COMP3014 Network Simulation Project
Part B: Network Simulator Setup

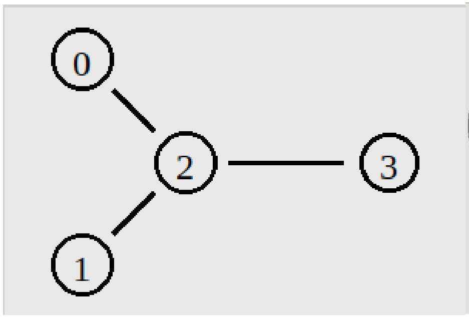
Nima Afraz

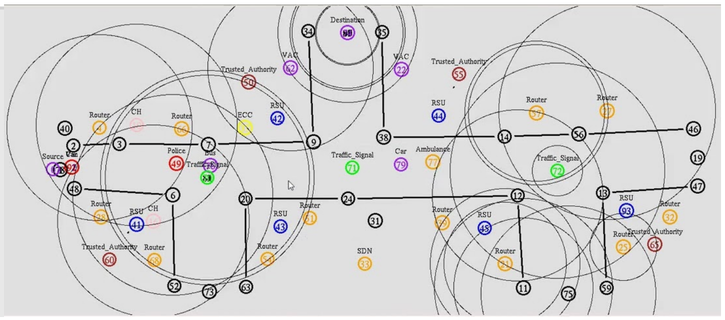
September 5, 2022

1 The NS2 Components

1.1 Topology

The network topology could be very simple as you have seend in Part A's test file, or very complex.





The following TCL code shows the topology of the test.tcl example in Part A.

```
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

1.2 Scenario

Like a theatre play a simulation also has a scenario. For example a simulation scenario could specify when (what time) a network node should send a packet, to which node, using what protocol. Similar to a play where the scenario will specify what actor should say what, to whom and in what language.

```
#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
```

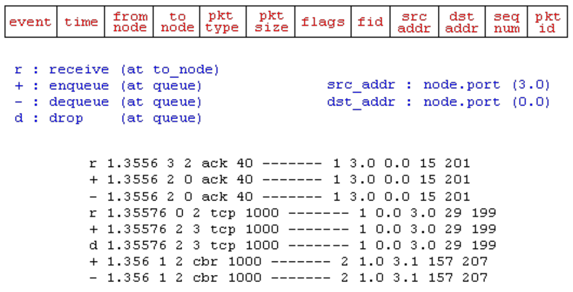
```
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
```

1.3 Trace File

The Trace file (.tr) is where ns2 records the outcome of the simulation in the format of events. In the following code you can see how we enable tracing in a Tcl file.

```
. . .

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Close the Trace file
        close $tf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

. . .
```

Each line of the trace file could be read as for example a packet was received in node x that was send from node y and the packet size and type was TCP and 1000). This file usually gets quite long for simulations with longer duration and larger number of nodes (it could be hundreds of thousands of lines). Therefore, it is almost impossible to manually analyse the results of the simulation and we often use scripting languages such as awk or Python to extract meaning (analyse) from the trace file.



event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)            src_addr : node.port (3.0)
- : dequeue (at queue)            dst_addr : node.port (0.0)
d : drop    (at queue)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

1.4 Trance Data Analyser Script and Plotting

Having simulation trace data at hand, all one has to do is to transform a subset of the data of interest into a comprehensible information and analyze it. Down below is a small data transformation example. This example uses a command written in perl called "column" that selects columns of given input. To make the example work on your machine, you should download "column" and make it executable (i.e. "chmod 755 column"). Following is a tunneled shell command combined with awk, which calculates CBR traffic jitter at receiver node (n3) using data in "out.tr", and stores the resulting data in "jitter.txt".

```
cat out.tr | grep " 2 3 cbr " | grep ^r | column 1 10 |
awk '{dif = $2 - old2; if(dif==0) dif = 1; if(dif > 0) {
printf("%d\t%f\n", $2, ($1 - old1) / dif); old1 = $1; old2 = $2}}' > jitter.txt
```

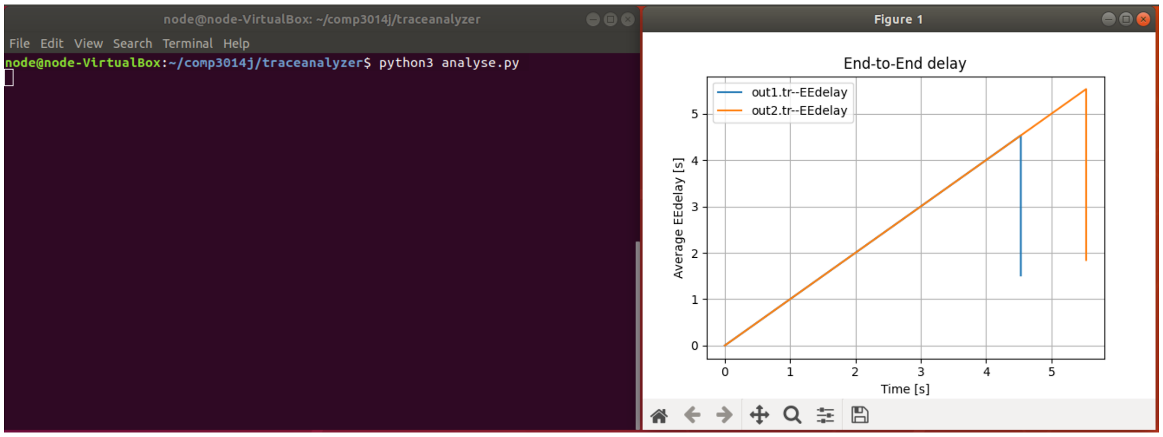
The above script can be found at Test/jitter.sh file on GitLab repository.

This shell command selects the "CBR packet receive" event at n3, selects time (column 1) and sequence number (column 10), and calculates the difference from last packet receive time divided by difference in sequence number (for loss packets) for each sequence number. The following is the corresponding jitter graph that is generated using gnuplot. The X axis show the packet sequence number and the Y axis shows simulation time in seconds.

1.5 Python library to Analyse .tr files

There is a Python library called 'traceanalyzer' that alalyses ns2 trace files and generates meaningful metrics. traceanalyzer provide four main classes : Eedelay, Pdr, Throughput and Nrl. These main classes compute the average of end-to-end delay, the Packet Delivery Ratio, the Throughput and the Normalized Routing Load from ns2 trace file. Those provide also plotting and data array extracted from ns2 trace file.

```
#! /bin/bash
# Install traceanalyzer
pip3 install traceanalyzer
cd ~/comp3014j/traceanalyzer
python3 analyse.py
```



```
node@node-VirtualBox:~/comp3014j/traceanalyzer$ python3 analyse.py
```