CNL ASSIGNMENT 07
NS2 NETWORK SIMULATOR

ANTRIKSH SHARMA **27 / 08 / 2022**
20070122021
CS-A1

Objective: Create 4/5 nodes and corresponding links with different bandwidths and delays. Attach agents like TCP, UDP sources, sinks. Attach traffic like FTP, CBR at different time instances. Show simulation and effects of changing bandwidth and delay of different links.

Theory:

TCL

According to the problem statement it is required to create a network which has got a few nodes and these nodes have to be connected in some fashion. Also it is required to attach traffic on the links connected with the nodes. In Network Simulator, a provision is made available namely Tool Command Language. This language can be used to set up any network and traffic can be attached to it. Let us take an example of a ring topology.
In order to create a ring topology, open the terminal in your Linux environment (Note: In order to work on NS it is necessary to first install a compatible version of NS in your Linux environment i.e. Fedora, Ubuntu or Redhat). After opening the terminal, type the following command in it.

```
# gedit ring.tcl
```

The command shown above will open the editor available in the Linux environment named “gedit” and a file named “ring.tcl” will open in it. The extension “.tcl” specifies that the file contains commands written in Tool Command Language. In this file type the following commands in order to create the topology and attaching the traffic on links.

```
set ns [new Simulator]
$ns color 1 Blue
$ns color 2 Red $ns color 3
Black set nf [open new.nam w]
$ns namtrace-all $nf set tf [open
testout.tr w] $ns trace-all $tf set
n0 [$ns node] set n1 [$ns node]
set n2 [$ns node] set n3 [$ns
node] set n4 [$ns node] set n5

[$ns node] set n6 [$ns node] set
n7 [$ns node]
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 [lindex $argv 0] 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 10ms DropTail
$ns duplex-link $n3 $n4 2Mb 10ms DropTail
$ns duplex-link $n4 $n5 2Mb 10ms DropTail
$ns duplex-link $n5 $n6 2Mb 10ms DropTail
$ns duplex-link $n6 $n7 2Mb 10ms DropTail
$ns duplex-link $n7 $n0 2Mb 10ms DropTail
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n3 10
$ns queue-limit $n3 $n4 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n5 $n6 10
$ns queue-limit $n6 $n7 10 $ns
queue-limit $n7 $n0 10 set tcp [new
Agent/TCP] $ns attach-agent $n0
$tcp set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
```

```
$tcp set fid_ 1 set ftp [new
Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start" $ns at 4.0
"$ftp stop" proc finish {} {
global ns nf $ns
flush-trace close $nf
#close $tf
exec nam new.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

Implementing

The first line of the code is
set ns [new Simulator]
It creates a new simulator for the commands that follows.
The next lines are
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Black
The above lines allocate the color of the traffics which can be viewed in the Network Animator
discussed later.
The next lines are
set nf [open new.nam w]
$ns namtrace-all $nf

The above lines open a file called “new.nam” which will contain all the information regarding the
simulation to be seen using the Network Simulator.
The next lines are
set tf [open testout.tr w]
$ns trace-all $tf
The above lines open a file called “testout.tr”. This file is a trace file which contains all the
information regarding the data flow of the network. It could be data bytes sent, received,
dropped. It also contains the timing instances when the particular task has taken place and so
on.
The next lines are
set n0 [$ns node] set n1
[$ns node] set n2 [$ns
node] set n3 [$ns node] set
n4 [$ns node] set n5 [$ns
node] set n6 [$ns node] set
n7 [$ns node]
The above lines will create 8 nodes namely n0 through n7. These nodes can be used to form a

network of our desire. They can be made either a source or sink or just a router.
The next lines are

$ns duplex-link $n0 $n1 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 [lindex $argv 0] 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 10ms DropTail
$ns duplex-link $n3 $n4 2Mb 10ms DropTail
$ns duplex-link $n4 $n5 2Mb 10ms DropTail
$ns duplex-link $n5 $n6 2Mb 10ms DropTail
$ns duplex-link $n6 $n7 2Mb 10ms DropTail
$ns duplex-link $n7 $n0 2Mb 10ms DropTail

The above lines are used to make connections between various nodes. For example in the above lines a duplex link is set up between nodes n0 and n1, n1 and n2, n2 and n3 and so on.
Also the above commands attach the link bandwidths like 2Mb and the link delay like 10ms as shown above. Then a term “DropTail” is written. This is a command specifying what to do if there is congestion on that particular link. In case of Droptail, it will start dropping the data bytes if congestion occurs in that link. Note that on the duplex link between n1 and n2 the bandwidth is not specified. Instead, a command “[lindex $argv 0]” is written. This command is for allowing the users to enter the command line arguments (in this case the bandwidth on link between n1 and n2) at run time. Also note that the fashion in which the nodes are been connected we get a complete ring topology as we have taken the example of a ring topology.
The next lines are

$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 10
$ns queue-limit $n2 $n3 10
$ns queue-limit $n3 $n4 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n5 $n6 10
$ns queue-limit $n6 $n7 10
$ns queue-limit $n7 $n0 10

The above commands specify the buffer size or the queue limits on each of the links created before.
The next lines are

set tcp [new Agent/TCP] $ns attach-
agent $n0 $tcp set sink [new

Agent/TCPSink] $ns attach-agent $n3
$sink $ns connect $tcp $sink
$tcp set fid_ 1 set ftp [new

Application/FTP]
$ftp attach-agent $tcp
The above commands are used to attach TCP source to node n0 and a corresponding sink to
node n3.
The next lines are
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
The above commands specify the instances when the ftp traffic has to be started and stopped.
And the last lines of the code are

```
proc finish {} {
global ns nf $ns
flush-trace
close $nf
#close $tf
exec nam new.nam &
exit 0
}
$ns at 5.0 "finish"
$ns run
```

The above commands specify the instance when the simulation has to be stopped and also the
command “exec nam new.nam” will run the simulation using the information stored in the file
called “new.nam”.

NAM
After having written the code and now it is required to run the code. In order to ensure that
whether the topology has been set up properly and the traffic is flowing or not NS has got the
facility of Network AniMator commonly called as NAM. Since in the code itself it is mentioned
that the information of the topology has to be stored in the .nam file and also it is specified to
execute the nam file, in the terminal we just need to write the following command.

```
# ns ring.tcl 2Mb
```

Here, note that the command “ns ring.tcl” runs the .tcl file and the commands written in it.
Moreover, a provision is made available to the uses to enter command line argument at run
time. This command line argument is the bandwidth as explained earlier and is specified as
“2Mb” in the above command. This code will open the Animator with the topology created by
the previous code. On clicking the run and stop buttons, the user can run and stop the traffic in

the network created.
The following is a snapshot of the animator running the ring topology and the TCP traffic in it.

AWK
After running the animator, it is now required to do analysis of the trace file which contains all
the information about the data transfer in the network. The following is an example of a trace
file.

+ 1 0 1 tcp 40 ------- 1 0.0 3.0 0 0 - 1 0 1 tcp 40 -----
-- 1 0.0 3.0 0 0 r 1.01016 0 1 tcp 40 ------- 1 0.0 3.0
0 0 + 1.01016 1 2 tcp 40 ------- 1 0.0 3.0 0 0 -
1.01016 1 2 tcp 40 ------- 1 0.0 3.0 0 0 r 1.02032 1
2 tcp 40 ------- 1 0.0 3.0 0 0 + 1.02032 2 3 tcp 40 ---
---- 1 0.0 3.0 0 0 - 1.02032 2 3 tcp 40 ------- 1 0.0
3.0 0 0 r 1.03048 2 3 tcp 40 ------- 1 0.0 3.0 0 0 +
1.03048 3 2 ack 40 ------- 1 3.0 0.0 0 1 - 1.03048 3

2 ack 40 ------- 1 3.0 0.0 0 1 r 1.04064 3 2 ack 40 --
----- 1 3.0 0.0 0 1 + 1.04064 2 1 ack 40 ------- 1 3.0
0.0 0 1
- 1.04064 2 1 ack 40 ------- 1 3.0 0.0 0 1

The following is the format in which the trace file is being created
event Time Source
node
Destination
node
Packet
type
Packet
size
flags fid Source
address
Dest.
address
Seq.
number
Packet
id

r : receive (at source node) + :
enqueue (at queue) - : dequeue
(at queue)
d : drop (at queue)
But a problem with the trace file is that it has got exhaustive information content which may

prove to be tedious to analyze. So a tool called AWK comes to our rescue. This tool helps us to
analyze the trace file with ease.
Suppose we need to find out the success rate of the network then with the help of awk we can
make the analyses of the trace file and find the success rate. For that first of all type the
following command in the terminal.

```
# gedit packetr.awk
```

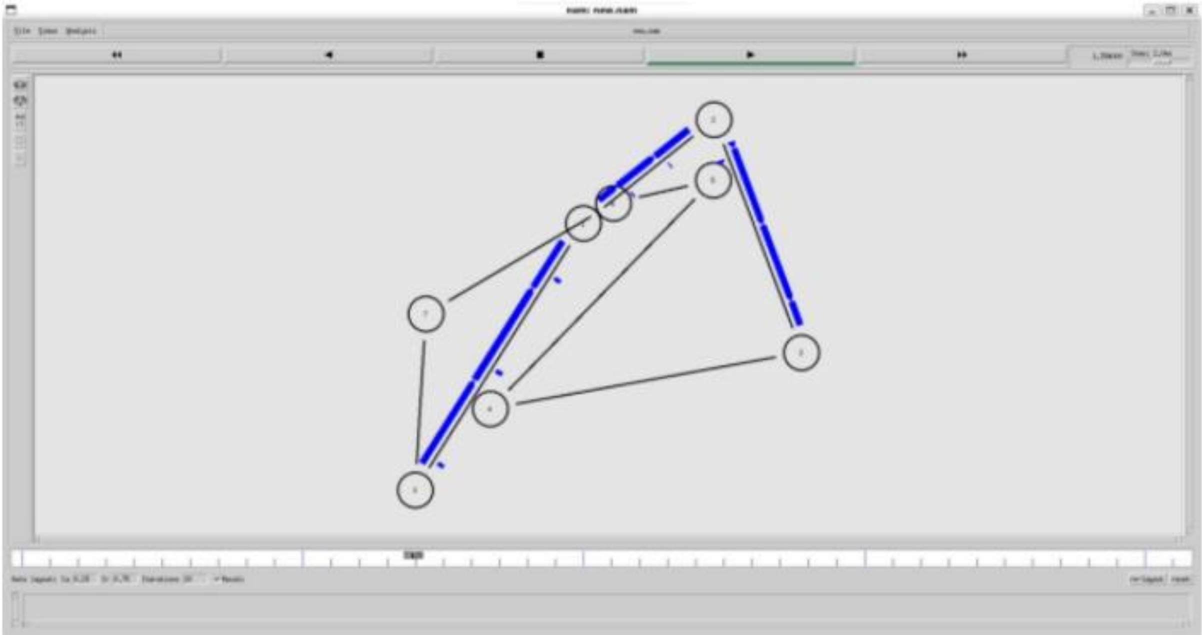
The above command will open a new file with the extension ".awk" in the editor named gedit. In
this file type in the following commands.

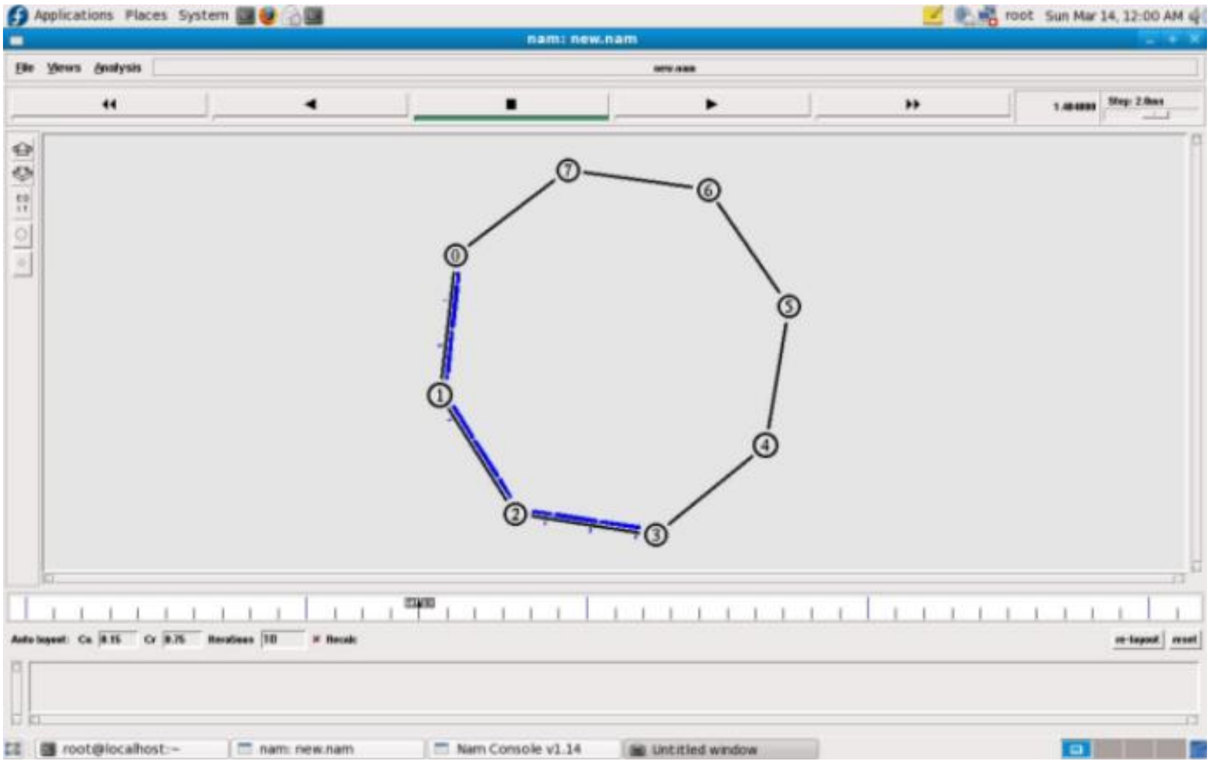
```
BEGIN{ rec=1;
send=1;
rate=0;
} {
if ( $1 == "+" && $5 == "tcp" && $3 == "0")
{
;print $0 ;
send = send + $6
}
if ( $1 == "r" && $5 == "tcp" && $4 == "3")
{
;print $0;
rec = rec + $6
}
rate=(rec/send)*100;
}
END{ printf("\n packet Send %d", send);
printf("\n packet recv %d", rec);
printf("%f\n", rate)>>"result1.log";
}
```

This awk file will help us to find the success rate of our network. The coding of this awk file is
quite similar to C language so it is easy to understand the code and to write it as well. This awk

file stores the value of the success rate in a log file named "result1.log". to run the awk file type
in the following command in the terminal

```
# awk –f packetr.awk testout.tr
```




Applications Places System
root Sun Mar 14, 12:00 AM
nam: new.nam
File Views Analysis
root@localhost:~
nam: new.nam
Nam Console v1.14
Untitled window