

1. NETWORK SIMULATOR-VERSION 2 (NS2)

1.1 INTRODUCTION

NS2 [Mark Greis tutorial] is an open source simulation tool available for researchers and students. It is discrete event-driven simulator i.e. each occurrence is processed as an event at different time intervals. It is invented in 1989, and has continuously gained tremendous importance. Now a day's NS2 has become the most widely used open source network simulator, and one of the most widely used network simulators.

NS-2 is very important tool and widely used by both the research community as well as by the people involved in the standardization protocols of IETF. The NS2 simulator covers a very large number of applications, of protocols, of network types, of network elements and of traffic models called as "simulated objects". For complete reference of ns2 in detail one must refer to "<http://www.isi.edu/nsnam/ns/>"

NS2 used in developing internet. It is the tool which is effectively used for exploring the Internet, for discovering properties of proposed protocols and for testing proposed solutions. It also used as a verification tool for proposed analytical models for networks. The NS2 simulator is open source and always free.

1.2 BACKGROUND OF NS2

1.2.1 NETWORK

It is a special arrangement of communicating devices/nodes which allows us to exchange the data/information. The connection between these nodes may be the wired/cabled media connection or wireless. Internet is the good example of network.

1.2.2 SIMULATION

Simulation is nothing but representing hardware efficiently in terms of software before implementing it. It helps finding bugs in the hardware deployment and also errors in the systems. The main application of this can be perceptive the behavior of the network.

1.2.3 SIMULATOR

Network Simulator is a hardware/software which is used to predict the actual behavior of the computer network before it is implemented. Typically, users can use the simulator to analysis their specific needs.

1.2.4 SIMULATION SCHEME

Simulation of the network is that the most complicated task. Most of the simulators are GUI only. And also most of them are discrete event simulation based only. Here a list of pending events with executable sequence is stored.

NS-2 simulator is the combination of two languages [8] i.e. an Object Oriented Simulator, written in C++, and an OTcl (an Object Oriented Extension of Tool Command Language) interpreter.

used at front end.

OTCL:

compile used at back end.

C++: Faster to

It's shown in fig. 3.1. Event in NS2 is an object in C++ hierarchy with a singular unique ID. It has a scheduled time and also the pointer to handle that event.

1.3 BASIC STRUCTURE OF NS2

Figure 3.1 depicts the architecture of NS2 in detail. As shown NS2 provides users with an executable command "ns" (usually object of Simulator class in ns2.35) which takes one input argument, the name of a Tcl simulation scripting file. As shown a TCL code is given input to the NS2 simulator. And output we get is in two forms a trace file and NAM file (we find out about these later).

Trace file is generated after the compilation of the input TCL code. It actually stores the complete information of network such as packet sending, receiving etc. it is very hard to analyze on our own as it contains huge data. NAM helps to visualize this.

It also depicts the linkage between C++ and OTCL. It is done through TclCL. The simulation objects of C++ and OTCL are connected/ linked as shown. They need to develop their own C++ classes and use an OTcl configuration interface to put together objects instantiated from these classes.

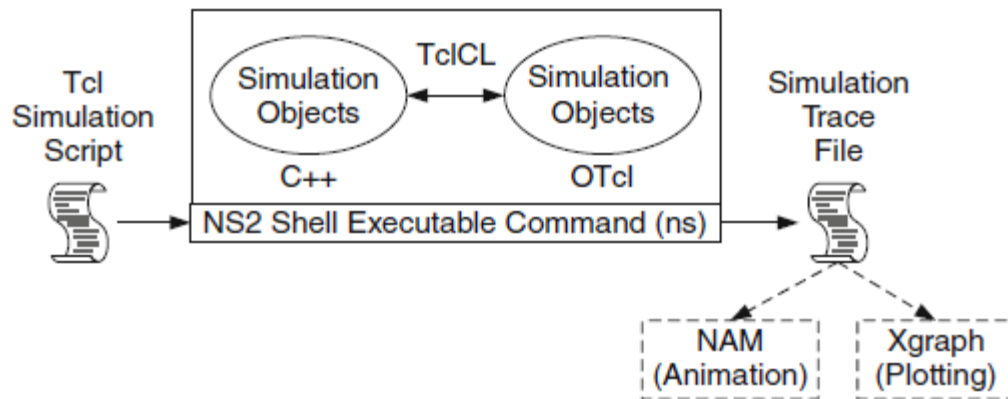


Figure 1.1: Structure of NS2.

1.4TCL PROGRAMMING

It is a language with a very simple syntax and it allows a very easy integration/ coordination with other languages. John Ousterhout was the man who invented this language. It has Faster development. Works and/or compatible with many platforms

Tcl Programming:

- “set” command is used to assigning a value to a variable. For example: “**set b 0**” assigns to the variable ‘b’ the value of ‘0’. This is equivalent to “b=0” in C.
- If we want use the value of declared variable use “\$” symbol. For example, if we want to assign to variable x the value that variable ‘a’ has, then we should write: “**set x \$a**”.
- The “expr” (expression) command is used for mathematical operation. For example, if we wish to assign to a variable x the sum of values of some variables a and b, we should write “**set x [expr \$a + \$b]**”.
- In Tcl the variables are not defined/typed. It can be a string or an integer depending on the value you assign to it. For example, If we write “**puts [expr 1/60]**”, then the result will be 0. To have the correct result, should type “**puts [expr 1.0/60.0]**”
- The sign “#” denotes a commented line that is not part of the program, so the tcl interpreter will not execute this line.
- To create a file in TCL, we have to give it a name, say “filename”. This is done with the command: “**set file1 [open filename w]**”. Here file1 is a pointer which points “filename”. ‘w’ indicates writing mode (we study it later).
- The command “**puts**” is used for printing an output.

- It has similar “if” structure in C++
- Execution of any UNIX command is done through “**exec**”. For example, the Xgraph command will be written as: “**exec xgraph data &**”. Here the “&” sign is used to have the command executed in the background).
- Tcl allows creating ‘**procedures**’ which is used to define new commands which are used in program. These return some value when they contain a "return" command. For ex. procedure “blue” is given as,

```
proc blue { par1 par2 ... } {
    global var1 var2
    <Commands>
    return $something }
```

The procedure receives some parameters as arguments that can be objects, files or variables as shown par1, par2 etc. The procedure is called by typing “**blue x y**”. On the other hand, if we wish the procedure to be able to affect directly variables external to it, we have to declare these variables as “**global**”.

1.5 FILE HANDLING IN TCL/OTCL

- In Tcl “open” command is used to open a channel with file to perform read or wrote operation as explained above.
- File can be opened in various modes [8] which allows type of operation is to be performed

These are

- ❖ Read Mode(r)
- ❖ Write Mode(w)
- ❖ Append Mode(a)
- ❖ Read & Write(r+)
- ❖ Write & Read(w+)

Example:

```
# Read Only Mode (r): “open exmaple1.txt r”
```

```
#for assigning open file handle to variable: “set file1 [open one.txt r]”
```

1.6 INSTALLING NS2

NS2 [8] is a free simulation tool, which can be obtained from <http://nsnam.isi.edu/>. It well suits on UNIX (or Linux). It is easily used in Linux because it is invented in Linux only. It is recommended that download ns all in one package.

1.6.1 Installation Steps of NS2 in Ubuntu

STEP1: Setting up the Prerequisites

Follow the steps discussed to install on Ubuntu 14.04 (NS2-2.35)

1. Download the latest NS2 software from <http://www.isi.edu/nsnam/ns/> and sure that it should be download into “/home/user_name/Documents” directory
2. Update Ubuntu with its latest components. For this open TERMINAL and run the fallowing commands [website 8]

a) sudo apt –get update

b) sudo apt –get dist-upgrade

c) sudo apt –get update

After this your Ubuntu is fully updated

3. Before installing NS, install some essential packages required by the NS software by typing fallowing commands

a) sudo apt –get install -essential autoconf automake

b) sudo apt –get install tcl8.5 –dev tk8.5 –dev

c) sudo apt –get install perl xgraph libxt –dev libx11 –dev libxmu –dev

d) sudo apt –get install gcc -4.4

The next step will be the extraction of NS package in a required folder

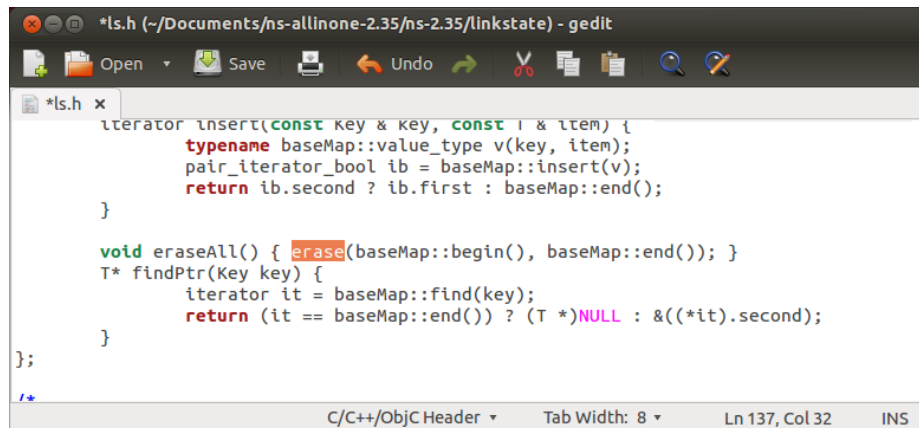
STEP2: Extract and install NS

1. Extract or Unzip or ‘**untar**’ it to “/home/user_name/Documents” folder by right clicking on downloaded NS 2.35 folder or using the fallowing commands from GUI terminal.

a) Cd /home/user_name/Documents

b) tar-xzvf ns-allinone-2.35.tar.gz

2. After extracting “ns-allinone-2.35” folder open up the file “/ns-allinone-2.35/ns-2.35/linkstate/ls.h” in an editor to make some changes in ls.h else it shows ERROR while installing NS. Once after opening ls.h go to line 137 and replace **erase** with **this->erase** and save it. This is shown in following figures

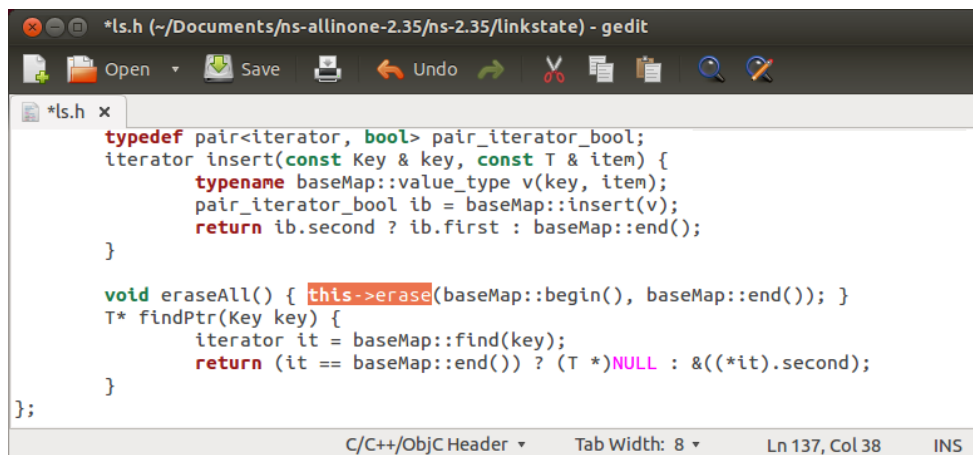


```
*ls.h x
iterator insert(const Key & key, const T & item) {
    typename baseMap::value_type v(key, item);
    pair_iterator_bool ib = baseMap::insert(v);
    return ib.second ? ib.first : baseMap::end();
}

void eraseAll() { erase(baseMap::begin(), baseMap::end()); }
T* findPtr(Key key) {
    iterator it = baseMap::find(key);
    return (it == baseMap::end()) ? (T *)NULL : &(*it).second;
}
};

C/C++/ObjC Header Tab Width: 8 Ln 137, Col 32 INS
```

Figure 1.2: Error at erase



```
*ls.h x
typedef pair<iterator, bool> pair_iterator_bool;
iterator insert(const Key & key, const T & item) {
    typename baseMap::value_type v(key, item);
    pair_iterator_bool ib = baseMap::insert(v);
    return ib.second ? ib.first : baseMap::end();
}

void eraseAll() { this->erase(baseMap::begin(), baseMap::end()); }
T* findPtr(Key key) {
    iterator it = baseMap::find(key);
    return (it == baseMap::end()) ? (T *)NULL : &(*it).second;
}
};

C/C++/ObjC Header Tab Width: 8 Ln 137, Col 38 INS
```

Figure 1.3: Changed to this ->erase

NOTE: if the above changes are not done ERROR occurs as follows

```
dhyans@dhyans: ~/Documents/ns-allinone-2.35
14 -I/home/dhyans/Documents/ns-allinone-2.35/include -I/home/dhyans/Documents/ns
-allinone-2.35/include -I/home/dhyans/Documents/ns-allinone-2.35/include -I/usr/
include/pcap -I./tcp -I./sctp -I./common -I./link -I./queue -I./adc -I./apps -I.
/mac -I./mobile -I./trace -I./routing -I./tools -I./classifier -I./mcast -I./dif
fusion3/lib/main -I./diffusion3/lib -I./diffusion3/lib/nr -I./diffusion3/ns -I./
diffusion3/filter_core -I./asim/ -I./qs -I./diffserv -I./satellite -I./wpan -o l
inkstate/ls.o linkstate/ls.cc
In file included from linkstate/ls.cc:67:0:
linkstate/ls.h: In instantiation of 'void LsMap<Key, T>::eraseAll() [with Key =
int; T = LsIdSeq]':
linkstate/ls.cc:396:28:   required from here
linkstate/ls.h:137:58: error: 'erase' was not declared in this scope, and no dec
larations were found by argument-dependent lookup at the point of instantiation
[-fpermissive]
    void eraseAll() { erase(baseMap::begin(), baseMap::end()); }
                        ^
linkstate/ls.h:137:58: note: declarations in dependent base 'std::map<int, LsIdS
eq, std::less<int>, std::allocator<std::pair<const int, LsIdSeq> > >' are not fo
und by unqualified lookup
linkstate/ls.h:137:58: note: use 'this->erase' instead
make: *** [linkstate/ls.o] Error 1
Ns make failed!
See http://www.isi.edu/nsnam/ns/ns-problems.html for problems
dhyans@dhyans:~/Documents/ns-allinone-2.35$
```

Figure 1.4: Expected error

3. Now it is the time to install NS. Open up a terminal and move to the folder where NS is extracted i.e. “/home/user_name/Documents” and Run this command to start ns-2 installation and wait until installation is over (it takes up to 15 minutes)

a) **Cd /home/user_name/Documents/ns-allinone-2.35**

b) **Sudo ./install.**

```
dhyans@dhyans-UB: ~/Documents/ns-allinone-2.35
dhyans@dhyans-UB:~/Documents/ns-allinone-2.35$ ./install
=====
* Testing for Cygwin environment
=====
Cygwin not detected, proceeding with regular install.
=====
* Testing for FreeBSD environment
=====
FreeBSD not detected
=====
* Build XGraph-12.2
=====
configure: error: expected an absolute directory name for --prefix: ..
make all-am
make[1]: Entering directory `/home/dhyans/Documents/ns-allinone-2.35/xgraph-12.2'
gcc -DHAVE_CONFIG_H -I.      -g      -MT xgraph.o -MD -MP -MF .deps/xgraph.Tpo -
c -o xgraph.o xgraph.c
xgraph.c: In function 'Traverse':
xgraph.c:464:4: warning: format not a string literal and no format arguments [-W
format-security]
xgraph.c:476:4: warning: format not a string literal and no format arguments [-W
format-security]
```

Figure 1.5: Proper Installation

STEP3: Set the Environment Variables

1. This can be done by editing the “**.bashrc**” file. Open a new terminal and open the file using following [8] **sudo gedit .bashrc**

2. Add the following lines AT TIP of the file. Change ‘path_to’ to the path of where you have extracted the NS i.e. here **/home/user_name/Documents/**

LD_LIBRARY_PATH

OTCL_LIB=/path_to/ns-allinone-2.35/otcl-1.14/

NS2_LIB=/path_to/ns-allinone-2.35/lib/

USR_Local_LIB=/usr/local/lib/

export

LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:\$OTCL_LIB:\$NS2_LIB:\$USR_Local_LIB

TCL_LIBRARY

TCL_LIB=/path_to/ns-allinone-2.35/tcl8.5.10/library/

USR_LIB=/usr/lib/

export TCL_LIBRARY=\$TCL_LIBRARY:\$TCL_LIB:\$USR_LIB

PATH

XGRAPH=/path_to/ns-allinone-2.35/xgraph-12.2:/path_to/ns-allinone-2.35/bin:/path_to/ns-allinone-2.35/tcl8.5.10/unix:/path_to/ns-allinone-2.35/tk8.5.10/unix/

NS=/path_to/ns-allinone-2.35/ns-2.35/

NAM=/path_to/ns-allinone-2.35/nam-1.15/

export PATH=\$PATH:\$XGRAPH:\$NS:\$NAM

3. Once done, save the file and restart the system. Alternatively just reload the .bashrc as

Source ~/.bashrc

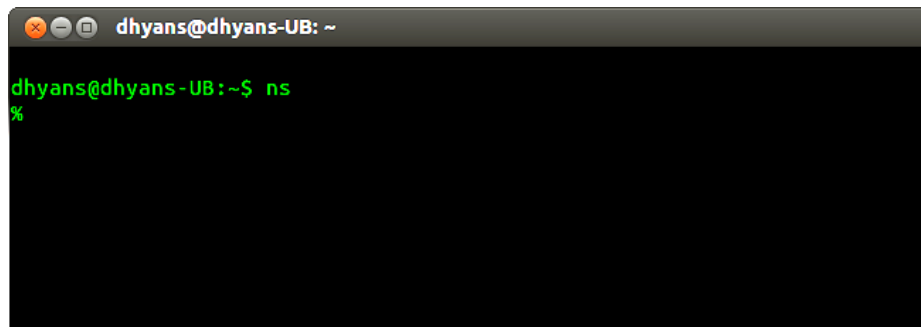
STEP4: INSTALLATION VALIDATION

1. Open a terminal and move to the directory `/home/user_name/Documents/ns-allinone-2.35/ns2.35/` and run:

`./validate`

2. Restart the system open the terminal and type “ns”

If % sign appears it means ns-2 is installed in system.

A terminal window with a dark background and light green text. The window title is 'dhyans@dhyans-UB: ~'. The prompt is 'dhyans@dhyans-UB:~\$' and the command 'ns' has been entered. The output is a '%' symbol, indicating successful installation.

```
dhyans@dhyans-UB: ~  
dhyans@dhyans-UB:~$ ns  
%
```

Figure 1.6: Successful Installation

1.7 NS-2 Simulator Preliminaries

In NS2 the following steps are very essential. They are

- Initialization and termination of NS-2 simulator,
- Nodes and links establishment
- Define agents& applications,
- The NAM,
- Tracing, and
- Random Variables.

3.7.1 VISUALISATION USING NAM:

The syntax used to open nam file is “**nam <nam-file>**”.

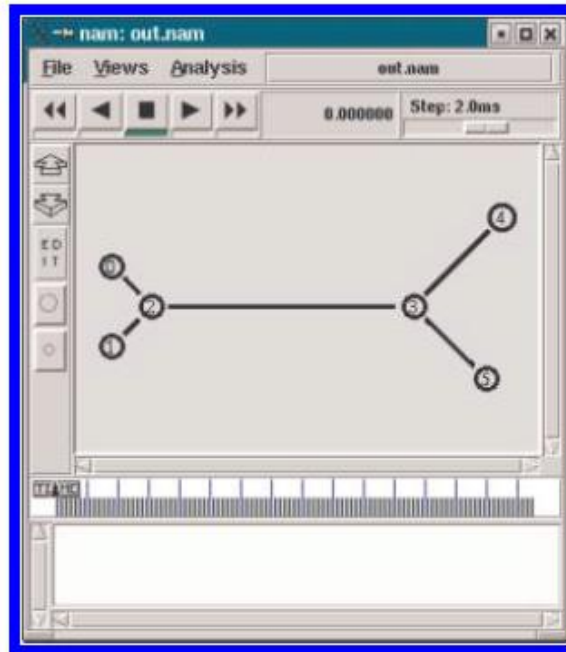


Figure 1.7: NAM Graphic Interface.

1.7.2 TRACING:

The file written by an application to store coverage information or entire network information is called as Trace File.

Syntax: **set exprv [open exout.tr w]**

Where “exout.tr” is the required trace file in which the information is stored, eff is the pointer used to point the trace file “exout.tr”. and ‘w’ is for writing mode.

```

+ 0.1 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.1 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.114 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.114 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.114 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.240667 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.240667 3 5 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.240667 3 5 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.286667 3 5 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.9 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
- 0.9 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
r 0.914 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
+ 0.914 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
- 0.914 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
+ 1 0 2 tcp 40 ----- 1 0.0 4.0 0 2
- 1 0 2 tcp 40 ----- 1 0.0 4.0 0 2
r 1.01016 0 2 tcp 40 ----- 1 0.0 4.0 0 2
+ 1.01016 2 3 tcp 40 ----- 1 0.0 4.0 0 2
- 1.01016 2 3 tcp 40 ----- 1 0.0 4.0 0 2
r 1.040667 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
+ 1.040667 3 5 cbr 1000 ----- 2 1.0 5.0 1 1
- 1.040667 3 5 cbr 1000 ----- 2 1.0 5.0 1 1
r 1.086667 3 5 cbr 1000 ----- 2 1.0 5.0 1 1
r 1.111227 2 3 tcp 40 ----- 1 0.0 4.0 0 2
+ 1.111227 3 4 tcp 40 ----- 1 0.0 4.0 0 2
- 1.111227 3 4 tcp 40 ----- 1 0.0 4.0 0 2
r 1.151867 3 4 tcp 40 ----- 1 0.0 4.0 0 2
+ 1.151867 4 3 ack 40 ----- 1 4.0 0.0 0 3
- 1.151867 4 3 ack 40 ----- 1 4.0 0.0 0 3
r 1.192507 4 3 ack 40 ----- 1 4.0 0.0 0 3
+ 1.192507 3 2 ack 40 ----- 1 4.0 0.0 0 3
- 1.192507 3 2 ack 40 ----- 1 4.0 0.0 0 3
r 1.293573 3 2 ack 40 ----- 1 4.0 0.0 0 3
+ 1.293573 2 0 ack 40 ----- 1 4.0 0.0 0 3
- 1.293573 2 0 ack 40 ----- 1 4.0 0.0 0 3
r 1.303733 2 0 ack 40 ----- 1 4.0 0.0 0 3
+ 1.303733 0 2 tcp 592 ----- 1 0.0 4.0 1 4
- 1.303733 0 2 tcp 592 ----- 1 0.0 4.0 1 4

```

Figure 1.8: Example Trace File

The format for the trace is given below

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

Figure 1.9 : Trace Format

It has 12 fields explained below.

1. Event/Type Identifier

+: Enque,

- : Deque

R: Reception

D: Drop

C: Collision

2. TIME: At which event started

3-4. Source and Destination Id

5. Packet Name

6. Packet Size

7. Flags

8. Flow Id

9-10. Source and Destination Address

11. Sequence Number

12. Packet Id

1.8 CONCLUSIONS

Network simulator (NS2) is the widely used simulation tool. It has the very wide range of applications. It also supports the analysis of files indifferent types like NAM and Trace. The entire information of network is stored in trace file which is further analyzed.

2. Study of different network Devices

Aim: Study of following Network Devices in Detail

- Repeater
- Hub
- Switch
- Bridge
- Router
- Gate Way

Apparatus (Software): No software or hardware needed.

Procedure: Following should be done to understand this practical.

1. **Repeater:** Functioning at Physical Layer. A **repeater** is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot be use to connect for more than two devices

2. **Hub:** An **Ethernet hub, active hub, network hub, repeater hub, hub** or **concentrator** is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

3. **Switch:** A **network switch** or **switching hub** is a computer networking device that connects network segments. The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

4. **Bridge:** A **network bridge** connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term *bridge* formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports. *Switch* or *Layer 2 switch* is often used interchangeably with *bridge*. Bridges can analyze incoming data packets to determine if the bridge is able to send the given packet to another segment of the network.

5. **Router:** A **router** is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

6. **Gate Way:** In a communications network, a network node equipped for interfacing with

another network that uses different protocols.

- A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks.
- A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

EXAMPLE: RING TOPOLOGY

```
#This
Program
will
create a
ring
topology
using less
number of
statements
in TCL
Language

    set ns [new Simulator]
    $ns rtproto DV

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

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

    #Creating Nodes
    for {set i 0} {$i<7} {incr i} {
        set n($i) [$ns node]
    }

    #Creating Links
    for {set i 0} {$i<7} {incr i} {
        $ns duplex-link $n($i) $n([expr ($i+1)%7]) 512Kb 5ms DropTail
    }

    $ns duplex-link-op $n(0) $n(1) queuePos 1
    $ns duplex-link-op $n(0) $n(6) queuePos 1

    #Creating UDP agent and attaching to node 0
    set udp0 [new Agent/UDP]
    $ns attach-agent $n(0) $udp0
```

```

#Creating Null agent and attaching to node 3
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0

#Creating a CBR agent and attaching it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 1024
$cbr0 set interval_ 0.01
$cbr0 attach-agent $udp0

$ns rtmodel-at 0.4 down $n(2) $n(3)
$ns rtmodel-at 1.0 up $n(2) $n(3)

$ns at 0.01 "$cbr0 start"
$ns at 1.5 "$cbr0 stop"

$ns at 2.0 "finish"
$ns run

```

EXAMPLE: STAR TOPOLOGY

```

#This
program
will
create a
Star
Topology
using for
loop in
tcl in
order to
use less
statements

```

```

set ns [new Simulator]

$ns color 1 blue
$ns color 2 red

$ns rtproto DV

set nf [open out.nam w]

```



```

$ns namtrace-all $nf

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

#creating Nodes
for {set i 0} {$i<7} {incr i} {
    set n($i) [$ns node]
}

#Creating Links
for {set i 1} {$i<7} {incr i} {
    $ns duplex-link $n(0) $n($i) 512Kb 10ms SFQ
}

#Orienting The nodes
$ns duplex-link-op $n(0) $n(1) orient left-up
$ns duplex-link-op $n(0) $n(2) orient right-up
$ns duplex-link-op $n(0) $n(3) orient right
$ns duplex-link-op $n(0) $n(4) orient right-down
$ns duplex-link-op $n(0) $n(5) orient left-down
$ns duplex-link-op $n(0) $n(6) orient left

#TCP_Config
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n(1) $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n(4) $sink0

$ns connect $tcp0 $sink0

#UDP_Config
set udp0 [new Agent/UDP]
$udp0 set class_ 2
$ns attach-agent $n(2) $udp0

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0

```

```

$ns connect $udp0 $null0

#CBR Config
set cbr0 [new Application/Traffic/CBR]
$cbr0 set rate_ 256Kb
$cbr0 attach-agent $udp0

#FTP Config
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

#Scheduling Events
$ns rtmodel-at 0.5 down $n(0) $n(5)
$ns rtmodel-at 0.9 up $n(0) $n(5)

$ns rtmodel-at 0.7 down $n(0) $n(4)
$ns rtmodel-at 1.2 up $n(0) $n(4)

$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"

$ns at 0.2 "$cbr0 start"
$ns at 1.3 "$cbr0 stop"

$ns at 2.0 "finish"
$ns run

```

EXAMPLE: GENERAL

Creating New Simulator

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

Setting up the traces

```
set f [open outEx1.tr w]
```

```
set nf [open outEx1.nam w]
```

```
$ns namtrace-all $nf
```

```
$ns trace-all $f
```

```
proc finish {} {
```

```
global ns nf f

$ns flush-trace

puts "Simulation completed."

close $nf

close $f

exit 0

}

#

#Create Nodes

#

set n0 [$ns node]

    puts "n0: [$n0 id]"

set n1 [$ns node]

    puts "n1: [$n1 id]"

set n2 [$ns node]

    puts "n2: [$n2 id]"

set n3 [$ns node]

    puts "n3: [$n3 id]"

set n4 [$ns node]

    puts "n4: [$n4 id]"

#

#Setup Connections

#
```

\$ns duplex-link \$n0 \$n2 100Mb 5ms DropTail

\$ns duplex-link \$n2 \$n4 54Mb 10ms DropTail

\$ns duplex-link \$n1 \$n2 100Mb 5ms DropTail

\$ns duplex-link \$n2 \$n3 54Mb 10ms DropTail

\$ns queue-limit \$n2 \$n3 40

\$ns simplex-link \$n3 \$n4 10Mb 15ms DropTail

\$ns simplex-link \$n4 \$n3 10Mb 15ms DropTail

#

#Set up Transportation Level Connections

#

set tcp0 [new Agent/TCP]

\$ns attach-agent \$n1 \$tcp0

set udp1 [new Agent/UDP]

\$udp1 set dst_addr_ Unicast

\$udp1 set fid_ 1

\$ns attach-agent \$n0 \$udp1

set null0 [new Agent/Null]

\$ns attach-agent \$n3 \$null0

set sink0 [new Agent/TCPSink]

\$ns attach-agent \$n4 \$sink0

#

#Setup traffic sources

#

```

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set rate_ 2Mb

$cbr0 set packetSize_ 1000

$cbr0 attach-agent $udp1

$ns connect $udp1 $null0

$udp1 set fid_ 0

$ns connect $tcp0 $sink0

$tcp0 set fid_ 1

#

#Start up the sources

#

$ns at 0.05 "$ftp0 start"

$ns at 0.1 "$cbr0 start"

$ns at 60.0 "$ftp0 stop"

$ns at 60.5 "$cbr0 stop"

$ns at 61.0 "finish"

$ns run

```

Example :

The network consists of five nodes n0 to n4. In this scenario, node n0 sends constant bit-rate (CBR) traffic to node n3, and node n1 transfers data to node n4 using a file transfer protocol (FTP). These two carried traffic sources are carried by transport layer protocols User Datagram

Protocol (UDP) and Transmission Control Protocol (TCP), respectively. In NS2, the transmitting object of these two protocols are a UDP agent and a TCP agent, while the receivers are a Null agent and a TCP sink agent, respectively.

```
# Creating New Simulator
set ns [new Simulator]

# Setting up the traces
set f [open outEx1.tr w]
set nf [open outEx1.nam w]
$ns namtrace-all $nf
$ns trace-all $f
proc finish {} {

    global ns nf f
    $ns flush-trace
    puts "Simulation completed."
    close $nf
    close $f
    exit 0
}

#
#Create Nodes
#

set n0 [$ns node]
    puts "n0: [$n0 id]"
set n1 [$ns node]
    puts "n1: [$n1 id]"
set n2 [$ns node]
```

```
        puts "n2: [$n2 id]"
set n3 [$ns node]
        puts "n3: [$n3 id]"
set n4 [$ns node]
        puts "n4: [$n4 id]"

#
#Setup Connections
#
$ns duplex-link $n0 $n2 100Mb 5ms DropTail
$ns duplex-link $n2 $n4 54Mb 10ms DropTail
$ns duplex-link $n1 $n2 100Mb 5ms DropTail
$ns duplex-link $n2 $n3 54Mb 10ms DropTail
$ns queue-limit $n2 $n3 40
$ns simplex-link $n3 $n4 10Mb 15ms DropTail
$ns simplex-link $n4 $n3 10Mb 15ms DropTail
#

#Set up Transportation Level Connections
#
set tcp0 [new Agent/TCP]
$ns attach-agent $n1 $tcp0
set udp1 [new Agent/UDP]
$udp1 set dst_addr_ Unicast
$udp1 set fid_ 1
$ns attach-agent $n0 $udp1
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n4 $sink0
#
#Setup traffic sources
```

```

#
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set rate_ 2Mb

$cbr0 set packetSize_ 1000
$cbr0 attach-agent $udp1
$ns connect $udp1 $null0
$udp1 set fid_ 0
$ns connect $tcp0 $sink0
$tcp0 set fid_ 1
#
#Start up the sources
#
$ns at 0.05 "$ftp0 start"
$ns at 0.1 "$cbr0 start"
$ns at 60.0 "$ftp0 stop"
$ns at 60.5 "$cbr0 stop"
$ns at 61.0 "finish"
$ns run

```

To run nam execute following

```

proc finish {} {

    global ns nf f

    $ns flush-trace

    close $nf

    close $f

```



```

        exec nam outEx1.nam &

        exit 0

}

```

EXAMPLE : GENERAL

```

set ns [new Simulator]\”Defining Simulator”\
settracefd [open Smart_Grid_Network_Topology.tr w]\”Intializing trace file”\
settopo [new Topography]\”Defining Topography”\
$topoload_flatgrid $val(x) $val(y)

set windowVsTime2 [open win.tr w]

setnamtrace [open Smart_Grid_Network_Topology.nam w]

$ns trace-all $tracefd

$ ns namtrace-all-wireless $namtrace $val(x) $val(y)

create-god (1)\”Creating GOD”\
set god_ [God instance]

$ns at 0.5000 “$node_(0) setdest 100 400 0.15667896543”

$node_(0) set Z_ 0.0

$ node_(0) set Y_ 128

$node_(0) set X_ 142

$ns at 0.5000 “$node_(1) setdest 150 450 0.15667896543”

$node_(1) set Z_ 0.0

$ node_(1) set Y_ 128

$node_(1) set X_ 44

$ns at 0.5000 “$node_(2) setdest 140 400 0.15667896543”

$node_(2) set Z_ 0.0

$ node_(2) set Y_ 193

```

```
$node_(2) set X_ 144
$ns at 0.5000 "$node_(3) setdest 160 350 0.15667896543"
$node_(3) set Z_ 0.0
$ node_(3) set Y_ 199
$node_(3) set X_ 55
$ns at 0.5000 "$node_(4) setdest 120 270 0.15667896543"
$ node_(4) set Z_ 0.0
$node_(4) set Y_ 243
$ node_(4) set X_ 107
if { "$val(traffic)" == "tcp" } {
setpktTypetcp
} else {
setpktTypeexp
}
$ns duplex-link $node_(0) $node_(2) 2Mb 40ms RED
$ ns duplex-link $node_(2) $node_(4) 2Mb 40ms RED
$ns duplex-link $node_(4) $node_(3) 2Mb 40ms RED
$ ns duplex-link $node_(3) $node_(1) 2Mb 40ms RED
$ns duplex-link-op $node_(0) $node_(2) color "lavender"
$ ns duplex-link-op $node_(2) $node_(4) color "lavender"
$ns duplex-link-op $node_(4) $node_(3) color "lavender"
$ ns duplex-link-op $node_(3) $node_(1) color "lavender"
settcp [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
$ ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
$ ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
$ns at 5.0 "finish"
puts "CBR packet size = [$cbr set packet_size_]"
puts"CBR interval = [$cbr set interval_]"
$ns run

```

Random nodes:

```

*****Random Topology Creation*****#

#Random Location for a two nodes

for {set i 0} {$i < 2} {incr i} {

$node_($i) set X_ [expr rand()*500]

$node_($i) set Y_ [expr rand()*400]

$node_($i) set Z_ 0

}

```

PYTHON CODE:

Create a simulator object

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

Define different colors

for data flows (for NAM)

```
$ns color 1 Blue
```

```
$ns color 2 Red
```

Open the NAM trace file

```
set nf [open out.nam w]
```

```
$ns namtrace-all $nf
```

Define a 'finish' procedure

```
proc finish {} {
```

```
    global ns nf
```

```
    $ns flush-trace
```

```
    # Close the NAM trace file
```

```
    close $nf
```

```
# Execute NAM on the trace file

exec nam out.nam &

exit 0

}
```

```
# 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
```

```
# Monitor the queue for link (n2-n3). (for NAM)
```

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

```
# 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"
```

```
# Detach tcp and sink agents
```

```
# (not really necessary)
```

```
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
```

```
# Call the finish procedure after

# 5 seconds of simulation time

$ns at 5.0 "finish"

# Print CBR packet size and interval

puts "CBR packet size = [$cbr set packet_size_]"

puts "CBR interval = [$cbr set interval_]"

# Run the simulation

$ns run
```


NS2

NS2 can not be act as the emulator.

NS3 scripts can not run in NS2 environment

NS2 is written with the help of TCL and C++.

C++ recompilation takes more time more than TCL so most of the scripts are written using TCL

Simulation script is not possible with NS2

Only TCL can be used as the scripting language

t The header part of the NS2 includes all the information of header parts in the specified protocol

NS2 never reuse or re allocate the memory until it get terminated.

The packet of ns2 has headers and data for payload.

.tr-> files used for trace parameters

.nam -> files used for Network Animation

.xg -> files used for graph

Python visualizer , Network Animator visualization is available

Memory allocation is not good as NS3

Unnecessary parameters can not be prevented.

Total Computation time is high when compared to NS3

Step-5 Difference between ns2 and ns3

NS3

NS3 can be act as the emulator that it can connect to the real world

Some of the NS2 models can be imported to NS3

NS3 is written using C++

Compilation time is not a matter

A Simulation script can be written in ns3

Python is available for the scripting language

Information needed to send through the packet can be added at the header ,trailer, buffer ,etc

NS3 frees the memory that used to store the packets

The packet of ns3 consist of single buffer and small tags

.tr-> files used for trace analysis

.XML->files are used for network Animation

.csv-> files used for gnu plot

Nam animator is available for visualization.

Memory allocation is good

System prevents unnecessary parameters to be stored.

Total computation is less when compared to NS2

This message is shown once a day. To disable it please create the
/home/ganesh416/.hushlogin file.

```
ganesh416@DESKTOP-GT7OR0A:~$ pwd
/home/ganesh416
```

```
ganesh416@DESKTOP-GT7OR0A:~$ ls
```

```
ganesh416@DESKTOP-GT7OR0A:~$ cd ..
```

```
ganesh416@DESKTOP-GT7OR0A:/home$ cd ..
```

```
ganesh416@DESKTOP-GT7OR0A:/home$ cd /mnt/c/Users
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users$ cd GANESH\ BOGA/
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA$ cd Desktop/cnlab/
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ gedit example.tcl
Unable to init server: Could not connect: Connection refused
```

```
(gedit:85): Gtk-WARNING **: 11:22:30.403: cannot open display:
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ export DISPLAY=0:0
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ gedit example.tcl
```

```
Unable to init server: Could not connect: Connection refused
```

```
(gedit:88): Gtk-WARNING **: 11:22:51.404: cannot open display: 0:0
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ export DISPLAY=0:0
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ gedit example.tcl
```

```
Unable to init server: Could not connect: Connection refused
```

```
(gedit:92): Gtk-WARNING **: 11:23:04.020: cannot open display: 0:0
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ export DISPLAY=0:0
```

```
ganesh416@DESKTOP-GT7OR0A:/mnt/c/Users/GANESH BOGA/Desktop/cnlab$ gedit example.tcl
```