Lab 3A Understanding network protocols thru simulations - Network Simulator (ns2)

A *simulator* is a program that behaves like anotther system, which allows you to study and analyze complex systems like networks. Simulators consist of objects that represent the entities in the system, and timed “events” that represent the changes happening in the system Events are created according to some rates and times specified by the user. Events change the states of objects in the simulator, and this change can result in more events. This is how a simulation continues.

Any *network* simulator typically takes as input the following:

- Description of the physical topology of the network - the nodes, the links, the bandwidth and delay specifications of the links.
- Description of “flows” from end hosts of in this network. The application type, the packet size, packet generation rate, etc.
- Specification of protocols to be simulated by the hosts and routers of the network. Protocol parameters if any (e.g. window size).

A network simulator then creates the underlying node objects, link objects, dynamically creates packet objects at the specified rates, and queues these packets at links or nodes at times corresponding to given bandwidths and delays. It records the timestamps of various events (e.g. packet arrivals, drops, etc) as they happen in the system and logs them into a *trace file*. You can then process this trace file to get aggregate quantities such as throughputs, packet delays, drop rates etc.

ns2- Network Simulator-2

You will be using ns2 in Lab3B to understand the Sliding Window Protocol. Before that, you need to learn to use ns2. This is what you are doing in Lab 3A.

The following tutorial (by Mark Greis - it is a popular tutorial found on the net) is a must before you can do the actual task of the lab. ns2 accompanied by “**nam**” a network animator, that allows you to visualize the simulation.

In this section, you are going to develop a Tcl script for ns which simulates a simple topology. You are going to learn how to set up nodes and links, how to send data from one node to another, how to monitor a queue and how to start nam from your simulation script to visualize your simulation.

How to start

Now we are going to write a 'template' that you can use for all of the first Tcl scripts. You can write your Tcl scripts in any text editor (e.g. gedit). Open a file **lab3a.tcl.**
Now read, understand, and start typing as instructed. Everything to be typed into lab3a.tcl is in
this colour and font.

First of all, you need to create a simulator object. This is done with the command

```
set ns [new Simulator]
```

Now we open a file for writing that is going to be used for the nam trace data.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Open a file for creating an ascii trace file

```
set f [open simple.tr w]
$ns trace-all $f
```

The first line opens the file 'out.nam' for writing and gives it the file handle 'nf'. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file.
The next step is to add a 'finish' procedure that closes the trace file and starts nam.

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

You don't really have to understand all of the above code yet. It will get clearer to you once you see what the code does.
The next line tells the simulator object to execute the 'finish' procedure after 5.0 seconds of simulation time.

```
$ns at 5.0 "finish"
```

You probably understand what this line does just by looking at it. ns provides you with a very simple way to schedule events with the 'at' command.
The last line finally starts the simulation.

```
$ns run
```

You can actually save the file now and try to run it **at the bash terminal** as follows

```
ns lab02.tcl
```

You are going to get an error message like 'nam: empty trace file out.nam' though, because until now we haven't defined any objects (nodes, links, etc.) or events.

Two nodes, one link

In this section we are going to define a very simple topology with two nodes that are connected by a link. The following two lines define the two nodes. (Note: You have to insert the code in this section before the line '$ns run', or even better, before the line '$ns at 5.0 "finish"').

```
set n0 [$ns node]
set n1 [$ns node]
```

A new node object is created with the command '$ns node'. The above code creates two nodes and assigns them to the handles 'n0' and 'n1'.
The next line connects the two nodes.

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

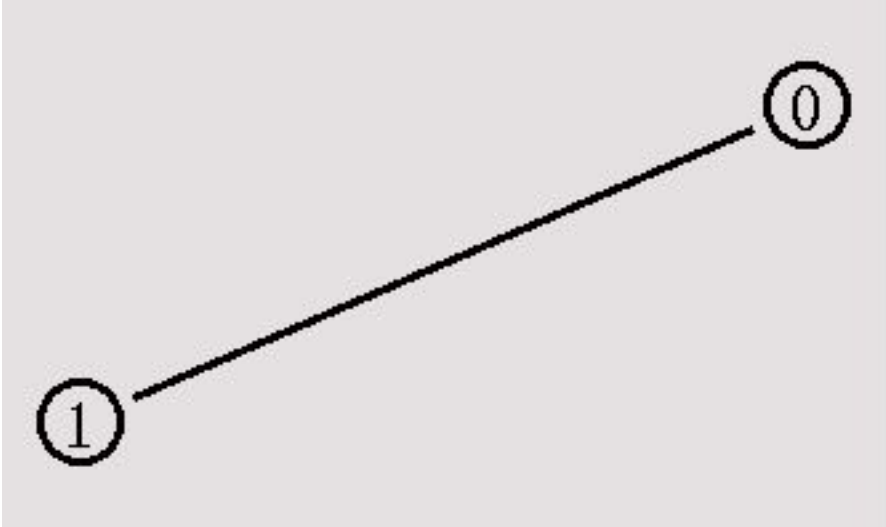
This line tells the simulator object to connect the nodes n0 and n1 with a duplex link with the bandwidth 1Megabit/s, a propagation delay of 10ms and a DropTail queue. “Droptail” is nothing but the normal FIFO queuing policy in which packets which arrive to find the buffer full are dropped.

Add a queue limit to this link:

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

This means that 20 packets can be waiting to be transmitted at this link. If a packet arrives when 20 packets are already waiting this new packet will be DROPPED.

Now you can save your file and start the script with 'ns example1.tcl'. nam will be started automatically and you should see an output that resembles the picture below.



Sending data

Of course, this example isn't very satisfying yet, since you can only look at the topology, but nothing actually happens, so the next step is to send some data from node n0 to node n1. In ns, data is always being sent from one 'agent' to another. So the next step is to create an agent object that sends data from node n0, and another agent object that receives the data on node n1.

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $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
```

These lines create a UDP agent and attach it to the node n0, then attach a CBR traffic generator to the UDP agent. CBR stands for 'constant bit rate'. Constant Bit Rate is traffic type in which a packet of fixed size is generated at every fixed interval. There is no variation. It is the kind of network traffic generated by voice or music over the network. This kind of data is actually continuously generated, but it is digitized, made into fixed size packets that are made every fixed interval, and are played out exactly at that interval at the destination - Line 7 and 8 (out of the 9 lines above) should be self-explaining. The packetSize is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second).
The next lines create a Null agent which acts as traffic sink and attach it to node n1.

```
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

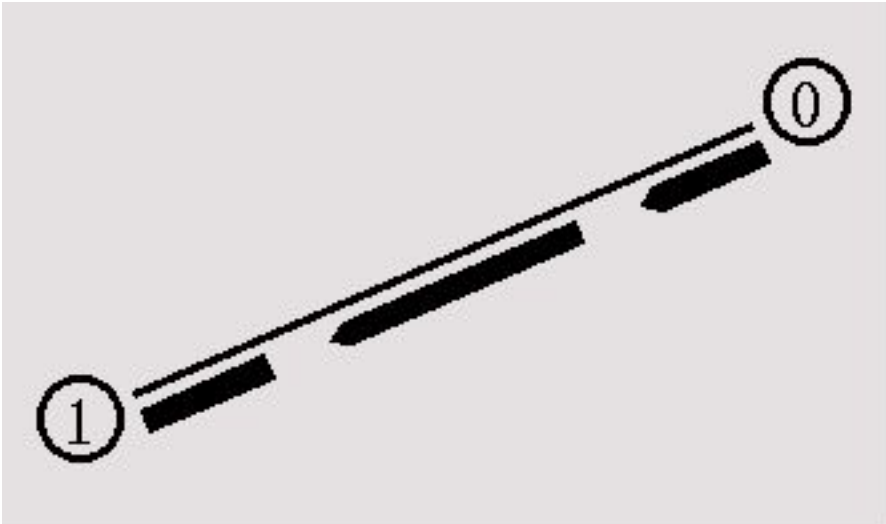
Now the two agents have to be connected with each other.

```
$ns connect $udp0 $null0
```

And now we have to tell the CBR agent when to send data and when to stop sending. Note: It's probably best to put the following lines just before the line '$ns at 5.0 "finish"'.

```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

This code should be self-explaining again.
Now you can save the file and start the simulation again. When you click on the 'play' button in the nam window, you will see that after 0.5 simulation seconds, node 0 starts sending data packets to node 1. You might want to slow nam down then with the 'Step' slider.



I suggest that now you start some experiments with nam and the Tcl script. You can click on any packet in the nam window to monitor it, and you can also click directly on the link to get some graphs with statistics.

Run the simulation script you created, and the run the network animator. Play with nam any way you wish. See the text trace. See here for the interpretation of the trace.
Now answer the following questions on bodhitree Report Grading.

1. From the trace file produced by the simulation, how many packet drops do you see? Do this by using “grep” to pick the lines from simple.tr corresponding to drops, and then piping to “wc” (word count) which gives #of lines, words and characters in any file (grep … simple.tr | wc) . Figure out the grep command by looking at simple.tr (Guide is given in the lab document).
2. Change the simulation setup in any way you want to see some packet drops. Try two different ways of getting packet drops.
    a. Method 1 of getting drops
    b. Number of drops with method 1.

c. Method 2 of getting drops
d. Number of drops with method 2:

ns2 trace guide

(reproduced from: http://ns2ultimate.tumblr.com/post/2496927327/post-processing-ns2-result-using-ns2-trace-trace)

Trace file format

The format of a trace string is shown below:

Type Identifier	Time	Source Node	Destination Node	Packet Name	Packet Size	Flags	Flow ID	Source Address	Destination Address	Sequence Number	Packet Unique ID

where 12 fields of the trace string are as follows.

1. Type Identifier:
   - "+": a packet enque event
   - "-": a packet deque event
   - "r": a packet reception event
   - "d": a packet drop (e.g., sent to dropHead_) event
   - "c": a packet collision at the MAC level

2. Time: at which the packet tracing string is created.

3-4. Source Node and Destination Node: denote the IDs of the source and the destination nodes of the tracing object.

5. Packet Name: Name of the packet type

6. Packet Size: Size of the packet in bytes.

7. Flags: A 7-digit flag string
   - "-": disable
   - 1st = "E": ECN (Explicit Congestion Notification) echo is enabled.
   - 2nd = "P": the priority in the IP header is enabled.
   - 3rd : Not in use
   - 4th = "A": Congestion action
   - 5th = "E": Congestion has occurred.
   - 6th = "F": The TCP fast start is used.
   - 7th = "N": Explicit Congestion Notification (ECN) is on.

8. Flow ID

9-10. Source Address and Destination Address: the format of these two fields is "a.b", where "a" is the address and "b" is the port.

11. Sequence Number

12. Packet Unique ID