**Aim:** demonstrate communication between nodes using NS-2

**Theory:**

S2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks.

**Features of NS2**

1. It is a discrete event simulator for networking research.

2. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.

3. It simulates wired and wireless network.

4. It is primarily Unix based.

5. Uses TCL as its scripting language.

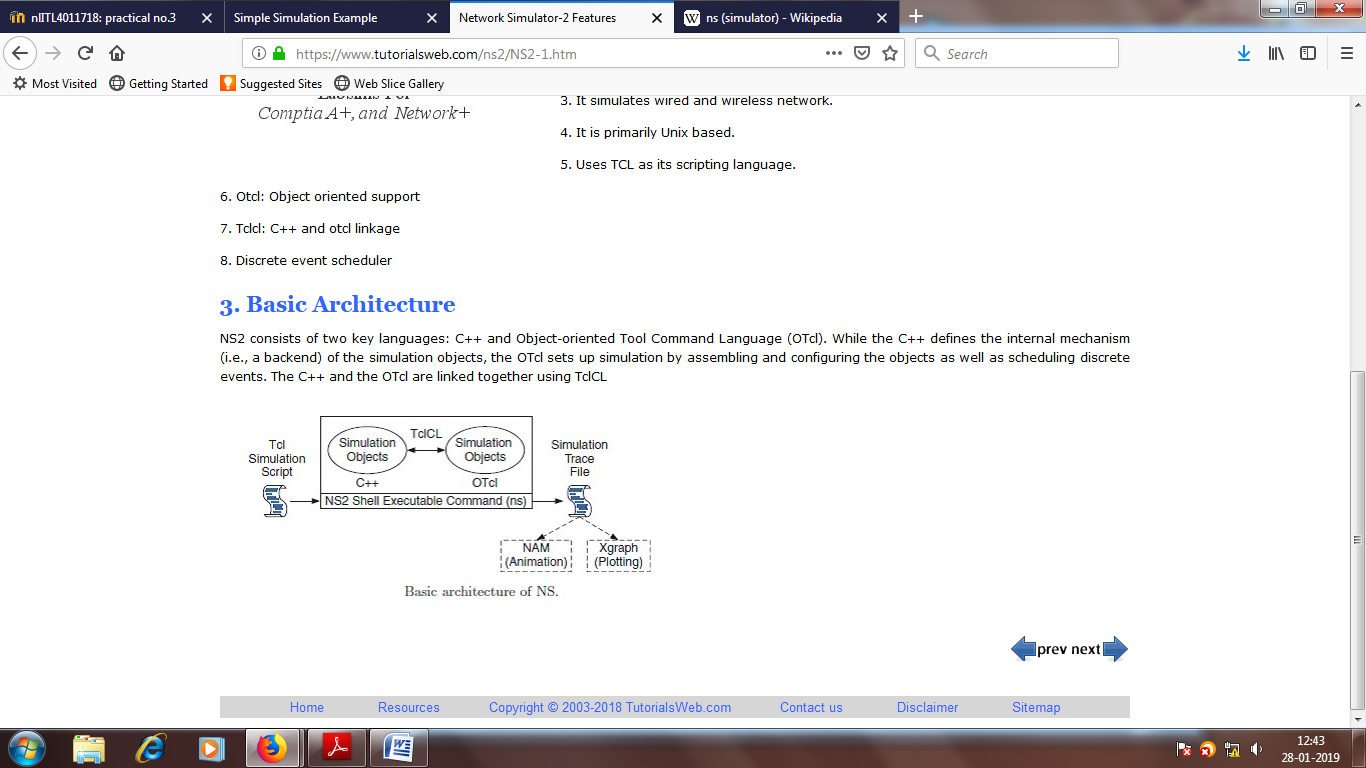
6. Otcl: Object oriented support

7. Tclcl: C++ and otcl linkage

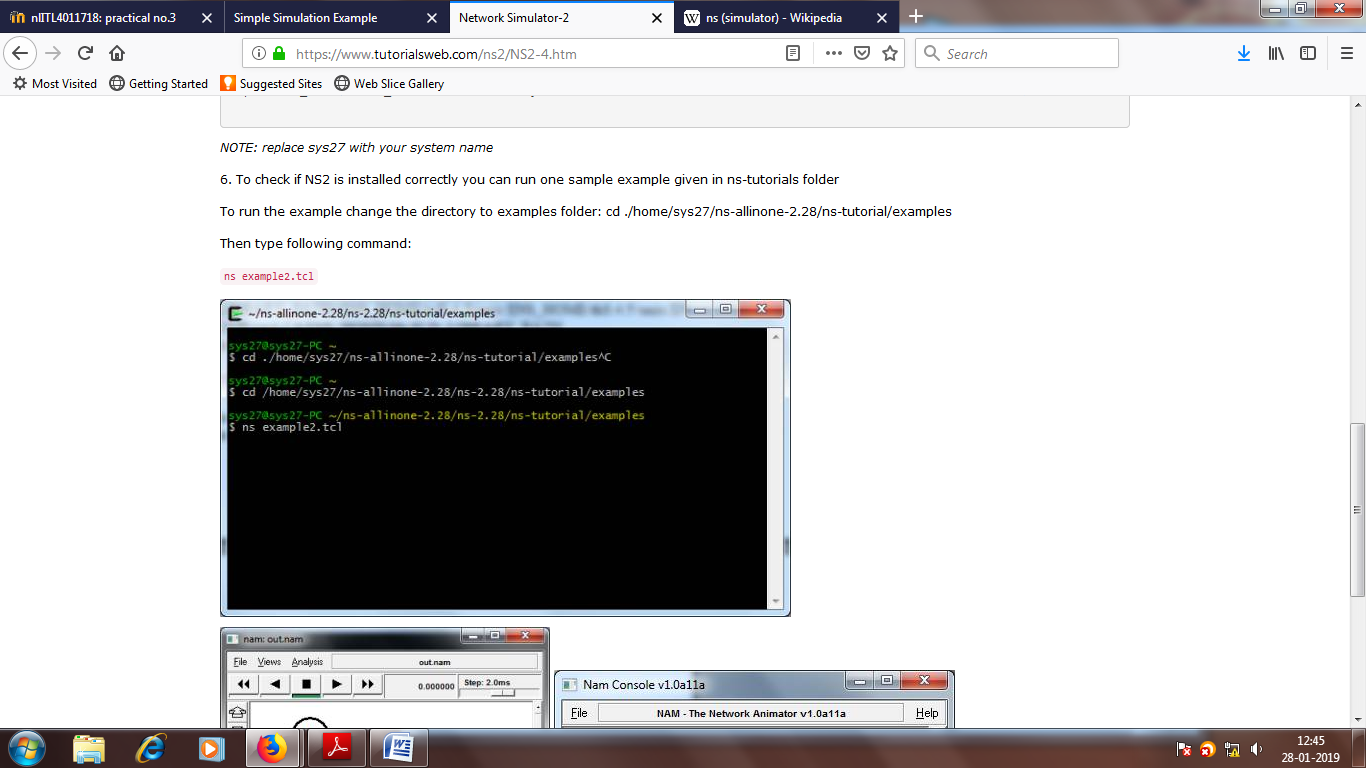
8. Discrete event scheduler

## Basic Architecture

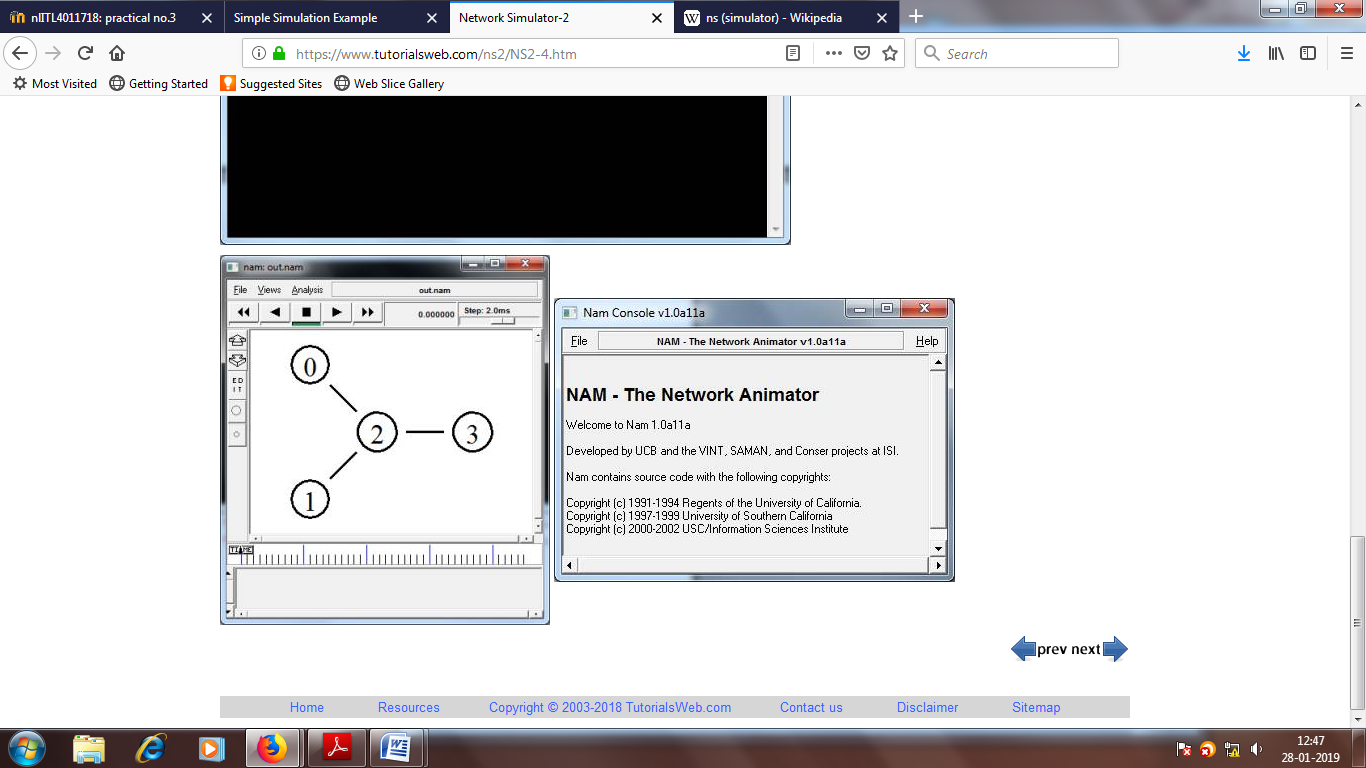
NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL



To check if NS2 is installed correctly you can run one sample example given in ns-tutorials folder

****

**The output will be displayed as follows**



**Advantages and Disadvantages of NS2**

**Advantages**

* Cheap- Does not require costly equipment
* Complex scenarios can be easily tested.
* Results can be quickly obtained-more ideas can be tested in a smaller time frame.
* Supported protocols
* Supported platforms
* Modularity
* Popular

**Disadvantages**

* Real system too complex to model. i.e. complicated structure.
* Bugs are unreliable

**Parameters used in the NS2 script**

**set *ns* [new Simulator]:** generates an NS simulator object instance, and assigns it to variable *ns* (italics is used for variables and values in this section). What this line does is the following:

* + Initialize the packet format (ignore this for now)
  + Create a scheduler (default is calendar scheduler)
  + Select the default address format (ignore this for now)

The "Simulator" object has member functions that do the following:

* + Create compound objects such as nodes and links (described later)
  + Connect network component objects created (ex. attach-agent)
  + Set network component parameters (mostly for compound objects)
  + Create connections between agents (ex. make connection between a "tcp" and "sink")
  + Specify NAM display options
  + Etc.

Most of member functions are for simulation setup (referred to as plumbing functions in the Overview section) and scheduling, however some of them are for the NAM display. The "Simulator" object member function implementations are located in the "ns-2/tcl/lib/ns-lib.tcl" file.

* ***$ns* color *fid color*:** is to set color of the packets for a flow specified by the flow id (fid). This member function of "Simulator" object is for the NAM display, and has no effect on the actual simulation.
* ***$ns* namtrace-**all *file-descriptor*: This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command *$ns* flush-trace. Similarly, the member function trace-all is for recording the simulation trace in a general format.
* **proc *finish* {}:** is called after this simulation is over by the command *$ns* at 5.0 "*finish*". In this function, post-simulation processes are specified.
* **set *n0* [*$ns* node]:** The member function node creates a node. A node in NS is compound object made of address and port classifiers (described in a later section). Users can create a node by separately creating an address and a port classifier objects and connecting them together. However, this member function of Simulator object makes the job easier. To see how a node is created, look at the files: "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-node.tcl".
* *$****ns* duplex-link *node1 node2 bandwidth delay queue-type***: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used. If the reader wants to use a RED queue, simply replace the word DropTail with RED. The NS implementation of a link is shown in a later section. Like a node, a link is a compound object, and users can create its sub-objects and connect them and the nodes. Link source codes can be found in "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl" files. One thing to note is that you can insert error modules in a link component to simulate a lossy link (actually users can make and insert any network objects).
* ***$ns* queue-limit *node1 node2 number***: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified. At this point, the authors do not know how many of these kinds of member functions of Simulator objects are available and what they are. Please take a look at "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl",.
* ***$ns* duplex-link-op *node1 node2*** *...*: The next couple of lines are used for the NAM display. To see the effects of these lines, users can comment these lines out and try the simulation.

Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

* **set *tcp* [new *Agent/TCP*]:** This line shows how to create a TCP agent. But in general, users can create any agent or traffic sources in this way. Agents and traffic sources are in fact basic objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on).. But one shortcut is to look at the "ns-2/tcl/libs/ns-default.tcl" file. This file contains the default configurable parameter value settings for available network objects. Therefore, it works as a good indicator of what kind of network objects is available in NS and what are the configurable parameters.
* ***$ns* attach-agent *node agent*:** The attach-agent member function attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp. Similarly, each agent object has a member function attach-agent that attaches a traffic source object to itself.
* ***$ns* connect *agent1 agent2***: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them. This line establishes a network connection by setting the destination address to each others' network and port address pair.

Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is the following:

* ***$ns* at *time "string"*:** This member function of a Simulator object makes the scheduler (scheduler\_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, *$ns* at *0.1 "$cbr start"* will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data. In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.

After all network configuration, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by *$ns* run.

**Procedure:**

1. Study various parameters used in the NS2 script.

2. Write a program in NS2 and save it using “wireless.tcl”.

3. Run the script using command “ns wireless.tcl”

4. Observe the animation using the command:”nam wireless-out.nam”

**CONCLUSION:**

Signature of Faculty :