMIT_=75
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 modify limit_=75
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 q_weight_=0.003
Unknown event: Q_WEIGHT_=0.003
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 modify q_weight_=0.003
[sc558ag@users ~]$ tevc -e USC558L/dynsched now link0 modify bandwidth=20000
[sc558ag@users ~]$
```

3.Program Objects:

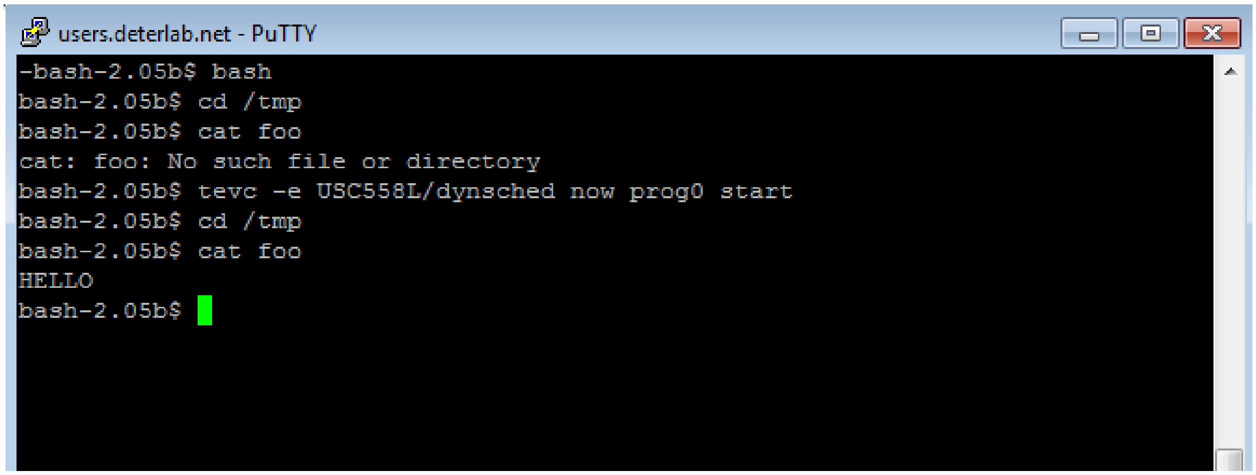
NS Script:

# 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
```

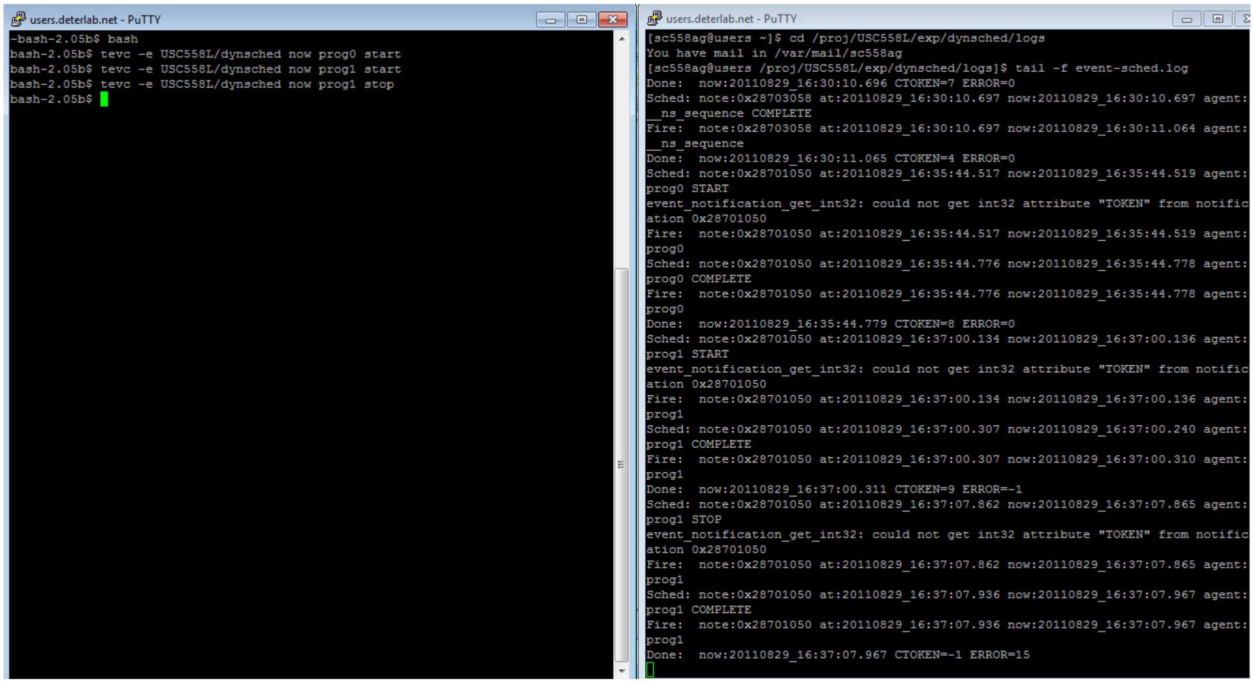
The above ns script allows us to make a program object that prints “HELLO” and puts the output in the file foo under tmp directory.We can see that before the invoking of the tevc command there was no foo file created under the tmp directory but after the tevc command is invoked, the foo file is created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

At node A:



```
-bash-2.05b$ bash
bash-2.05b$ cd /tmp
bash-2.05b$ cat foo
cat: foo: No such file or directory
bash-2.05b$ tevc -e USC558L/dynsched now prog0 start
bash-2.05b$ cd /tmp
bash-2.05b$ cat foo
HELLO
bash-2.05b$ 
```

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.



```
-bash-2.05b$ bash
bash-2.05b$ tevc -e USC558L/dynsched now prog0 start
bash-2.05b$ tevc -e USC558L/dynsched now prog1 start
bash-2.05b$ tevc -e USC558L/dynsched now prog1 stop
bash-2.05b$
```

```
[sc558ag@users ~]$ cd /proj/USC558L/exp/dynsched/logs
You have mail in /var/mail/sc558ag
[sc558ag@users /proj/USC558L/exp/dynsched/logs]$ tail -f event-sched.log
Done:  now:20110829_16:30:10.696 CTOKEN=7 ERROR=0
Sched: note:0x28703058 at:20110829_16:30:10.697 now:20110829_16:30:10.697 agent:
__ns_sequence COMPLETE
Fire:  note:0x28703058 at:20110829_16:30:10.697 now:20110829_16:30:11.064 agent:
__ns_sequence
Done:  now:20110829_16:30:11.065 CTOKEN=4 ERROR=0
Sched: note:0x28701050 at:20110829_16:35:44.517 now:20110829_16:35:44.519 agent:
prog0 START
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:35:44.517 now:20110829_16:35:44.519 agent:
prog0
Sched: note:0x28701050 at:20110829_16:35:44.776 now:20110829_16:35:44.778 agent:
prog0 COMPLETE
Fire:  note:0x28701050 at:20110829_16:35:44.776 now:20110829_16:35:44.778 agent:
prog0
Done:  now:20110829_16:35:44.779 CTOKEN=8 ERROR=0
Sched: note:0x28701050 at:20110829_16:37:00.134 now:20110829_16:37:00.136 agent:
prog1 START
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:37:00.134 now:20110829_16:37:00.136 agent:
prog1
Sched: note:0x28701050 at:20110829_16:37:00.307 now:20110829_16:37:00.240 agent:
prog1 COMPLETE
Fire:  note:0x28701050 at:20110829_16:37:00.307 now:20110829_16:37:00.310 agent:
prog1
Done:  now:20110829_16:37:00.311 CTOKEN=9 ERROR=-1
Sched: note:0x28701050 at:20110829_16:37:07.862 now:20110829_16:37:07.865 agent:
prog1 STOP
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_16:37:07.862 now:20110829_16:37:07.865 agent:
prog1
Sched: note:0x28701050 at:20110829_16:37:07.936 now:20110829_16:37:07.967 agent:
prog1 COMPLETE
Fire:  note:0x28701050 at:20110829_16:37:07.936 now:20110829_16:37:07.967 agent:
prog1
Done:  now:20110829_16:37:07.967 CTOKEN=-1 ERROR=15
```

4.Link Tracing and Monitoring:

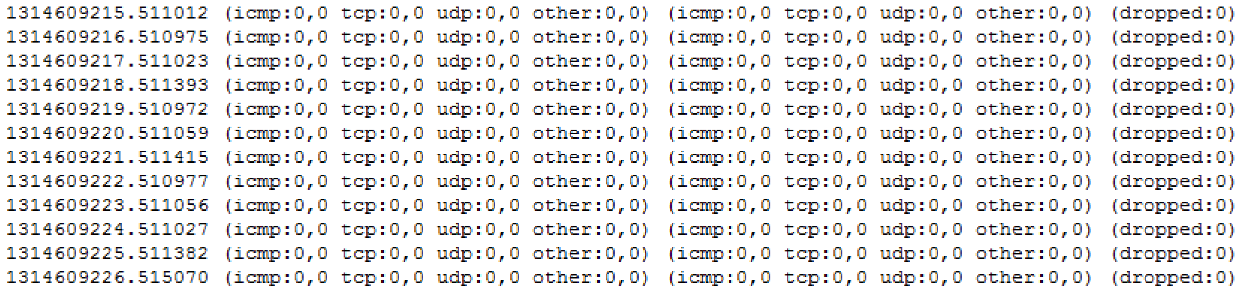
NS Script:

```
# 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.

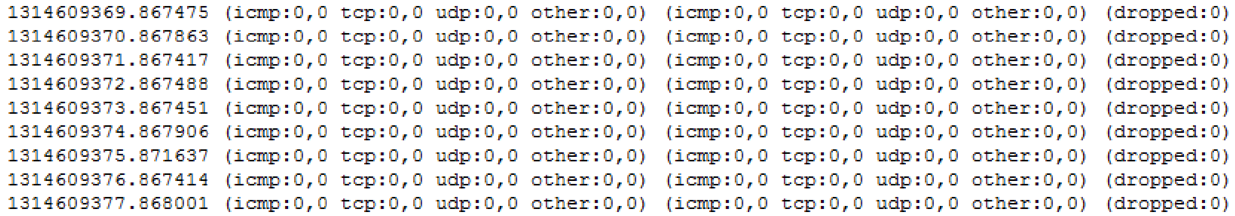
We can monitor the packet capturing real time at node A and node B by using interactive web-interface. We can pause packet capturing, restart it or kill it using the web interface. Below is the output from interactive 'Link Tracing/Monitoring':

At node A:



```
1314609215.511012 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609216.510975 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609217.511023 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609218.511393 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609219.510972 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609220.511059 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609221.511415 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609222.510977 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609223.511056 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609224.511027 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609225.511382 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609226.515070 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
```

At node B:



```
1314609369.867475 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609370.867863 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609371.867417 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609372.867488 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609373.867451 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609374.867906 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609375.871637 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609376.867414 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314609377.868001 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
```

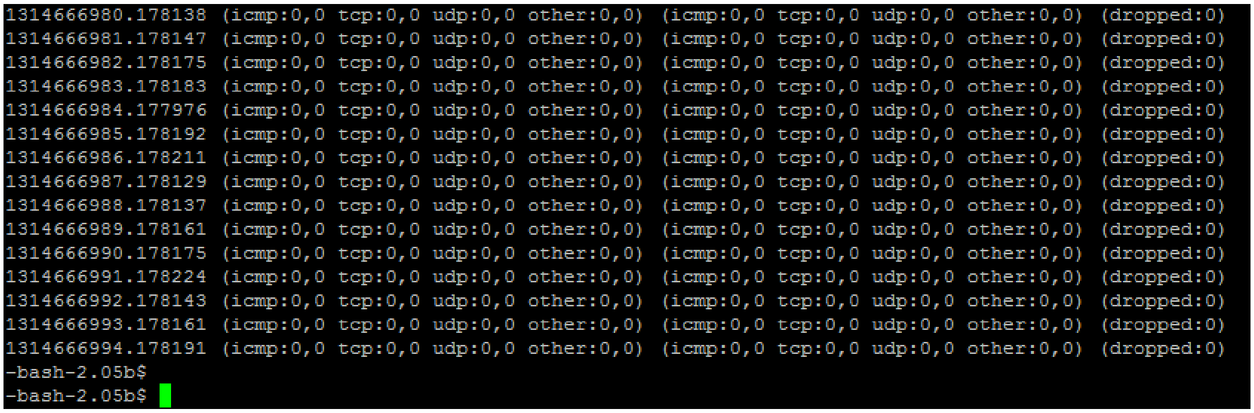
The first thing on all the lines above denotes the timestamp. The number of the brackets show the number of interfaces to that node. Icmp:0,0 indicates that there were 0 bytes of icmp traffic sent in 0 packets.

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.)



```
1314666980.178138 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666981.178147 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666982.178175 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666983.178183 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666984.177976 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666985.178192 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666986.178211 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666987.178129 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666988.178137 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666989.178161 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666990.178175 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666991.178224 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666992.178143 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666993.178161 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
1314666994.178191 (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (icmp:0,0 tcp:0,0 udp:0,0 other:0,0) (dropped:0)
-bash-2.05b$
-bash-2.05b$
```

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:

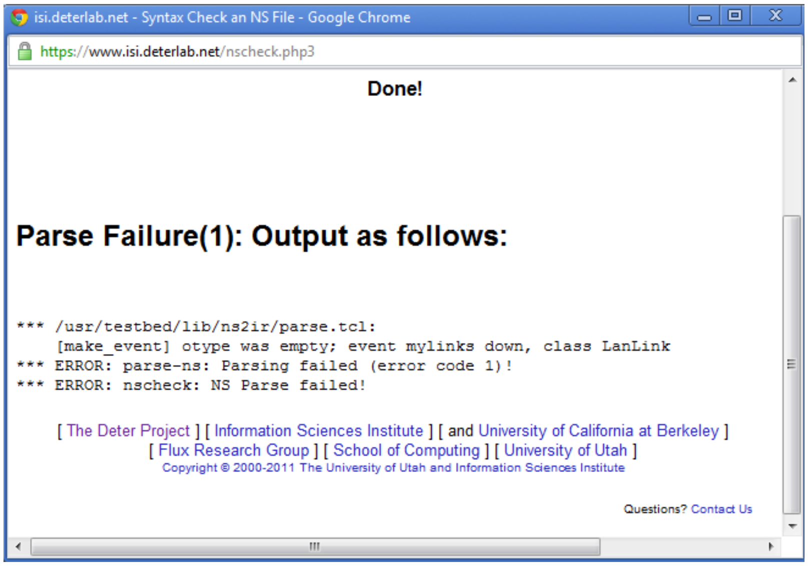
NS Script:

```
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.


isi.deterlab.net - Syntax Check an NS File - Google Chrome
https://www.isi.deterlab.net/nscheck.php3

Done!

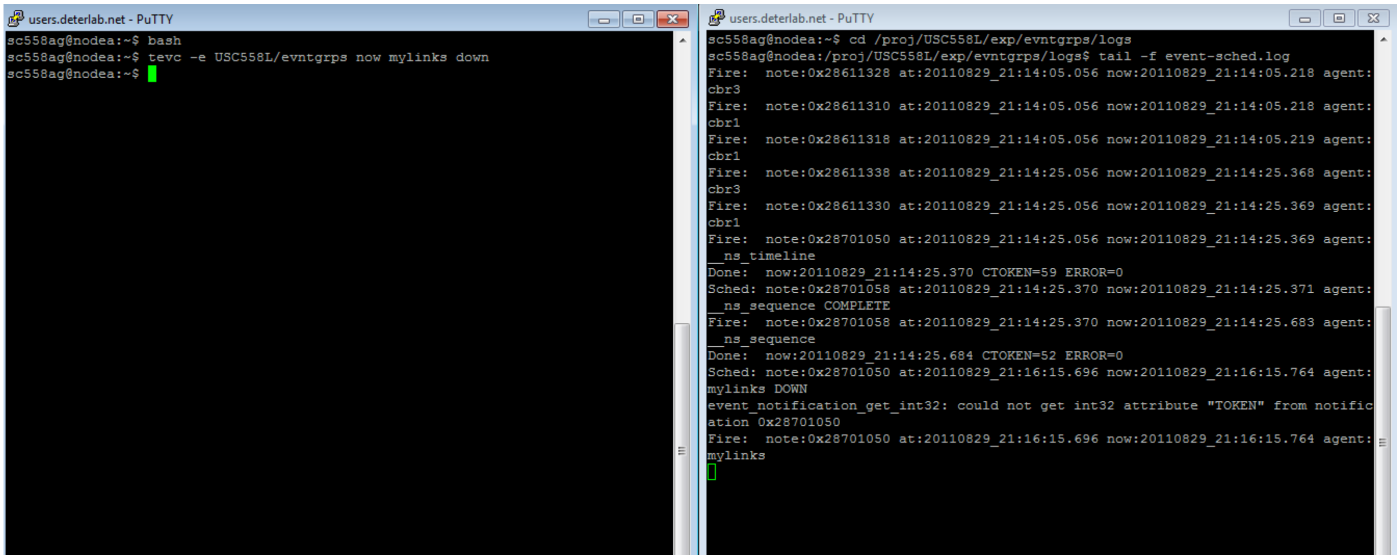
Parse Failure(1): Output as follows:

```
*** /usr/testbed/lib/ns2ir/parse.tcl:
    [make_event] otype was empty; event mylinks down, class LanLink
*** ERROR: parse-ns: Parsing failed (error code 1)!
*** ERROR: nscheck: NS Parse failed!
```

[The Deter Project] [Information Sciences Institute] [and University of California at Berkeley]
[Flux Research Group] [School of Computing] [University of Utah]
Copyright © 2000-2011 The University of Utah and Information Sciences Institute

Questions? Contact Us

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.*



```
sc558ag@nodea:~$ bash
sc558ag@nodea:~$ tevc -e USC558L/evntgrps now mylinks down
sc558ag@nodea:~$
```

```
sc558ag@nodea:~$ cd /proj/USC558L/exp/evntgrps/logs
sc558ag@nodea:/proj/USC558L/exp/evntgrps/logs$ tail -f event-sched.log
Fire:  note:0x28611328 at:20110829_21:14:05.056 now:20110829_21:14:05.218 agent:
cbr3
Fire:  note:0x28611310 at:20110829_21:14:05.056 now:20110829_21:14:05.218 agent:
cbr1
Fire:  note:0x28611318 at:20110829_21:14:05.056 now:20110829_21:14:05.219 agent:
cbr1
Fire:  note:0x28611338 at:20110829_21:14:25.056 now:20110829_21:14:25.368 agent:
cbr3
Fire:  note:0x28611330 at:20110829_21:14:25.056 now:20110829_21:14:25.369 agent:
cbr1
Fire:  note:0x28701050 at:20110829_21:14:25.056 now:20110829_21:14:25.369 agent:
__ns_timeline
Done:  now:20110829_21:14:25.370 CTOKEN=59 ERROR=0
Sched: note:0x28701058 at:20110829_21:14:25.370 now:20110829_21:14:25.371 agent:
__ns_sequence COMPLETE
Fire:  note:0x28701058 at:20110829_21:14:25.370 now:20110829_21:14:25.683 agent:
__ns_sequence
Done:  now:20110829_21:14:25.684 CTOKEN=52 ERROR=0
Sched: note:0x28701050 at:20110829_21:16:15.696 now:20110829_21:16:15.764 agent:
mylinks DOWN
event_notification_get_int32: could not get int32 attribute "TOKEN" from notific
ation 0x28701050
Fire:  note:0x28701050 at:20110829_21:16:15.696 now:20110829_21:16:15.764 agent:
mylinks
```

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.