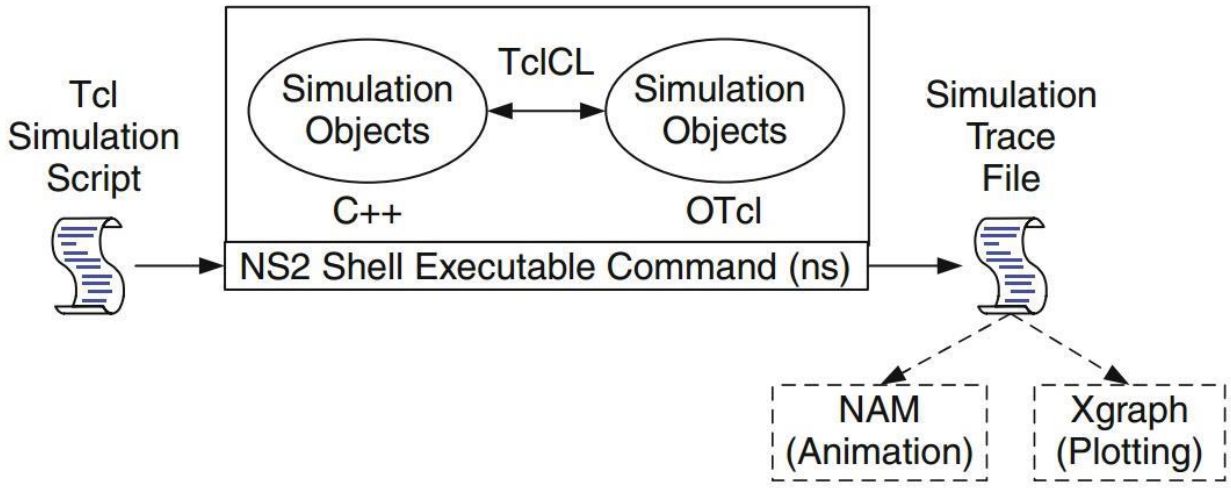
STUDY OF NETWORK SIMULATOR (NS)

Introduction:

- Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks.
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviours.

Basic Architecture:



- Above figure shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.
- In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

- Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively.
- NS2 provides a large number of built-in C++ objects.
- After simulation, NS2 outputs either text-based or animation-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used.
- NS uses two languages, C++ and OTcl.

Tcl scripting:

Tcl is a general-purpose scripting language. [Interpreter]

- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3
puts stdout{Hello, World!} Hello, World!

Hello World!

Variables Command Substitution

set a 5 set len [string length foobar]
set b $a set len [expr [string length foobar] + 9]

Wired TCL Script Components

1. Create the event scheduler
2. Open new files & turn on the tracing
3. Create the nodes
4. Setup the links
5. Configure the traffic type (e.g., TCP, UDP, etc)
6. Set the time of traffic generation (e.g., CBR, FTP)
7. Terminate the simulation

NS Simulator Preliminaries:

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

- Which is thus the first line in the tcl script. This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object [new Simulator] is indeed the installation of the class Simulator using the reserved word new.

- In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using —open command:

#Open the Trace file

set tracefile1 [open out.tr w]

$ns trace-all $tracefile1

#Open the NAM trace file

set namfile [open out.nam w]

$ns namtrace-all $namfile

- The above creates a trace file called out.tr and a nam visualization trace file called out.nam.
- Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called —tracefile1 and —namfile respectively.

- Remarks begins with a # symbol. The second line open the file —out.tr to be used for writing, declared with the letter —w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

Define a "finish" procedure

```
Proc finish { } {
        global ns tracefile1 namfile
        $ns flush-trace
        Close $tracefile1
        Close $namfile
        Exec nam out.nam &
        Exit 0
}
```

Definition of a network of links and nodes

- To define a node is

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

- Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

  ```
  $ns duplex-link $n0 $n2 10Mb 10ms DropTail
  ```

- Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.
- To define a directional link instead of a bi-directional one, we should replace ***—duplex-link*** by ***—simplex-link***.
- In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20
        $ns queue-limit $n0 $n2 20
```

FTP over TCP

- TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.
- There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

set tcp [new Agent/TCP]

- The command ***$ns attach-agent $n0 $tcp*** defines the source node of the tcp connection.
- The command ***set sink [new Agent /TCPSink]*** defines the behaviour of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_2

#setup a CBR over UDP connection

- The command ***$ns attach-agent $n4 $sink*** defines the destination node. The command ***$ns connect $tcp $sink*** finally makes the TCP connection between the source and destination nodes.

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false

- TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000 bytes. This can be changed to another value, say 552bytes, using the command ***$tcp set packetsize_ 552***.
- When we have several flows, we may wish to distinguish them so that we can identify them with different colours in the visualization part. This is done by the command ***$tcp set fid_ 1*** that assigns to the TCP connection a flow identification of —1. We shall later give the flow identification of —2‖ to the UDP connection.