

Jaypee Institute of Information Technology, Noida

Lab Manual

Telecommunication Networks Lab (15B17EC671)



Department of Electronics and Communication Engineering

LIST OF EXPERIMENTS**TELECOMMUNICATION NETWORKS LAB:- 15B17EC671**

1. (a) To learn about network simulator, and use NS2 for conducting network simulation including LINUX commands.
(b) To learn installing NS2 in Fedora.
2. (a) Introduction to OSI, TCP & UDP.
(b) To set up a network with two nodes; link them with duplex link, 10ms propagation delay, 1Mbps rate and DropTail procedure. Use Agent UDP with CBR traffic source.
3. To set up a network with two nodes; link them with duplex link, 10ms propagation delay, 1Mbps rate and DropTail procedure. Use FTP over Agent TCP.
4. To implement wired LAN connection in NS2.
5. To create a mobile ad-hoc network with 3 nodes in 500*400 topography with following initial positions and movements:
Node 0 (5, 5) Node 1 (490,285) Node 2 (150,240) At t = 10, 0
moves towards (250,250) at 3m/sec.

At t =15, 10 moves towards (45,285) at 5m/sec.

At t =110, 100 moves towards (480,300) at 5m/sec.
6. To create a Heterogeneous Network (wired cum wireless network).
7. To interpret data trace file (.tr) of Wired, Wireless and LAN Networks.
8. Throughput calculation for TCP or UDP in Wired network.
9. To create a network with 4 nodes 0-2, 1-2, 2-3 with TCP from 0-3 and UDP from 1-3. Apply an error model on link 2-3 with error rate 0.2 and uniform distribution. Apply queue monitor on 2-3 link and interpret any five lines of qm.out file.

10. To create a network with 5 nodes, and apply uniform, exponential and constant error model with error rate 1% on 3 different links.
11. Simulate an Ethernet LAN using 'n' nodes and set multiple traffic nodes and determine the collision across different nodes.
12. Write a program for error detecting code using CRC-CCITT (16bit).
13. To simulate a link failure and to observe distance vector routing protocol.
14. Write a program for distance vector algorithm to find suitable path for transmission.
15. Write a program for congestion control using Leaky Bucket algorithm.

Virtual Lab Experiments:

- 1) Simulating a Wireless Sensor Network
- 2) Simulating a Wi-Fi Network

Details of Virtual Lab Used:

Advanced Network Technologies Virtual Lab (IIT Kharagpur)

The vlab link used is:

<http://vlabs.iitkgp.ernet.in/ant/>

APPLICATIONS OF THIS LAB

1. Real-Time Scheduling:- This lab implements a soft real-time scheduler which ties event execution within the simulator to real time.

2. Educational uses:

- (i) General information about using ns/nam for networking education
- (ii) Web index of educational scripts.
- (iii) Research using ns

3. Other applications:

- (i) Research using NAM (Network Animator)
- (ii) Topology Generation for large simulations
- (iii) Scenario generation in ns.

LIMITATIONS AND PRECAUTIONS TO BE TAKEN IN THIS LAB

1. Real system are too complex to model. i.e. they have complicated structure.
2. The simulator virtual time should closely track real-time. If the simulator becomes too slow to keep up with elapsing real time, a warning is continually produced if the skew exceeds a pre-specified constant "slope factor" (currently 10ms).
3. The Bugs are unreliable.
4. Modeling is a very complex and time-consuming task in NS-2, since it has no GUI and one needs to learn scripting language, queuing theory and modeling techniques. Also, of late, there have been complaints that results are not consistent (probably because of continuous changes in the code base) and that certain protocols have unacceptable bug
5. Precautions should be taken while programming because it is a very highly case sensitive language.

EXPERIMENT - 1

AIM: -(a) To learn about network simulator, and use NS2 for conducting network simulation including LINUX commands.

(b) To learn installing NS2 in Fedora.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34.

THEORY: -

History and UNIX

In order to understand the popularity of Linux, we need to travel back in time, about 30 years ago. Imagine computers as big as houses, even stadiums. While the sizes of those computers posed substantial problems, there was one thing that made this even worse: every computer had a different operating system. Software was always customized to serve a specific purpose, and software for one given system didn't run on another system. Being able to work with one system didn't automatically mean that you could work with another. It was difficult, both for the users and the system administrators. Computers were extremely expensive then, and sacrifices had to be made even after the original purchase just to get the users to understand how they worked. The total cost per unit of computing power was enormous.

Technologically the world was not quite that advanced, so they had to live with the size for another decade. In 1969, a team of developers in the Bell Labs laboratories started working on a solution for the software problem, to address these compatibility issues. They developed a new operating system, which was

1. Simple and elegant.
2. Written in the C programming language instead of in assembly code.
3. Able to recycle code.

The Bell Labs developers named their project "UNIX."

The code recycling features were very important. Until then, all commercially available computer systems were written in a code specifically developed for one system. UNIX on the other hand needed only a small piece of that special code, which is now commonly named the kernel. This kernel is the only piece of code that needs to be adapted for every specific system and forms the base of the UNIX system. The operating system and all other functions were built around this kernel and written in a higher programming language, C.

Linus and Linux

By the beginning of the 90s home PCs were finally powerful enough to run a full blown UNIX. Linus Torvalds, a young man studying computer science at the University of Helsinki, thought it would be a good idea to have some sort of freely available academic version of UNIX, and promptly started to code.

He started to ask questions, looking for answers and solutions that would help him get UNIX on his PC.

From the start, it was Linus' goal to have a free system that was completely compliant with the original UNIX. That is why he asked for POSIX standards, POSIX still being the standard for UNIX. In those days plug-and-play wasn't invented yet, but so many people were interested in having a UNIX system of their own, that this was only a small obstacle. New drivers became available for all kinds of new hardware, at a continuously rising speed. Almost as soon as a new piece of hardware became available, someone bought it and submitted it to the Linux test, as the system was gradually being called, releasing more free code for an ever wider range of hardware. These coders didn't stop at their PC's; every piece of hardware they could find was useful for Linux. Back then, those people were called "nerds" or "freaks", but it didn't matter to them, as long as the supported hardware list grew longer and longer. Thanks to these people, Linux is now not only ideal to run on new PC's, but is also the system of choice for old and exotic hardware that would be useless if Linux didn't exist. Two years after Linus' post, there were 12000 Linux users. The project, popular with hobbyists, grew steadily, all the while staying within the bounds of the POSIX standard. All the features of UNIX were added over the next couple of years, resulting in the mature

operating system Linux has become today. Linux is a full UNIX clone, fit for use on workstations as well as on middle-range and high-end servers. Today, a lot of the important players on the hard- and software market each have their team of Linux developers; at your local dealer's you can even buy pre-installed Linux systems with official support –even though there is still a lot of hard- and software that is not supported, too.

Logging in, activating the user interface and logging out – Introduction

In order to work on a Linux system directly, you will need to provide a user name and password. You always need to authenticate to the system. Most PC-based Linux systems have two basic modes for a system to run in: either quick and sober in text console mode, which looks like DOS with mouse, multitasking and multi-user features, or in graphical mode, which looks better but eats more system resources.

Graphical mode

This is the default nowadays on most desktop computers. You know you will connect to the system using graphical mode when you are first asked for your user name, and then, in a new window, to type your password. To log in, make sure the mouse pointer is in the login window, provide your user name and password to the system and click OK or press enter.

After entering your user name/password combination, it can take a little while before the graphical environment is started, depending on the CPU speed of your computer, on the software you use and on your personal settings.

To continue, you will need to open a *terminal window* or *xterm* for short (X being the name for the underlying software supporting the graphical environment). This program can be found in the Applications->Utilities, System Tools or Internet menu, depending on what window manager you are using. There might be icons that you can use as a shortcut to get an xterm window as well, and clicking the right mouse button on the desktop background will usually present you with a menu containing a terminal window application.

While browsing the menus, you will notice that a lot of things can be done without entering commands via the keyboard. For most users, the good old point-'n'-click method of dealing with the computer will do. But this guide is for future network and system administrators, who will need to meddle with the heart of the system. They need a stronger tool than a mouse to handle all the tasks they will face. This tool is the shell, and when in graphical mode, we activate our shell by opening a terminal window.

The terminal window is your control panel for the system. Almost everything that follows is done using this simple but powerful text tool. A terminal window should always show a command prompt when you open one. This terminal shows a standard prompt, which displays the user's login name, and the current working directory, represented by the twiddle (~):

Terminal window



Another common form for a prompt is this one:

```
[user@host dir]
```

In the above example, *user* will be your login name, *hosts* the name of the machine you are working on, and *dir* an indication of your current location in the file system. Later we will discuss prompts and their behavior in detail. For now, it suffices to know that prompts can display all kinds of information, but that they are not part of the commands you are giving to your system.

To disconnect from the system in graphical mode, you need to close all terminal windows and other applications. After that, hit the logout icon or find Log Out in the menu. Closing everything is not really necessary, and the system can do this for you, but session management might put all currently

open applications back on your screen when you connect again, which takes longer and is not always the desired effect. However, this behavior is configurable.

When you see the login screen again, asking to enter user name and password, logout was successful.

Absolute basics

The commands - Quickstart commands

Command and Meaning

ls Displays a list of files in the current working directory, like the **dir** command in DOS

cd directory change directories

passwd change the password for the current user

file filename display file type of file with name filename

cat textfile throws content of textfile on the screen

pwd display present working directory

exit or **logout** leave this session

man command read man pages on **command**

info command read Info pages on **command**

apropos string search the *whatis* database for strings

General remarks

You type these commands after the prompt, in a terminal window in graphical mode or in text mode, followed by **Enter**.

Commands can be issued by themselves, such as **ls**. A command behaves different when you specify an *option*, usually preceded with a dash (-), as in **ls -a**. The same option character may have a different meaning for another command. GNU programs take long options, preceded by two dashes (--), like **ls --all**. Some commands have no options.

The argument(s) to a command are specifications for the object(s) on which you want the command to take effect. An example is **ls /etc**, where the directory /etc is the argument to the **ls** command.

This indicates that you want to see the content of that directory, instead of the default, which would be the content of the current directory, obtained by just typing **ls** followed by **Enter**. Some commands require arguments, sometimes arguments are optional.

You can find out whether a command takes options and arguments, and which ones are valid, by checking the online help for that command.

In Linux, like in UNIX, directories are separated using forward slashes, like the ones used in web addresses (URLs). The symbols **.** and **..** have special meaning when directories are concerned.

File related commands:

Command	Description
ls	Lists files and directories.
ls -l	Lists files in 'long format' i.e containing useful info. About file such as who owns it, exact size of file.
ls -a	Lists all files and directories.
ls abc*	Lists all files and directories starting with abc.
mv f1 f2	Moves a file i.e gives it a different name or moves it into a different directory.
cp f1 f2	Copy file1 to file2.
rm f1	Removes a file.
rm -i	Ask for a confirmation before deleting anything.
diff f1 f2	Compares files and shows where they differ.
wc f1	Gives no of lines, words and characters in a file.
cat f1	To view a file.
cat > f1	To create a file.
cat f1 f2 > f3	Concatenates file1 and file2 to file3.
chmod options	To change the read, write, and execute permissions of files e.g. chmod

f1	go-rwx. Symbols used: u->user, g->group, o->others, a->all, r->read, w->write, x->execute, '+' -> add permissions, '-' ->take away the permissions.
gzip f1	Produces a compressed a file with .gz extension.
gunzip f1	Decompress a file compressed with gzip.
lpr	Prints a file.
sort options f1	Sorts a file in the order specified in the options.
head-n f1	Displays first 10 lines of file f1 unless specified using 'n'.
tail f1 -n no	Displays the last 10 lines of file unless specified using 'no'.
more f1	Shows first part of file just as such as will fit on screen.
less f1	It has the extended capability of allowing both forward and backward navigation through the file.
grep string f1	Prints all lines of file that contain the string.
grep-v string f1	Prints all lines of the file except those that contain the string.
tar-cvf f.tar f1 f2	To combine multiple files into a single file.
tar-xvf f.tar	To separate an archive created by tar into separate files.
ln f1 link	Creates a hard link forfile f1 .
ln-s f1 link	Creates a symbolic link for file f1 .
tee f1	Place the data flowing through a pipe, in file f1.

Directories related commands:

Command	Description
mkdir dirname	Makes a new directory.
pwd	Prints the current working directory.

rm-r/rmdir	Removes an empty directory.
cd	Changes to home directory.
cd ~	Changes to home directory.
cd dirname	Changes to named directory.
cd ..	Changes to parent directory.
rename from to file	Renames the specified files by replacing the first occurrence of from in their name by to.

Process related commands:

Command	Description
ps	Process status, shows a list of current processes in the system.
ps-aef	To see all the process.
ps-u username	To see the processes belonging to a particular user.
kill pid	For killing a process having the given pid.(process id)
./ prog.exe.	To run a executable file and hence create a process.
./ prog.exe.&	The '&' symbol is used to put the process in background.
sleep time(sec)	Creates a process that pauses (sleeps) , terminal would just hang, and you would get the prompt back after time specified.
fg	To put a process in foreground.
nohup	Prefixing nohup to the command, ensures that it would not be killed once the user logs out of the system.
jobs	To view the current stopped and background processes.
Cntrl+Z	Cause the current command running to be stopped.
Cntrl+C	To terminate the current running process.
bg	Causes the stopped process to be in background.

Login related commands:

Command	Description
passwd	Lets you change the password.
ftp hostname	For file transfer between one system and other.
get <absolute path on linux server>	To get a file present on the LINUX machine to your local machine.
put <absolute path on local machine>	To put a file from local machine to the remote machine.

System related commands:

Commands	Description
date	Prints the current date and time of the system.
who	Get details about the total number of users currently logged on.
whoami	Returns your username.
time<.prog exe.>	To measure the time statistics for a process.
uname	To get the system details.
top	To view the system usage at any given point of time.
vmstat	Gives the virtual memory usage at any point of time.
which command	To find the absolute path of some command.
find	To find the path of a file.
ff filename	Finds files anywhere on the system.

man	To search in system manuals for details of a specific command.
info	To read info documents,gives detailed info. of commands.

C program related commands:

Command	Description
indent filename	To indent files containing C programs.
gcc abc.c -o abc	Compile the file abc.c and create the executable abc.out.

Shell commands and variables:

Command	Description
Echo variable	Prints the value of its arguments. To print the value of a variable, we must prefix the variable's name with \$ symbol.
Variable	Description
SHELL	Name of shell.
HOME	Path (or location) of the home directory.
PATH	The directories that are to be searched, when path of an executable file is not given.

Few other important keys:

Command	Description
Cntrl+d	To close a session.
q	To get out of some editors such as info and to go back to prompt.

Q. Write a program to calculate factorial of 5.**Background on NS2:**

NS began development in 1989 as a variant of REAL network simulator. By 1995, NS had gained support from DARPA, the VINT project at LBL, Xerox PARC, UCB and USC/ISI. NS is now developed in collaboration between number of different researchers and institution, including SAMAN (supported by DARPA), CONSER (through the NSF), and ICIR (former ACIRI). Long-running contributions have also come from Sun Microsystems and the UCB Daedalus and Carnegie Mellon Monarch projects, cited by the NS homepage for wireless code additions. NS is based on two languages, an object oriented simulator written in C++ and an Otcl interpreter, used to execute users command scripts. NS have a rich library network and protocol objects. There are two class hierarchies: the compiled c++ hierarchy and the interpreted Otcl one, with one to one correspondence between them. NS-2 is a discrete event simulator, which means that events (e.g. packets to send, timeouts, etc) are scheduled in a global event queue according to their time of execution. When a simulation is run, the simulator removes events from the head of the queue, moves the simulator time to that of the currently removed event and executes it. When done, it continues to the next event and so forth.

Each simulation is defined by a scenario that contains a number of predefined events that define the scenario. NS-2 scenarios are implemented in TCL scripts that contain the command to initialize the simulator and to create the nodes and their configuration. Each simulation run generates a trace file containing all the data packets that are sent between the nodes during the course of the scenario. By analyzing this file it is possible to determine the performance effect of parameter variations, different routing protocols and more. A simulation can be very useful because it is possible to scale the networks easily and therefore to eliminate the need for time consuming and costly real world experiments. While the simulator is a powerful tool, it is important to remember that the ability to do predictions about the performance in the real world is dependent on the accuracy of the models in the simulator.

NS-2 contains the following components:

- **Tcl:** Creating Layouts.
- **Otcl:** Network Protocols design
- **C++:** for back end logic
- **TclCL:** linking C++ and Tcl code.
- **NAM (Network Animator):** to see results or simulation environment.
- **X-graph:** For analysis (optional).

NS-2 can be easily extended by 2 ways, modifying the OTcl or C++ code. In order to add a new component in ns2, modification on C++ code is the most efficient way to do it, because the C++ code represent the core of the simulator and by it we can able to modify and access every class variable or function or derive every kind of object in the c ++ hierarchy. Before new component will be added in to NS2, there are some guidelines. NS2 support two distinct types of monitoring traces and monitors. A trace use to record each individual packet as it arrives, departs or is dropped at a link or queue. Tracing in NS2 could be trace packets on all links or just for specific event by using a simple OTcl code. Since NS2 is a packet level simulator the trace is tracing packet in all link during simulation. Packet traces describe packets as they travel through the network. It can trace transmission of the packet on a link, the receipt of the packet and packet drops as well. The NS2 contains various trace formats for suitable use in different simulation such as wireless trace format, Ad hoc On Demand Distance Vector (AODV), routing algorithm trace format, Dynamic Destination- Sequenced Distance Vector Routing (DSDV) routing algorithm trace format and others. The network simulator uses two languages because simulator has two different kinds of things it needs to do. On one hand detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers and implement algorithms that run over large data sets.

For these tasks run-time speed is important and turn –around time (run simulation, find bug, Fix bug, Recompile, Re-run) is less important.

The simulator is initialized using the TCL interface. It is in this file that a new simulation object is created and several tcl commands relating to the network formation and its properties are added. When a new simulation object is created, the initialization procedure performs the following tasks:

- ☐ initializes the packet formats
- ☐ create a scheduler
- ☐ create a null agent

Packets may be handed to NS Objects at scheduled points in time since a node is an event handler and a tasks performed over a packet is an event. Tasks over a packet should be the only type of event scheduled on a node. The initialization of the packet formats will set up the field offsets used for the entire simulation. The scheduler is an event driven time management mechanism, which launches the earliest scheduled event, executing it to completion and returning back to execute the next event. An event comprises of the time at which it need to be launched along with a handler function. One property of the scheduler to be noted here is when two or more events occur at the same time (meaning, they have to be executed at the same time). Under these circumstances it would decide the precedence based on the order in which they have been scheduled.

NS2 Architecture:

As shown in the simplified user's view of figure below. NS is an Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries and network set-up (plumbing) module libraries.

The event scheduler in NS-2 performs the following tasks:

- ☐ Organizes the simulation timer.
- ☐ Fires events in the event queue.
- ☐ Invokes network components in the simulation.

Depending on the user's purpose for an OTcl simulation script, Simulation Results are stored as trace files which can be loaded for analysis by an external application:

1. A NAM trace file (file. nam) for use with the Network Animator Tool
2. A Trace file (file.tr) for use with XGraph or Trace graph.

NS2 uses two languages. NS-2 is written in C++ with OTcl interpreter as a front end. For efficiency reason NS separates the data path implementation from control path implementations.

TCL (Tool Command Language):

Tcl was created by John Ousterhout. Tcl (Tool Command Language) is a very powerful but easy to learn dynamic programming language suitable for a very wide range of uses including web and desktop applications, networking, Administration, testing and many more. Tcl is a mature yet evolving language that is truly cross platform easily deployed and highly extensible.

Tcl features include:

- ☐ It is available free of cost.
- ☐ Everything is a command including language structures.
- ☐ Everything can be dynamically redefined and overridden.
- ☐ All data types can be manipulated as strings including code.
- ☐ Extremely simple syntactic rules.
- ☐ Event-driven interface to sockets and files. Time based and user defined events are also possible.
- ☐ Simple exception handling using exception code returned by all command executions.
- ☐ All commands defined by Tcl itself generate informative error messages on incorrect usage.
- ☐ Readily extensible, via C, C++, Java and Tcl
- ☐ Interpreted language using byte code for improved speed while maintaining dynamic modifiability
- ☐ Platform independent: Win 32, UNIX, Linux Mac, etc.
- ☐ Easy to maintain code. Tcl scripts are often more compact and readable than functionally equivalent code in other languages.

Otcl (Object Tool Command Language):

Object Tcl is an extension to Tcl / Tk for object-oriented programming. An OTcl script will do the following.

- Initiates an event scheduler.
- Sets up the network topology using the network objects.

·Tells traffic sources when to start /stop transmitting packets through the event scheduler.

A user can add OTcl modules to NS – 2 by writing a new object class in OTcl. These then have to be compiled together with original source code. TclCL is the language used to provide a linkage between C ++ and OTcl. Toolkit Command Language (Tcl/OTcl) scripts are written to set up/configure network topologies .TclCL provides linkage for class hierarchy, object instantiation, variable binding and command dispatching. OTcl is used for periodic or triggered events NAM. NAM provides a visual interpretation of the network topology created. The application was developed as part of the VINT project. Its features are as follows.

- ☐ Provides a visual interpretation of the network created
- ☐ Can be executed directly from a Tcl script
- ☐ Controls include play, stop ff, rw, pause, a display speed controller and a packet monitor facility.
- ☐ Presents information such as throughput, number packets on each link.
- ☐ Provides a drag and drop interface for creating topologies.

NAM is an animation tool for viewing network simulation traces and real world packet trace data. The design theory behind nam was to create an animator that is able to read large animation data sets and be extensible enough so that it could be used indifferent network visualization situations. Under this constraint nam was designed to read simple animation event commands from a large trace file. In order to handle large animation data sets a minimum amount of information is kept in memory. Event commands are kept in the file and reread from the file whenever necessary. The first step to use nam is to produce the trace file. The trace file contains topology information, e.g. nodes. Links, as well as packet traces. Usually, the trace file is generated by NS. During an NS simulation, user can produce topology configurations, layout information, and packet traces using tracing events in Ns. However any application can generate a nam trace file. When the trace file is generated, t is ready to be animated by nam. Upon startup, nam will read the trace file, create topology, pop up a window, do layout if necessary, and then pause at time 0, through its user interface, nam provides control over many aspects of animation.

Starting up nam will first create the nam console window. You can have multiple animations running under the same nam instance. At the top of all nam windows is a menu bar. For the nam

console there are 'File' and 'Help' menus. Under the File there is a 'New' command for creating a NS topology using the nam editor, an 'Open' command which allows you to open existing trace files, a 'Win List' commands that popup a window will the names of all currently opened trace files, and a 'Quit' command which exits nam. The 'Help' menu contains a very limited popup help screen and a command to show version and copyright information. Once a tracefile has been loaded into nam (either by using the Open menu command or by specifying the tracefile on the command line) an animation window will appear. It has a 'Save layout' command which will save the current network layout to a file and a 'Print' command which will print the current network layout.

XGraph

XGraph is an X-Windows application that includes:

- ☐ Interactive plotting and graphing
- ☐ Animation and derivatives

To use XGraph in NS-2 the executable can be called within a TCL Script. This will then load a graph displaying the information visually displaying the information of the trace file produced from the simulation. Trace Graph is a trace file analyzer that runs under Windows, Linux and UNIX Systems. Trace Graph supports the following trace file formats.

- ☐ Wired
- ☐ Satellite
- ☐ Wireless (old and new trace)

PROCEDURE: -

Code: To set-up a network of two nodes.

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

```
# defining trace files(trace file & nam file)
```

```
set tracefile1 [open out.tr w]
```

```
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile

#define finish procedure
proc finish {} {
    global ns tracefile1 namfile
    $ns flush-trace
    close $tracefile1
    close $namfile
    exec nam out.nam &
    exit 0
}

#define nodes
set n0 [$ns node]
set n1 [$ns node]
#link
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

$ns run
```

EXPERIMENT – 2

AIM: - (a) Introduction to OSI, TCP & UDP.

(b) To set up a network with two nodes; link them with duplex link, 10ms propagation delay, 1Mbps rate and DropTail procedure. Use Agent UDP with CBR traffic source.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: -

OSI

Open Systems Interconnection (OSI) is a set of internationally recognized, non-proprietary standards for networking and for operating system involved in networking functions.

1. Application Layer
2. Presentation Layer
3. Session Layer
4. Transport Layer
5. Network Layer
6. Data Link Layer
7. Physical Layer

Physical Layer

The physical layer defines the means of transmitting raw bits rather than logical data packets over a physical link connecting network nodes. The bit stream may be grouped into code words or symbols

and converted to a physical signal that is transmitted over a hardware transmission medium. The physical layer provides an electrical, mechanical, and procedural interface to the transmission medium.

Data Link Layer

Handles special data frames (packets) between the Network layer and the Physical layer. At the receiving end, this layer packages raw data from the physical layer into data frames for delivery to the Network layer. At the sending end this layer handles conversion of data into raw formats that can be handled by the Physical Layer. The data link layer is responsible for moving frames from one hop (node) to the next.

Network Layer

Handles addressing messages for delivery, as well as translating logical network addresses and names into their physical counterparts. It is responsible for deciding how to route transmissions between computers. This layer also handles the decisions needed to get data from one point to the next point along a network path. This layer also handles packet switching and network congestion control. The network layer is responsible for the delivery of individual packets from the source host to the destination host.

Transport Layer

It manages the transmission of data across a network. Manages the flow of data between parties by segmenting long data streams into smaller data chunks (based on allowed “packet” size for a given transmission medium). Reassembles chunks into their original sequence at the receiving end. Provides acknowledgements of successful transmissions and requests resends for packets which arrive with errors. The transport layer is responsible for the delivery of a message from one process to another.

Session Layer

It enables two networked resources to hold ongoing communications (called a session) across a network. Applications on either end of the session are able to exchange data for the duration of the session. This layer is Responsible for initiating, maintaining and terminating sessions, responsible for security and access control to session information (via session participant identification), Responsible for synchronization services, and for checkpoint services . The session layer is responsible for dialog control and synchronization.

Presentation Layer

Manages data-format information for networked communications (the network's translator). For outgoing messages, it converts data into a generic format for network transmission; for incoming messages, it converts data from the generic network format to a format that the receiving application can understand. This layer is also responsible for certain protocol conversions, data encryption/decryption, or data compression/decompression. A special software facility called a "*redirector*" operates at this layer to determine if a request is network related or not and forward network-related requests to an appropriate network resource. The presentation layer is responsible for translation, compression, and encryption.

Application Layer

Provides a set of interfaces for sending and receiving applications to gain access to and use network services, such as: networked file transfer, message handling and database query processing. The application layer is responsible for providing services to the user.

TCP

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another

computer. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details

UDP

The User Datagram Protocol (UDP) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768. UDP uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.[1] If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

Difference between TCP and UDP

TCP	UDP
Reliability: TCP is connection-oriented protocol. When a file or message send it will get delivered unless connections fails. If connection lost, the server will	Reliability: UDP is connectionless protocol. When you a send a data or message, you

request the lost part. There is no corruption while transferring a message.	don't know if it'll get there, it could get lost on the way. There may be corruption while transferring a message.
Ordered: If you send two messages along a connection, one after the other, you know the first message will get there first. You don't have to worry about data arriving in the wrong order.	Ordered: If you send two messages out, you don't know what order they'll arrive in i.e. no ordered.
Heavyweight: - when the low level parts of the TCP "stream" arrive in the wrong order, resend requests have to be sent, and all the out of sequence parts have to be put back together, so requires a bit of work to piece together.	Lightweight: No ordering of messages, no tracking connections, etc. It's just fire and forget! This means it's a lot quicker, and the network card / OS have to do very little work to translate the data back from the packets.
Streaming: Data is read as a "stream," with nothing distinguishing where one packet ends and another begins. There may be multiple packets per read call.	Datagrams: Packets are sent individually and are guaranteed to be whole if they arrive. One packet per one read call.
Examples: World Wide Web (Apache TCP port 80), e-mail (SMTP TCP port 25 Postfix MTA), File Transfer Protocol (FTP port 21) and Secure Shell (OpenSSH port 22) etc.	Examples: Domain Name System (DNSUDP port 53), streaming media applications such as IPTV or movies, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and online multiplayer games etc.

PROCEDURE:-**CODE:**

```
set ns [new Simulator]
# defining trace files(trace file & nam file)
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile
#define finish procedure
proc finish {} {
    global ns tracefile1 namfile
    $ns flush-trace
    close $tracefile1
    close $namfile
    exec nam out.nam &
    exit 0
}
#define nodes
set n0 [$ns node]
set n1 [$ns node]
#link
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
#create udp agent
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
#create a null agent [sink]
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

```
#create CBR
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set rate_ 0.05ms
$cbr0 attach-agent $udp0
# connect agents
$ns connect $udp0 $null0
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

Exercise: Create a wired network with 4 or more nodes with UDP agent.

EXPERIMENT - 3

AIM: - To set up a network with two nodes; link them with duplex link, 10ms propagation delay, 1Mbps rate and DropTail procedure. Use FTP over Agent TCP.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: -

TCP: - The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details

Difference between TCP and UDP

TCP	UDP
Reliability: TCP is connection-oriented protocol. When a file or message send it will get delivered unless connections fails. If connection lost, the server will request the lost part. There is no	Reliability: UDP is connectionless protocol. When you a send a data or message, you don't know if it'll get there, it could get lost on the way. There may be corruption while transferring a message.

corruption while transferring a message.	
Ordered: If you send two messages along a connection, one after the other, you know the first message will get there first. You don't have to worry about data arriving in the wrong order.	Ordered: If you send two messages out, you don't know what order they'll arrive in i.e. no ordered
Heavyweight: - when the low level parts of the TCP "stream" arrive in the wrong order, resend requests have to be sent, and all the out of sequence parts have to be put back together, so requires a bit of work to piece together.	Lightweight: No ordering of messages, no tracking connections, etc. It's just fire and forget! This means it's a lot quicker, and the network card / OS have to do very little work to translate the data back from the packets.
Streaming: Data is read as a "stream," with nothing distinguishing where one packet ends and another begins. There may be multiple packets per read call.	Datagrams: Packets are sent individually and are guaranteed to be whole if they arrive. One packet per one read call.
Examples: World Wide Web (Apache TCP port 80), e-mail (SMTP TCP port 25 Postfix MTA), File Transfer Protocol (FTP port 21) and Secure Shell (OpenSSH port 22) etc.	Examples: Domain Name System (DNS UDP port 53), streaming media applications such as IPTV or movies, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and online multiplayer games etc

PROCEDURE: -**CODE: -**

```
set ns [new Simulator]
# defining trace files(trace file & nam file)
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile
#define finish procedure
proc finish {} {
    global ns tracefile1 namfile
    $ns flush-trace
    close $tracefile1
    close $namfile
    exec nam out.nam &
    exit 0
}

#define nodes
set n0 [$ns node]
set n1 [$ns node]
#link
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
#create tcp agent
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
#create a tcpsink agent [TCPSink]
set sink0 [new Agent/TCPSink]
```



```
$ns attach-agent $n1 $sink0
#create FTP
set ftp0 [new Application/FTP]
$tcp0 set packetSize_ 500
$ftp0 attach-agent $tcp0
# connect agents
$ns connect $tcp0 $sink0
$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"
$ns at 5.0 "finish"
$ns run
```

Exercise: Create a wired network with 4 or more nodes with TCP agent as well as UDP agent.

EXPERIMENT - 4

AIM: - To implement wired LAN connection in NS2.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: -

NS simulator simulates the levels related to the local area network: link layer Protocols (such as ARQ (automatic repeat request), the MAC protocol (Ethernet or token ring)) and the physical channel.

The best way to define a LAN through which a group of nodes is connected is by the command

Set lan [\$ns newLan <arguments>]

There are seven arguments:

1. a group of nodes ex.”\$n3 \$n4 \$n5”,
2. the delay ,
3. the bandwidth
4. a link layer type (eg ””TTL)
5. the interference queue type ,ex “QueueDropTail”,
6. the MAC type (eg “Mac/Csma/Cd”or “ Mac/802_3”)
7. the channel type (eg.”channel”)

As an eg, consider the network in the figure below in which node n3,n4,n5 are put on a common LAN.

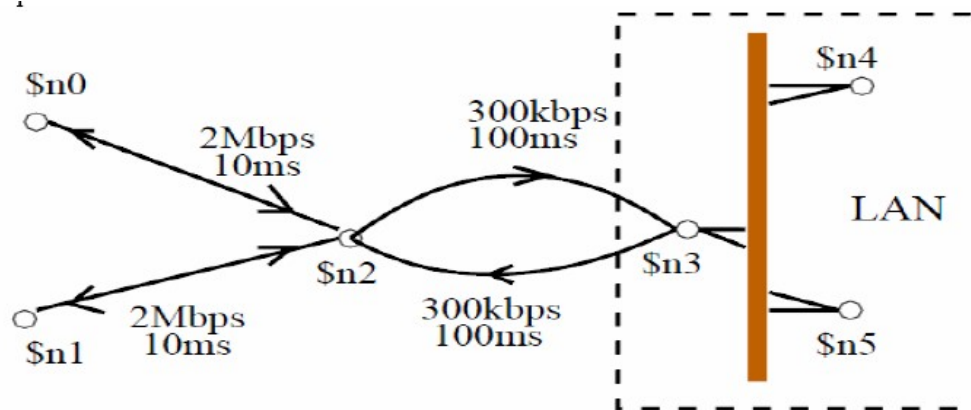


Fig: A LAN example

This means that the TCP packets designated to node n4 arrive also at node n5 (and are dropped there), TCP acknowledgements sent by node n4 to node n0 also arrive at node n5 (and are dropped there) and the UDP packets Designated to node n5 also arrive at node n4 (and are dropped there).

PROCEDURE:-

Code:

```
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
```

```
$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail
```

Is replaced By the command

```
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]
```

Exercise: Create a LAN connection in network involving seven nodes; such as node1 and node2 are linked with duplex link, (10ms propagation delay, 1Mbps rate), node 2 and node 3 with 10ms propagation delay, 2Mbps rate, such as node 3 and node4 are linked with duplex link, 10ms

propagation delay, 1Mbps rate while node 4,5 ,6 ,7 are put on a common LAN. Node 2 is acting as a TCP host while node 5 is acting as TCP sink, and node 1 is acting as UDP host and node 3 as UDP sink and label all the node.

EXPERIMENT - 5

AIM: - To Create a mobile ad-hoc network with 3 nodes in 500*400 topography with following initial positions and movements:

- a)- Node 0 (5, 5) Node 1 (490,285) Node 2 (150,240)
- b)- At $t=10$, 0 moves towards (250,250) at 3m/sec.
- c)- At $t=15$, 1 moves towards (45,285) at 5m/sec.
- d)- At $t=110$, 0 moves towards (480,300) at 5m/sec.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: -

There are two approaches for wireless communication between two hosts. The first is the centralized cellular network in which each mobile is connected to one or more fixed base stations, so that a communication between two mobile stations require to involve one or more base stations. A second decentralized approach is based on adhoc network between users that wish to communicate between each other. Due to the more limited range of a mobile terminal (with respect to a fixed base station) this requires the mobile nodes not only to be sources or destination of packets but also to forward packets between mobiles. Cellular station has much larger range than ad-hoc networks. However, ad-hoc networks have the advantage of being quickly deployable as they do not require an existing infrastructure.

Simulating scenario:

We start by presenting simple script that runs a single TCP connection over a 3- node network over an area of size of 500m x 400m depicted below. The location process is as follows:



Fig: example of a three node ad-hoc network

1. The initial locations of nodes 0, 1 and 2 are respectively (5, 5), (490,285) and (150,240) (the z – coordinate is assumed throughout to be 0).

2. At time 10, node 0 starts moving towards point (250,250) at a speed of 3 m/sec.

At time 15, node 1 starts moving towards point (45,285) at a speed of 5 m/sec.

At time 110, node 0 starts moving towards point (480,300) at a speed of 5 m/sec.

Node 2 is still throughout the simulations.

The simulations lasts 150 sec. At time 10, a TCP connection is initiated between node 0 and node 1.

Code: A 3-node example for adhoc simulation with DSDV

PROCEDURE:-

We begin our script wireless1xx.tcl with a list of these different parameters described above, as follows:

```
# Define options
set val(chan)    Channel/WirelessChannel ; # channel type
set val(prop)    Propagation/TwoRayGround ;# radio-propagation model
set val(ant)     Antenna/OmniAntenna      # Antenna type
set val(ll)      LL                        # Link layer type
```

```

set val(ifq)      Queue/DropTail/PriQueue  # Interface queue type
set val(ifqlen)   50                      # max packet in ifq
set val(netif)    Phy/WirelessPhy         # network interface type
set val(mac)      Mac/802_11              # MAC type
set val(rp)       DSDV                    # ad-hoc routing protocol
set val(nn)       3                       # number of mobile nodes
set val(x)        500                     # X dimension of the topography
set val(y)        400                     # Y dimension of the topography
set val(stop)     150                     # simulation time

```

Create an instance of the simulator,

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

After creation of ns, the simulator instance, open a file (wireless1-out.tr) for wireless traces. Also we are going to set up nam traces.

```

set tracefd [open wireless1-out.tr w] ;      # for wireless traces

$ns trace-all $tracefd

set namtrace [open wireless1-out.nam w] ;# for nam tracing

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

```

Next create a topology object that keeps track of movements of mobile nodes within the topological boundary.

```
set topo [new Topography]
```

We had earlier mentioned that mobile nodes move within a topology of 500mX400m. We provide the topography object with x and y co-ordinates of the boundary, (x = 500, y = 400):

```
$topo load_flatgrid $val(x) $val(y)
```

The topography is broken up into grids and the default value of grid resolution is 1. A different value can be passed as a third parameter to load_flatgrid {} above.

Next we create the object God, as follows:

```
create-god $val(nn)
```

Quoted from CMU document on god, "God (General Operations Director) is the object that is used to store global information about the state of the environment, network or nodes that an omniscient observer would have, but that should not be made known to any participant in the simulation." Currently, God object stores the total number of mobile nodes and a table of shortest number of hops required to reach from one node to another. The next hop information is normally loaded into god object from movement pattern files, before simulation begins, since calculating this on the fly during simulation runs can be quite time consuming. However, in order to keep this example simple we avoid using movement pattern files and thus do not provide God with next hop information.

The procedure create-god is defined in ~ns/tcl/mobility/com.tcl, which allows only a single global instance of the God object to be created during a simulation. In addition to the evaluation functionalities, the God object is called internally by MAC objects in mobile nodes. So even though we may not utilize God for evaluation purposes, (as in this example) we still need to create God.

```
# Configure nodes
```

```
$ns node-config -adhocRouting $val(rp) \  
-llType $val(ll) \  
-topo $val(topo) -x $val(x) -y $val(y) -z $val(z) -chanType $val(chanType) -macType $val(macType) -mobilityType $val(mobilityType) -rtt $val(rtt) -nsd $val(nsd) -nsdType $val(nsdType) -nsdSize $val(nsdSize) -nsdType $val(nsdType) -nsdSize $val(nsdSize) -nsdType $val(nsdType) -nsdSize $val(nsdSize)
```



```
-macType $val(mac) \  
-ifqType $val(ifq) \  
-ifqLen $val(ifqlen) \  
-antType $val(ant) \  
-propType $val(prop) \  
-phyType $val(netif) \  
-topoInstance $topo \  
-channelType $val(chan) \  
-agentTrace ON \  
-routerTrace ON \  
-macTrace OFF \  
-movementTrace OFF
```

Next, we create mobile nodes.

```
for {set i 0} {$i < $val(nn)} {incr i} {  
    set node_($i) [$ns node ]  
    $node_($i) random-motion 0    ;# disable random motion  
}
```

The random-motion for nodes is disabled here, as we are going to provide node position and movement (speed & direction) directives next. Now that we have created mobile nodes, we need to give them a position to start with,

```
# Provide initial (X,Y, for now Z=0) co-ordinates for node_(0) and node_(1)
```

```
$node_(0) set X_ 5.0
```

```
$node_(0) set Y_ 5.0
```

\$node_(0) set Z_ 0.0

\$node_(1) set X_ 490.0

\$node_(1) set Y_ 285.0

\$node_(1) set Z_ 0.0

\$node_(2) set X_ 150.0

\$node_(2) set Y_ 240.0

\$node_(2) set Z_ 0.0

Next produce some node movements,

Node_(0) starts to move towards point 250,250

\$ns at 10.0 "\$node_(0) setdest 250.0 250.0 3.0"

Node_(1) starts to move towards point 45,285

\$ns at 15.0 "\$node_(1) setdest 45.0 285.0 5.0"

Node_(0) starts to move towards point 480,300

\$ns at 110.0 "\$node_(0) setdest 480.0 300.0 5.0"

Next setup traffic flow between the two nodes as follows:

```
# TCP connections between node_(0) and node_(1)
```

```
set tcp [new Agent/TCP]
```

```
$tcp set class_2
```

```
set sink [new Agent/TCPSink]
```

```
$ns attach-agent $node_(0) $tcp
```

```
$ns attach-agent $node_(1) $sink
```

```
$ns connect $tcp $sink
```

```
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
```

```
$ns at 10.0 "$ftp start"
```

This sets up a TCP connection between the two nodes with a TCP source on node0.

Next add the following lines for providing initial position of nodes in nam.

```
# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

    # 20 defines the node size in nam, must adjust it according to your scenario size.

    # The function must be called after mobility model is defined

    $ns initial_node_pos $node_($i) 20

}
```

Then we need to define stop time when the simulation ends and tell mobile nodes to reset which actually resets their internal network components,

```
# Tell nodes when the simulation ends

#

for {set i 0} {$i < $val(nn)} {incr i} {

    $ns at 150.0 "$node_($i) reset";

}

# Ending nam and the simulation

#

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "stop"

$ns at 150.0002 "puts \"NS EXITING...\" ; $ns halt"
```

```
proc stop {} {  
    global ns tracefd namtrace  
    close $tracefd  
    close $namtrace  
    exec nam wireless1-out.nam &  
}
```

At time 150.0s, the simulation shall stop. The nodes are reset at that time and the "\$ns_ halt" is called at 150.0002s, a little later after resetting the nodes. The procedure stop{} is called to flush out traces and close the trace file.

And finally the command to start the simulation,

```
puts "Starting Simulation..."  
  
$ns run
```

Save the file wireless1xx.tcl. Next run the simulations in the usual way (type at prompt: "ns wireless1xx.tcl")

Exercise: Create a mobile network of four nodes.

EXPERIMENT - 6

AIM: - To create a Heterogeneous Network (wired cum wireless network).

APPARATUS/PLATFORM USED: -Ubuntu, NS 2.34

THEORY: -

A simple wired-cum-wireless scenario

The wireless simulation described in experiment -4 supports multi-hop ad-hoc networks. But we may need to simulate a topology of multiple LANs connected through wired nodes, or in other words we need to create a wired-cum-wireless topology.

In this section we are going to extend the simple wireless topology to create a mixed scenario consisting of a wireless and a wired domain, where data is exchanged between the mobile and non-mobile nodes. We are going to make modifications to the tcl script called wirelessxx.tcl and name the resulting wired-cum-wireless scenario file wireless2xx.tcl.

For the mixed scenario, we are going to have 2 wired nodes, W(0) and W(1), connected to our wireless domain consisting of 3 mobile nodes (nodes 0, 1 & 2) via a base-station node, BS. Base station nodes are like gateways between wireless and wired domains and allow packets to be exchanged between the two types of nodes.

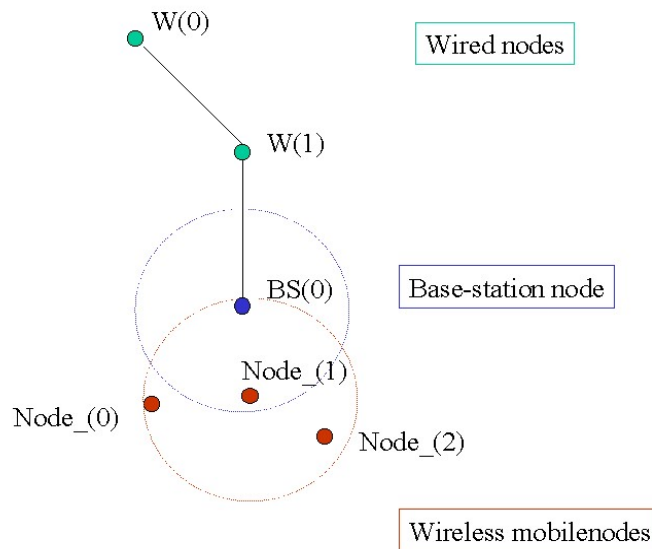


Fig1. Topology for wired-cum-wireless simulation example.

Let us begin by checking what changes need to be made to the list of variables defined at the beginning of wireless1xx.tcl.

The Adhoc routing protocol is changed to DSDV. Also, we define TCP and CBR connections between the wired and wireless nodes in the script itself. So we won't need to use the connection pattern file used in earlier simulation. Also change the simulation stop time. Note here that we use array opt() instead of val() simply to illustrate that this is no longer a global array variable and its scope is defined only in the test script.

PROCEDURE:-

```
set opt(adhocRouting) DSDV

set opt(cp) "" ; # cp file not used
```

```
set opt(stop)      300    ; # time to stop simulation
```

We define the start times for TCP flows here:

```
set opt(ftp1-start) 160.0
```

```
set opt(ftp2-start) 170.0
```

Also add the following line to define number of wired and base-station nodes:

```
set num_wired_nodes 2
```

```
set num_bs_nodes    1
```

Now we move to the main part of the program. For mixed simulations we need to use hierarchical routing in order to route packets between wireless and wired domains. The routing information for wired nodes are based on connectivity of the topology, i.e., how are nodes connected to one another through Links. This connectivity information is used to populate the forwarding tables in each wired node. However wireless nodes have no concept of "links". Packets are routed in a wireless topology using their adhoc routing protocols which build forwarding tables by exchanging routing queries among its neighbours. So, in order to exchange packets among these wired and wireless nodes, we use base-stations which act as gateways between the two domains. We segregate wired and wireless nodes by placing them in different domains. Domains and sub-domains (or clusters as they are called here) are defined by means of hierarchical topology structure as shown below. After line "set ns [new Simulator]", add the following lines:

```
$ns node-config -addressType hierarchical
```



```
AddrParams set domain_num_ 2; # number of domains

lappend cluster_num 2 1 ;      # number of clusters in each domain.

AddrParams set cluster_num_ $cluster_num

lappend eilastlevel 1 1 4;      # number of nodes in each cluster

AddrParams set nodes_num_ $eilastlevel; # for each domain
```

In the above lines we first configure the node object to have address type as Hierarchical.

Next the topology hierarchy is defined. Number of domains in this topology is 2 (one for the wired nodes and one for the wireless). Number of clusters in each of these domains is defined as "2 1" which indicates the first domain (wired) to have 2 clusters and the second (wireless) to have 1 cluster. The next line defines the number of nodes in each of these clusters which is "1 1 4"; i.e, one node in each of the first 2 clusters (in wired domain) and 4 nodes in the cluster in the wireless domain. So the topology is defined into a 3-level hierarchy (see the topology figure above).

Next we setup tracing for the simulation. Note here that for wired-cum-wireless simulation traces may be generated for both wired and wireless domains. Both the traces are written into the same output file defined here as wireless2-out.tr. In order to differentiate wireless traces from wired ones, all wireless traces begin with "WL". We also setup nam traces.

```
set tracefd [open wireless2-out.tr w]

set namtrace [open wireless2-out.nam w]

$ns trace-all $tracefd

$ns namtrace-all-wireless $namtrace $opt(x) $opt(y)
```

Next we need to create the wired, wireless and base-station nodes. Note here that for all node creations, you have to pass the hierarchical address of the node.

Create god, edit the following lines for creating wired nodes:

```
create-god [expr $opt(nn) + $num_bs_nodes]

# create wired nodes

set temp {0.0.0 0.1.0}      ; # hierarchical addresses to be used

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

    set W($i) [$ns node [lindex $temp $i]] }
```

In order to create base-station node, we need to configure the node structure. Since base-station nodes are gateways between wired and wireless domains they need to have wired routing mechanism turned on which is done by setting node-config option -wiredRouting ON. After creating the base-station node we reconfigure for wireless node and so turn wiredRouting OFF. All other node-config options used for base-station remains the same for mobile node. Also the BS(0) node is assigned as the base-station node for all the mobile nodes in the wireless domain, so that all packets originating from mobile nodes and destined outside the wireless domain, will be forwarded by mobile nodes towards their assigned base-station.

Note that it is important for the base-station node to be in the same domain as the wireless nodes. This is so that all packets originating from the wired domain, and destined for a wireless node will reach the base-station which then uses its adhoc routing protocol to route the packet to its correct destination. Thus in a mixed simulation involving wired and wireless nodes its necessary:

- 1) To turn ON hierarchical routing
- 2) To create separate domains for wired and wireless nodes. There may be multiple wired and wireless domains to simulate multiple networks.

3) To have one base-station node in every wireless domain, through which the wireless nodes may communicate with nodes outside their domain.

Let us go step by step for this example to see how the hierarchy is created. Here we have two domains, domain 0, for wired and domain 1, for wireless. The two wired nodes are placed in 2 separate clusters, 0 and 1; thus their addresses look like 0(domain 0).0(cluster 0).0(only node) and 0 (same domain 0).1(cluster 1).0(again only node).

As for the wireless nodes, they are in domain 1; we have defined one cluster (0), so all nodes are in this cluster. Hence the addresses are:

Base-station: 1(second domain,1).0(cluster 0).0(first node in cluster)

WL node#1 : 1.0.1(second node in cluster)

WL node#2 : 1.0.2(third node)

WL node#3 : 1.0.3(fourth node)

We could have placed the two wired nodes in the same cluster in wired domain 0. Also we could have placed other wireless nodes in different clusters in wireless domain 1. Also depending on our topology we may have got rid of clusters altogether, and simply have had 2 layers of hierarchy, the domains and the nodes.

```
# configure for base-station node and mobile nodes
```

```
$ns node-config -adhocRouting $opt(adhocRouting) \
```

```
-llType $opt(ll) \
```

```
-macType $opt(mac) \
```

```
-ifqType $opt(ifq) \
```

```
-ifqLen $opt(ifqlen) \
```

```
-antType $opt(ant) \  
  
-propType $opt(prop) \  
  
-phyType $opt(netif) \  
  
-channelType $opt(chan)  
  
-topoInstance $topo \  
  
-wiredRouting ON \  
  
# for base-station node onl  
  
-agentTrace ON \  
  
-routerTrace OFF \  
  
-macTrace OFF  
  
  
#create base-station node  
  
set temp {1.0.0 1.0.1 1.0.2 1.0.3} ;# hier address to be used for  
;# wireless domain.  
  
set BS(0) [ $ns node [lindex $temp 0]]  
  
$BS(0) random-motion 0 ;# disable random motion  
  
#provide some co-ordinates (fixed) to base station node  
  
$BS(0) set X_ 1.0  
  
$BS(0) set Y_ 2.0
```

```
$BS(0) set Z_ 0.0

# create mobilenodes in the same domain as BS(0)

# Give the position and movement of mobilenodes.

# Note there has been a change of the earlier AddrParams function 'set-hieraddr' to
'addr2id'.

#configure for mobilenodes

$ns node-config -wiredRouting OFF

# now create mobilenodes

for {set j 0} {$j < $opt(nn)} {incr j} {

    set node_($j) [ $ns node [lindex $temp \ [expr $j+1]] ]

    $node_($j) base-station [AddrParams addr2id \ [$BS(0) node-addr]];

    # provide each mobilenode with hier address of its base-station

}
```

Next connect wired nodes and BS and setup TCP traffic between wireless node, node_(0) and wired node W(0), and between W(1) and node_(2), as shown below:

```
#create links between wired and BS nodes

$ns duplex-link $W(0) $W(1) 5Mb 2ms DropTail

$ns duplex-link $W(1) $BS(0) 5Mb 2ms DropTail
```

\$ns duplex-link-op \$W(0) \$W(1) orient down

\$ns duplex-link-op \$W(1) \$BS(0) orient left-down

setup TCP connections

set tcp1 [new Agent/TCP]

\$tcp1 set class_2

set sink1 [new Agent/TCPSink]

\$ns attach-agent \$node_(0) \$tcp1

\$ns attach-agent \$W(0) \$sink1

\$ns connect \$tcp1 \$sink1

set ftp1 [new Application/FTP]

\$ftp1 attach-agent \$tcp1

\$ns at \$opt(ftp1-start) "\$ftp1 start"

set tcp2 [new Agent/TCP]

\$tcp2 set class_2

set sink2 [new Agent/TCPSink]

\$ns attach-agent \$W(1) \$tcp2

\$ns attach-agent \$node_(2) \$sink2

```
$ns connect $tcp2 $sink2  
  
set ftp2 [new Application/FTP]  
  
$ftp2 attach-agent $tcp2  
  
$ns at $opt(ftp2-start) "$ftp2 start"
```

This would be followed by the remaining lines from wireless1xx.tcl (defining mobile nodes position and movement, and telling mobile nodes when to stop and finally running ns).

Run the script. The ns and nam trace files are generated at the end of simulation run. Running wireless2-out.nam shows the movement of mobile nodes and traffic in the wired domain. In trace file wireless2-out.tr we see traces for both wired domain and wireless domain (preceding with "WL" for wireless). At 160.0s, a TCP connection is setup between _3_, (which is node_(0)) and 0, (which is W(0)). Note that the node-ids are created internally by the simulator and are assigned in the order of node creation. At 170s, another TCP connection is setup in the opposite direction, from the wired to the wireless domain.

Exercise: Create a heterogeneous network of 8 nodes.

EXPERIMENT - 7

AIM: - To Interpret data trace file (.tr) of Wired, Wireless and LAN Networks.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: -

NS simulation can produce both the visualization trace (NAM) as well as an ASCII file trace corresponding to the events registered at the network. When we use tracing ns inserts four objects in the link: EnqT, DeqT, RecvT and DrpT as indicated below:

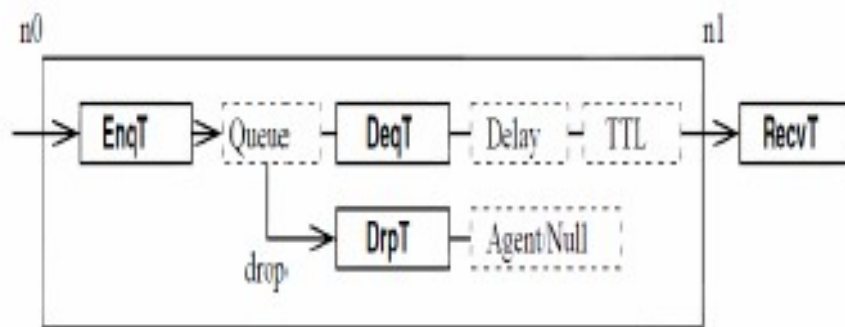


Fig: tracing objects in a simplex link

EnqT registers information concerning a packet that arrives and is queued at the input queue of the link. If the packet overflows then information concerning the dropped packet are handled by the DrpT. DeqT registers information at the instant the packet is dequeued. Finally RecvT gives us information about packets that have been received at the output of the link.

Structure of trace files:

Event	Time	From node	To node type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id
-------	------	-----------	--------------	----------	-------	-----	----------	----------	---------	--------

Fig : fields appearing in a trace

1. Type Identifier:

- “+”: a packet enqueue 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: This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script. One can further use this field for analysis purposes; it is also used when specifying stream colour for the NAM file.

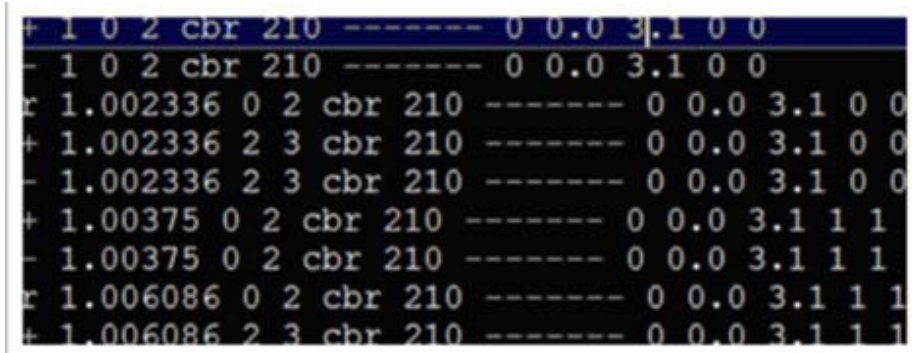
9-10. Source Address and Destination Address: the format of these two fields is “a.b”, where “a” is the address and “b” is the port.

11. Sequence Number: This is the network layer protocol’s packet sequence number, ns keeps track of UDP implementations in real network do not use sequence number, ns keeps track of the UDP packet sequence number for analysis purposes.

12. Packet Unique ID: The last field shows the unique id of the packet.

Example of trace files

When using trace-all in NS2, a trace string is created in a trace file. The trace file would look like this



```

+ 1 0 2 cbr 210 ----- 0 0.0 3.1 0 0
- 1 0 2 cbr 210 ----- 0 0.0 3.1 0 0
r 1.002336 0 2 cbr 210 ----- 0 0.0 3.1 0 0
+ 1.002336 2 3 cbr 210 ----- 0 0.0 3.1 0 0
- 1.002336 2 3 cbr 210 ----- 0 0.0 3.1 0 0
+ 1.00375 0 2 cbr 210 ----- 0 0.0 3.1 1 1
- 1.00375 0 2 cbr 210 ----- 0 0.0 3.1 1 1
r 1.006086 0 2 cbr 210 ----- 0 0.0 3.1 1 1
+ 1.006086 2 3 cbr 210 ----- 0 0.0 3.1 1 1
  
```

How to interpret the NS2 tracefile for wireless simulation?

To find the interpretation of all possible trace format when you do the wireless simulation, you'd better read the code of ns2 in file *ns2home/trace/cmu-trace{.h, .cc}* Mostly, the format would be as

ACTION: [s|r|D]: s -- sent, r -- received, D – dropped

WHEN: the time when the action happened

WHERE: the node where the action happened

LAYER: AGT -- application,

RTR -- routing,

LL -- link layer (ARP is done here)

IFQ -- outgoing packet queue (between link and mac layer)

MAC -- mac,

PHY – physical flags:

SEQNO: the sequence number of the packet

TYPE: the packet type

CBR -- CBR data stream packet

DSR -- DSR routing packet (control packet generated by routing)

RTS -- RTS packet generated by MAC 802.11

ARP -- link layer ARP packet

SIZE: the size of packet at current layer, when packet goes down, size increases, goes up size decreases

[a b c d]: a -- the packet duration in mac layer header

b -- the mac address of destination

c -- the mac address of source

d -- the mac type of the packet body

flags:

[.....]: [

source node ip : port_number

destination node ip (-1 means broadcast) : port_number

ip header ttl

ip of next hop (0 means node 0 or broadcast)

]

So we can interpret the below trace

s 76.000000000 _98_ AGT --- 1812 cbr 32 [0 0 0 0] ----- [98:0 0:0 32 0]

as Application 0 (port number) on node 98 sent a CBR packet whose ID is 1812 and size is 32 bytes, at time 76.0 second, to application 0 on node 0 with TTL is 32 hops. The next hop is not decided yet.

And we can also interpret the below trace

r 0.010176954 _9_ RTR --- 1 gpsr 29 [0 ffffffff 8 800] ----- [8:255 -1:255 32 0]

in the same way, as The routing agent on node 9 received a GPSR broadcast (mac address 0xff, and ip address is -1, either of them means broadcast) routing packet whose ID is 1 and size is 19 bytes, at time 0.010176954 second, from node 8 (both mac and ip addresses are 8), port 255 (routing agent).

Exercise: Interpret data trace file (.tr) of Wired, LAN, Mobile and Heterogeneous Networks.

EXPERIMENT - 8

AIM: - Throughput calculation for TCP or UDP in Wired network.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: - The definition of link throughput from node F to T is given below:

$$\gamma = \frac{\text{no. of bits from node } F \text{ to node } T}{\text{Observation duration}}$$

PROCEDURE: -

Throughput.awk

```
BEGIN {  
  
FromNode=1;  
  
toNode=2;  
  
lineCount = 0;  
  
totalBits = 0;  
  
}
```

```
/^r/==$3==fromNode==$4==toNode {  
  
totalBits += 8*$6;  
  
if ( lineCount==0 ) {  
  
timeBegin = $2; lineCount++;  
  
} else {  
  
timeEnd = $2;  
  
};  
  
};  
  
END{  
  
duration = timeEnd-timeBegin;  
  
print "Number of records is " NR;  
  
print "Output: ";  
  
print "Transmission: N" fromNode "->N" toNode;  
  
print " - Total transmitted bits = " totalBits " bits";  
  
print " - duration = " duration " s";  
  
print " - Throughput = " totalBits/duration/1e3 " kbps.";  
  
};
```

Again, an AWK usually consists of three sections: BEGIN, Pattern matching, and END.

The Beginning

Again, the initialization of an AWK program is given in curly braces following the keyword “BEGIN”. Here we set the beginning node fromNode to 1 and ending node toNode to 2. We also reset totalBits and lineCount to zero.

The End

This is where we process the results (see the program in the curly braces following END):

- Simulation duration = timeEnd - timeBegin
- timeBegin = time of the first line in the tracefile
- timeEnd = time of the last line in the tracefile
- Throughput is computed as the total number of bits (totalBits) divided by the simulation duration (duration)

Pattern Matching

This is where the real fun begins. It’s the place we filter out the unnecessary information. The main line here is

```
/^r/&&$3==fromNode&&$4==toNode {
```

```
...
```

```
};
```

which simply says that if a line match “/^r/&&\$3==fromNode&&\$4==toNode”, do whatever in the curly braces {...}.

So what we would like to do now is to collect

1. totalBit: Total number of bits transmitted over the link connecting the node fromNode to the node toNode
2. (timeBegin,timeEnd): in order to compute the simulation duration

Total Number of Bits

Total number of bits is defined as the number of bits leaving node fromNode and arrive node toNode. Therefore, it corresponds to the lines which matches 3 following conditions:

- Begins with 'r',
- Have the 3rd column as fromNode
- Have the 4th column as toNode

This condition corresponds to the following AWK statement:

```
/^r/ && $3 == fromNode && $4 == toNode
```

For a line matching with the above condition, we shall do things in the curly braces. The first thing here is `totalBits += 8*$6;`

which is to add up the 6th column of the line to totalBits. From [NS Trace Format (exp 7)], the 6th column is the packet size in bytes. By doing this, we add up all data bytes which leaves node fromNode and arrives node toNode.

Simulation Duration

Simulation duration is collected by recording

- The time corresponds to the first matching records (timeBegin); i.e., the time that the

first data packet leaving node fromNode arrives the node toNode

- The time corresponds to timeBegin is set the last matching records (**timeEnd**); i.e., the time that the last data packet leaving node fromNode arrives the node toNode

In order to differentiate the first from other records, a counter is used to **count** which initialized to zero at the beginning of the program. When count is zero (meaning the first matching line), to the value in the 2nd column (i.e., time of the record). For any other matched record, the time is also recorded into timeEnd. So at the end of process time corresponding to the last matched record would be stored in timeEnd.

- Exercise:**
1. Calculate throughput for a TCP link.
 2. Calculate throughput for a UDP link.

EXPERIMENT - 9

AIM: - To Create a network with 4 nodes 0-2, 1-2, 2-3 with TCP from 0-3 and UDP from 1-3. Apply an error model on link 2-3 with error rate 0.2 and uniform distribution. Apply queue monitor on 2-3 link and interpret any five lines of qm.out file.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: - Error model simulates link-level errors or loss by either marking the packet's error flag or dumping the packet to a drop target. In simulations, errors can be generated from a simple model such as the packet error rate, or from more complicated statistical and empirical models. To support a wide variety of models, the unit of error can be specified in term of packet, bits, or time-based.

Queue monitoring refers to the capability of tracking the dynamics of packets at a queue (or other object). A queue monitor tracks packet arrival/departure/drop statistics, and may optionally compute averages of these values. Monitoring may be applied to all packets (aggregate statistics), or per-flow statistics (using a Flow Monitor).

PROCEDURE: -

CODE:

```
set ns [new Simulator]

#defining trace files
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
set namfile [open out.nam w]
$ns namtrace-all $namfile

proc finish {} {
    global ns tracefile1 namfile
```

```
$ns flush-trace
close $tracefile1
close $namfile
exec nam out.nam &
exit 0
}
```

```
#define nodes
```

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

```
$n0 label "N0"
```

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

```
$n1 label "N1"
```

```
set n2 [$ns node]
```

```
$n2 label "N2"
```

```
set n3 [$ns node]
```

```
$n3 label "N3"
```

```
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
```

```
$ns duplex-link-op $n0 $n2 orient right-down
```

```
$ns duplex-link-op $n0 $n2 color "green"
```

```
$ns duplex-link-op $n0 $n2 label "Link1"
```

```
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
```

```
$ns duplex-link-op $n1 $n2 orient up-right
```

```
$ns duplex-link-op $n1 $n2 color "blue"
```

```
$ns duplex-link-op $n1 $n2 label "Link2"
```

```
$ns simplex-link $n3 $n2 0.07Mb 20ms DropTail
```

```
$ns simplex-link-op $n3 $n2 orient down-left
```

```
$ns simplex-link-op $n3 $n2 color "blue"
```

```
$ns simplex-link-op $n3 $n2 label "Link3"
```

```
$ns simplex-link $n2 $n3 0.07Mb 20ms DropTail
```

```
$ns simplex-link-op $n2 $n3 orient up
$ns simplex-link-op $n2 $n3 color "green"
$ns simplex-link-op $n2 $n3 label "Link4"
```

```
set udpo [new Agent/UDP]
$ns attach-agent $n1 $udpo
```

```
#create a null agent [SINK]
set nullo [new Agent/Null]
$ns attach-agent $n3 $nullo
```

```
#create CBR(-constant bit rate) traffic source
set cbro [new Application/Traffic/CBR]
$cbro set packetize_ 500
$cbro set interval_ 0.005
$cbro attach-agent $udpo
set tcpo [new Agent/TCP]
$ns attach-agent $n0 $tcpo
```

```
#create [SINK]
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcpo $sink
$tcpo set packetize_ 500
```

```
#setup FTP over TCP
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcpo
$tcpo set fid_ 1
$ns color 1 Black
```

```
$udpo set fid_ 2
```

```
$ns color 2 Red
```

```
$ns at 0.1 "$ftp start"
```

```
$ns at 130 "$ftp stop"
```

```
#connect nodes
```

```
$ns connect $udpo $nullo
```

```
$ns at 0.1 "$cbro start"
```

```
$ns at 124 "$cbro stop"
```

```
$ns at 160 "finish"
```

```
set loss_module [new ErrorModel]
```

```
$loss_module set rate_ 0.1
```

```
$loss_module ranvar [new RandomVariable/Uniform]
```

```
$loss_module drop-target [new Agent/Null]
```

```
$ns lossmodel $loss_module $n2 $n3
```

```
#queue monitoring
```

```
set queue [$ns monitor-queue $n2 $n3 [open qm.out w] 0.1];
```

```
[$ns link $n2 $n3] queue-sample-timeout;
```

```
$ns queue-limit $n2 $n3 30
```

```
#after 5 sec of simulation finish procedure will be called
```

```
$ns run
```

Exercise: Create a network with 6 nodes 0-2, 2-3, 4-6, 1-5-4 with TCP from 0-3 and UDP from 1-3.

Apply an error model on link 2-3 with error rate 0.2 and uniform distribution. Apply queue monitor on 2-3 link and interpret any five lines of qm.out file

EXPERIMENT - 10

AIM: - To create a network with 5 nodes, and apply uniform, exponential and constant error model with error rate 1% on 3 different links.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: - In addition to the basic class ErrorModel described in details below, there are several other types of error modules not being completely documented yet, which include:

- SRMErrorModel, PGMErrormodel: error model for SRM and PGM.
- ErrorModel/Trace: error model that reads a loss trace (instead of a math/computed model)
- MrouteErrorModel: error model for multicast routing, now inherits from trace.
- ErrorModel/Periodic: models periodic packet drops (drop every nth packet we see). This model can be conveniently combined with a flow-based classifier to achieve drops in particular flows
- SelectErrorModel: for Selective packet drop.
- ErrorModel/TwoState: Two-State: error-free and error
- ErrorModel/TwoStateMarkov, ErrorModel/Expo, ErrorModel/Empirical: inherit from ErrorModel/TwoState.
- ErrorModel/List: specify a list of packets/bytes to drop, which could be in any order

To use an error model for wired networks, at first it has to be inserted into a SimpleLink object. Because a SimpleLink is a composite object, an error model can be inserted to many places. Currently we provide the following methods to insert an error module into three different places.

- Insert an error module in a SimpleLink BEFORE the queue module. This is provided by the following two OTcl methods:

rX SimpleLink::errormodule args & When an error model is given as a parameter, it inserts the error module into the simple link, right after the queue module, and set the drop-target of the error model to be the drop trace object of the simple link. Note that this requires the following configuration order: ns namtrace-all followed by link configurations, followed by error model

insertion. When no argument is given, it returns the current error model in the link, if there's any. This method is defined in *ns/tcl/lib/ns-link.tcl*

`Simulator::lossmodel em src dst` & Call `SimpleLink::errormodule` to insert the given error module into the simple link (src, dst). It's simply a wrapper for the above method. This method is defined in *ns/tcl/lib/ns-lib.tcl*.

- Insert an error module in a `SimpleLink` AFTER the queue but BEFORE the delay link. This is provided by the following two methods:

`rX SimpleLink::insert-linkloss args` & This method's behavior is identical to that of `SimpleLink::errormodule`, except that it inserts an error module immediately after the queue object. It's defined in *ns/tcl/lib/ns-link.tcl*

`Simulator::link-lossmodel em src dst` & This is a wrapper for `SimpleLink::insert-linkloss`. It's defined in *ns/tcl/lib/ns-lib.tcl*

The nam traces generated by error models inserted using these two methods do not require special treatment and can be visualized using an older version of nam.

- Insert an error module in a `Link` AFTER the delay link module. This can be done by `Link::install-error`. Currently this API doesn't produce any trace. It only serves as a placeholder for possible future extensions.

PROCEDURE:-

The following is a list of error-model related commands commonly used in simulation scripts:

```
set em [new ErrorModel]
```

```
$em unit pkt
```

```
$em set rate_ 0.02
```

```
$em ranvar [new RandomVariable/Uniform]
```

`$em drop-target [new Agent/Null]`

This is a simple example of how to create and configure an error model. The commands to place the error-model in a simple link will be shown next.

`$simplelink errormodule args:` This command inserts the error-model before the queue object in simple link. However, in this case the error-model's drop-target points to the link's drophead_ element.

`$ns_ lossmodel em src dst:` This command places the error-model before the queue in a simplelink defined by the <src> and <dst> nodes. This is basically a wrapper for the above method.

`$simplelink insert-linkloss args:` This inserts a loss-module after the queue, but right before the delay link_ element in the simple link. This is because nam can visualize a packet drop only if the packet is on the link or in the queue. The error-module's drop-target points to the link's drophead_ element.

`$ns_ link-lossmodel em src dst:` This too is a wrapper method for insert-linkloss method described above. That is this inserts the error-module right after the queue element in a simple link (src-dst).

Exercise: Performance analysis of a 4 nodes network after the introduction of any of the error model discussed above.

EXPERIMENT - 11

AIM: -Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source/destination.

APPARATUS/PLATFORM USED: -Ubuntu, NS 2.34

THEORY: -Ethernet Local Area Network (ELAN) enables any-to-any connectivity for businesses that need to connect all their locations on a single network. It is the most widely installed local area network (LAN) technology. Ethernet is a link layer protocol in the TCP/IP stack, describing how networked devices can format data for transmission to other network devices on the same network segment, and how to put that data out on the network connection. It touches both Layer 1 (the physical layer) and Layer 2 (the data link layer) on the OSI network protocol model. Ethernet defines two units of transmission, packet and frame. The frame includes not just the "payload" of data being transmitted but also addressing information identifying the physical "Media Access Control" (MAC) addresses of both sender and receiver, VLAN tagging and quality of service information, and error-correction information to detect problems in transmission. Each frame is wrapped in a packet, which affixes several bytes of information used in establishing the connection and marking where the frame starts.

Specified in the family of standards known as IEEE802.3, Ethernet was originally developed by Xerox in the 1970s. Ethernet was initially designed to run over coaxial cables, but a typical Ethernet LAN now uses special grades of twisted pair cables, or fiber optical cabling. Wi-Fi standards (IEEE 802.11a, b, g, n and now ac) define the equivalent of Ethernet for Wireless LANs.) Ethernet standards are steadily evolving to embrace new media, higher transmission speeds and changes in frame content (e.g., 802.3ac to accommodate VLAN and priority tagging) and functional requirements (e.g., 802.3af, defining Power Over Ethernet [POE] crucial to most Wi-Fi and IP telephony deployments).

Ethernet initially assumed a shared medium: multiple devices on each segment of the network, connected daisy chain at first but later in star topology via Ethernet hubs (which replicated all traffic received on any port to every other port). It therefore defines a means of sharing the medium: Carrier Sense Multiple

Access with Collision Detection (CSMA/CD). Ethernet devices will check to see if anyone else is transmitting at the moment (carrier sense of multiple access) and if so (collision detection) will wait a short time before retrying the transmission.

Over time, though, hubs were replaced by switches, which send to each port only the traffic directed to the device on that port. That, combined with the migration from coaxial to twisted pair cabling (with dedicated pairs for sending and receiving data) and optical fiber, made shared-medium problems a thing of the past.

So, basically ELAN provides a multipoint-to-multipoint Ethernet WAN that extends your

LAN to a Wide Area Network that allows your company's business-critical applications and data to travel seamlessly across the entire network. ELAN can move large amounts of information between sites, quickly and securely. ELAN brings together cost effectiveness, speed, simplicity and flexibility in a broadly customizable Ethernet solution.

ELAN Benefits are given below:

- Privacy: All data travels within the secure domain of a Layer 2 dedicated, high capacity, point-to-point connection at native Ethernet speeds.
- Standards-Based: Depend on Ethernet Local Area Network from the 1st North American Service Provider to earn all 8 MEF CE 2.0 Certifications.
- Single Interconnection: A hub aggregates all data traffic on a single network linking all your business locations.
- Traffic Separation: Maintain discrete pathways when you consolidate previously separate domains for specific applications or departments onto a single network.
- Cost Savings: A single handoff reduces network equipment and management costs.

PROGRAM OUTLINES:

- 1). Create Simulator and use colors to differentiate the traffics.
- 2). Open trace and NAM trace file, finish Procedure and execute the NAM animation file.
- 3). Now calculate the number of packets dropped due to collision.

```
puts "The number of packet drops due to collision is"
exec grep "^d" prog6.tr | cut -d " " -f 4 | grep -c "3" &
exit 0
}
```

- 4). Create 6 nodes and create duplex links between the nodes.
- 5). Consider the nodes n (3), n(4) and n(5) under LAN connection and apply orientation to the nodes.
- 6). Set up queue between two nodes n(2) and n(3) and monitor the queue.
- 7). Setup TCP connection between n(0) and n(4) nodes and apply FTP application over TCP.
- 8). Setup another TCP connection between n(5) and n(1) and apply FTP application on it.
- 9). Schedule the events and run the simulation.

Code:-

```
#Create Simulator
```

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

```
#Use colors to differentiate the traffics
```

```
$ns color 1 Blue
```

```
$ns color 2 Red
```

```
#Open trace and NAM trace file
```

```
set ntrace [open prog7.tr w]
```

```
$ns trace-all $ntrace
```

```
set namfile [open prog7.nam w]
```

```
$ns namtrace-all $namfile
```

```
#Use some flat file to create congestion graph windows
```

```
set winFile0 [open WinFile0 w]
```

```
set winFile1 [open WinFile1 w]
```

#Finish Procedure

```
proc Finish {} {
```

#Dump all trace data and Close the files

```
global ns ntrace namfile
```

```
$ns flush-trace
```

```
close $ntrace
```

```
close $namfile
```

#Execute the NAM animation file

```
exec nam prog7.nam &
```

Plot the Congestion Window graph using xgraph

```
exec xgraph WinFile0 WinFile1 &
```

```
exit 0
```

```
}
```

#Plot Window Procedure

```
proc PlotWindow {tcpSource file} {
```

```
global ns
```

```
set time 0.1
```

```
set now [$ns now]
```

```
set cwnd [$tcpSource set cwnd_]
```

```
puts $file "$now $cwnd"
```

```
$ns at [expr $now+$time] "PlotWindow $tcpSource $file"
```

```
}
```

#Create 6 nodes

```
for {set i 0} {$i<6} {incr i} {
```

```
set n($i) [$ns node]
```

```
}
```

#Create duplex links between the nodes

```
$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail
```

```
$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail
```

```
$ns duplex-link $n(2) $n(3) 0.6Mb 100ms DropTail
#Nodes n(3) , n(4) and n(5) are considered in a LAN
set lan [$ns newLan "$n(3) $n(4) $n(5)" 0.5Mb 40ms LL Queue/DropTail MAC/802_3 Channel]
#Orientation to the nodes
$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns duplex-link-op $n(2) $n(3) orient right
#Setup queue between n(2) and n(3) and monitor the queue
$ns queue-limit $n(2) $n(3) 20
$ns duplex-link-op $n(2) $n(3) queuePos 0.5
#Set error model on link n(2) to n(3)
set loss_module [new ErrorModel]
$loss_module ranvar [new RandomVariable/Uniform]
$loss_module drop-target [new Agent/Null]
$ns lossmodel $loss_module $n(2) $n(3)
#Set up the TCP connection between n(0) and n(4)
set tcp0 [new Agent/TCP/Newreno]
$tcp0 set fid_ 1
$tcp0 set window_ 8000
$tcp0 set packetSize_ 552
$ns attach-agent $n(0) $tcp0
set sink0 [new Agent/TCPSink/DelAck]
$ns attach-agent $n(4) $sink0
$ns connect $tcp0 $sink0
#Apply FTP Application over TCP
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP
#Set up another TCP connection between n(5) and n(1)
set tcp1 [new Agent/TCP/Newreno]
$tcp1 set fid_ 2
```

```
$tcp1 set window_ 8000
$tcp1 set packetSize_ 552
$ns attach-agent $n(5) $tcp1
set sink1 [new Agent/TCPSink/DelAck]
$ns attach-agent $n(1) $sink1
$ns connect $tcp1 $sink1
#Apply FTP application over TCP
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
#Schedule Events
$ns at 0.1 "$ftp0 start"
$ns at 0.1 "PlotWindow $tcp0 $winFile0"
$ns at 0.5 "$ftp1 start"
$ns at 0.5 "PlotWindow $tcp1 $winFile1"
$ns at 25.0 "$ftp0 stop"
$ns at 25.1 "$ftp1 stop"
$ns at 25.2 "Finish"
#Run the simulation
$ns run
```

EXPERIMENT - 12

AIM: - Write a program for Error Detection using CRC-CCITT (16 bits)

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: -

The cyclic redundancy check, or CRC, is a technique for detecting errors in digital data, but not for making corrections when errors are detected. It is used primarily in data transmission.

In the CRC method, a certain number of check bits, often called a checksum, are appended to the message being transmitted. The receiver can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission.

If an error occurred, the receiver sends a "negative acknowledgement" (NAK) back to the sender, requesting that the message be retransmitted. The technique is also sometimes applied to data storage devices, such as a disk drive. In this situation each block on the disk would have check bits, and the hardware might automatically initiate a reread of the block when an error is detected, or it might report the error to software. The material that follows speaks in terms of a "sender" and a "receiver" of a "message," but it should be understood that it applies to storage writing and reading as well.

algorithm

1. Start
2. Enter the message to be transmitted
3. Append the message with 16(since it is 16-bit CRC) 0's (i.e. if you input 5 digit message, the appended message should be 21-bits.)
4. XOR appended message and transmit it.(Here, you compare with an already existing string such as 10001000000100001 and replace the bits the same way XOR operation works)

5. Verify the message that is received is the same as the one sent.
6. End

Code:-

```
#include <iostream>

#include <string.h>

using namespace std;

int crc (char *ip, char *op, char *poly, int mode)
{
    Strcpy (op, ip);

    if (mode) {

        for (int i = 1; i < strlen(poly); i++)

            strcat(op, "0");

    }

    /* Perform XOR on the msg with the selected polynomial */

    for (int i = 0; i < strlen(ip); i++) {

        if (op[i] == '1') {

            for (int j = 0; j < strlen(poly); j++) {

                if (op[i + j] == poly[j])

                    Op [i + j] = '0';

                else
```

```
Op [i + j] = '1';

}

}

}

}

/* check for errors. return 0 if error detected */

for (int i = 0; i < strlen(op); i++)

if (op[i] == '1')

return 0;

return 1;

}

int main()

{

char ip[50], op[50], rcv[50];

char poly[] = "100010000000100001";

cout << "Enter the input message in binary" << endl;

cin >> ip;

crc(ip, op, poly, 1);

cout << "The transmitted message is: " << ip << op + strlen(ip) << endl;

cout << "Enter the received message in binary" << endl;

cin >>
```



```
recv;  
  
    if (crc(recv, op, poly, 0))  
  
        cout << "No error in data" << endl;  
  
    else  
  
        cout << "Error in data transmission has occurred" << endl;
```

EXPERIMENT - 13

AIM: -To simulate a link failure in wired network using NS-2.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34.

THEORY: -

In the wired network, a link between any two nodes can be broken to demonstrate the link failure. In real time applications such as online games, video conferencing, telephony, and trading systems, link failure occurs due to several reasons. Such link failure even if it exists only for a short period, it leads to unbearable performance degradations. In simulation, the link between any two nodes can be failed and recovered using the command “rtmodel-at” along with keyword “down” and “up” respectively and with the time specification. The test4.tcl illustrates the link failure between two nodes in the wired network.

```
# -----ESTABLISHING COMMUNICATION -----#
```

```
#-----CLIENT1 TO ENDSERVER -----#
```

```
set tcp0 [new Agent/TCP]
$tcp0 set maxcwnd_ 16
$tcp0 set fid_ 4
$ns attach-agent $Client1 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $Endserver1 $sink0
$ns connect $tcp0 $sink0

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

```
$ns add-agent-trace $tcp0 tcp
$tcp0 tracevar cwnd_
$ns at 0.5 "$ftp0 start"
$ns at 28.5 "$ftp0 stop"
# -----CLIENT2 TO ENDSERVER1 -----#
set tcp1 [new Agent/TCP]
$tcp1 set fid_ 2
$tcp1 set maxcwnd_ 16
$ns attach-agent $Client2 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $Endserver1 $sink1
$ns connect $tcp1 $sink1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns add-agent-trace $tcp1 tcp1
$tcp1 tracevar cwnd_
$ns at 0.58 "$ftp1 start"
$ns at 28.5 "$ftp1 stop"
# ----- CLIENT3 TO ENDSERVER -----#
set tcp2 [new Agent/TCP]
$tcp2 set fid_ 0
$tcp2 set maxcwnd_ 16
$tcp2 set packetize_ 100
$ns attach-agent $Client3 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $Endserver1 $sink2
$ns connect $tcp2 $sink2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
```

```
$ns add-agent-trace $tcp2 tcp2
$tcp2 tracevar cwnd_
$ns at 0.65 "$ftp2 start"
$ns at 28.5 "$ftp2 stop"
#-----CLIENT4 TO ENDSERVER-----#

set tcp3 [new Agent/TCP]
$tcp3 set fid_ 3
$tcp3 set maxcwnd_ 16
$tcp2 set packetize_ 100
$ns attach-agent $Client4 $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $Endserver1 $sink3
$ns connect $tcp3 $sink3

set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ns add-agent-trace $tcp3 tcp3
$tcp3 tracevar cwnd_
$ns at 0.60 "$ftp3 start"
$ns at 28.5 "$ftp3 stop"
#----- Link Failure -----#

$ns rtmodel-at 2.880511 down $Router3 $Router4

#----- Link Failure Recovery -----#

$ns rtmodel-at 5.880511 up $Router3 $Router4
```

EXPERIMENT - 14

AIM: -Write a program for distance vector algorithm to find suitable path for transmission.

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34

THEORY: - Distance Vector algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms. Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The starting assumption for distance-vector routing is each node knows the cost of the link of each of its directly connected neighbors. Next, every node sends a configured message to its directly connected neighbors containing its own distance table. Now, every node can learn and up its distance table with cost and next hops for all nodes network. Repeat exchanging until no more information between the neighbors. Consider a node A that is interested in routing to destination H via a directly attached neighbor J. Node A's distance table entry, $D_x(Y,Z)$ is the sum of the cost of the direct-one hop link between A and J, $c(A,J)$, plus neighboring J's currently known minimum-cost path (shortest path) from itself(J) to H. That is $D_x(H,J) = c(A,J) + \min_w \{D_j(H,w)\}$ The \min_w is taken over all the J's This equation suggests that the form of neighbor-to-neighbor communication that will take place in the DV algorithm - each node

must know the cost of each of its neighbors' minimum-cost path to each destination. Hence, whenever a node computes a new minimum cost to some destination, it must inform its neighbors of this new minimum cost.

Implementation algorithm: -

1. Send my routing table to all my neighbors whenever my link table changes
2. When I get a routing table from a neighbor on port P with link metric M:
 - a. add L to each of the neighbor's metrics
 - b. for each entry (D, P', M') in the updated neighbor's table:
 - i. if I do not have an entry for D, add (D, P, M') to my routing table
 - ii. if I have an entry for D with metric M'', add (D, P, M') to my routing table if $M' < M''$
 - iii. if my routing table has changed, send all the new entries to all my neighbors.

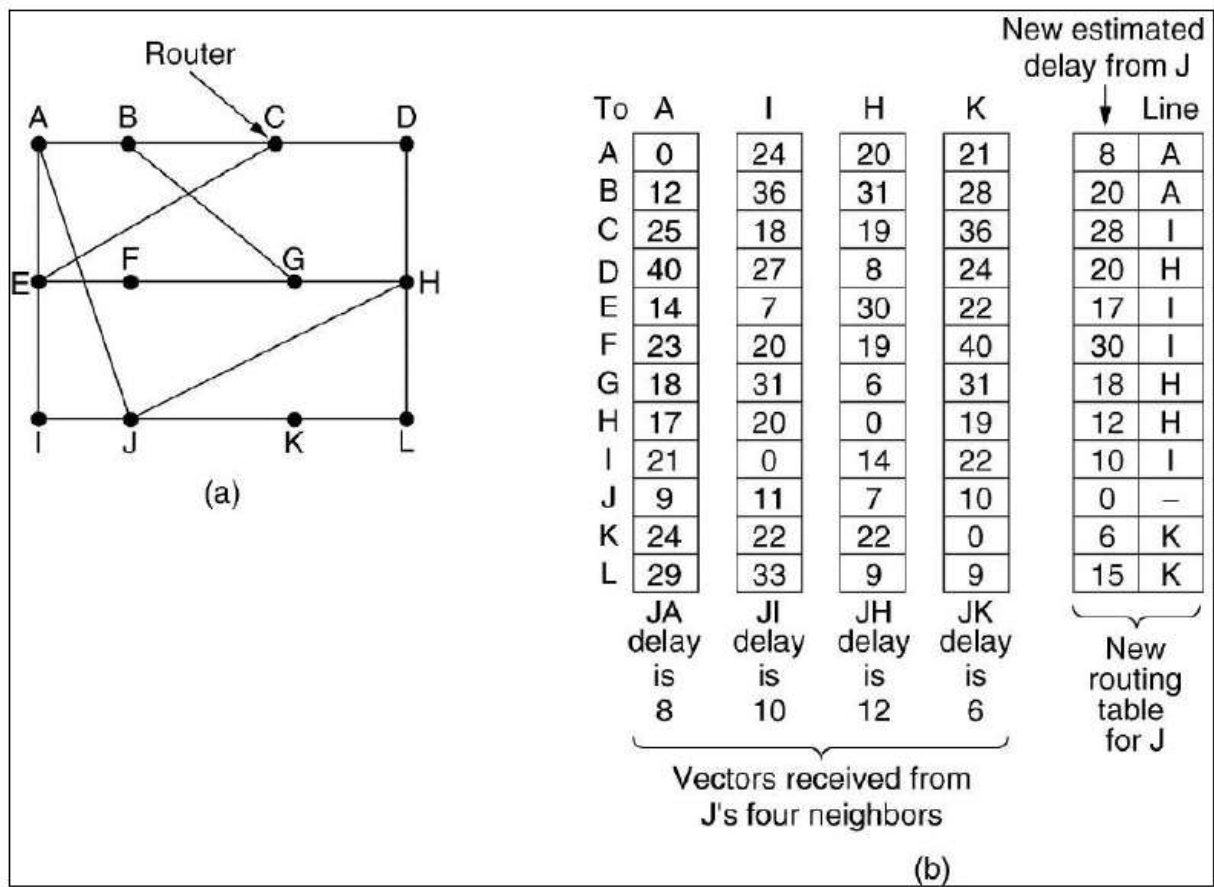


Figure (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

NS2 simulation using Distance Vector routing protocol

```
Set ns[new Simulator]
```

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

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

```
set tr [open out.tr w]
```

```
$ns trace-all $tr
```

```
proc finish {} {
```

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

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

```
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail
```

```
$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up
```

```
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
```

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
```

```
set udp [new Agent/UDP]
$ns attach-agent $n2 $udp
```


set cbr [new Application/Traffic/CBR]

\$cbr attach-agent \$udp

set null [new Agent/Null]

\$ns attach-agent \$n3 \$null

\$ns connect \$tcp \$sink

\$ns connect \$udp \$null

\$ns rtmodel-at 1.0 down \$n1 \$n3

\$ns rtmodel-at 2.0 up \$n1 \$n3

\$ns rtproto DV

\$ns at 0.0 "\$ftp start"

\$ns at 0.0 "\$cbr start"

\$ns at 5.0 "finish"

\$ns run

EXPERIMENT - 15

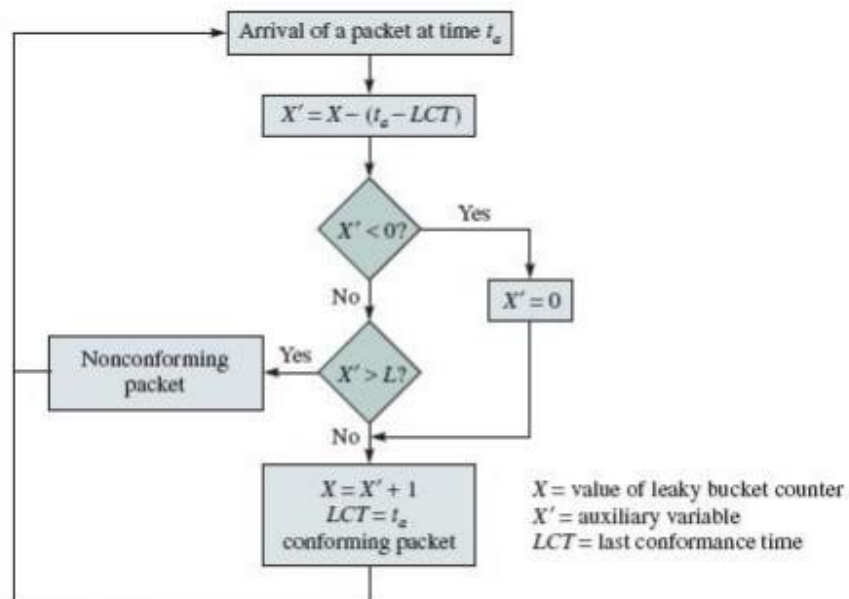
AIM: - Write a program for congestion control using Leaky Bucket algorithm

APPARATUS/PLATFORM USED: - Ubuntu, NS 2.34.

THEORY: - Network congestion in data networking and queueing theory is the reduced quality of service that occurs when a network node is carrying more data than it can handle. Typical effects include queueing delay, packet loss or the blocking of new connections. A consequence of the latter two effects is that an incremental increase in offered load leads either only to a small increase or even a decrease in network throughput.

Network protocols that use aggressive retransmissions to compensate for packet loss due to congestion can increase congestion, even after the initial load has been reduced to a level that would not normally have induced network congestion. Such networks exhibit two stable states under the same level of load. The stable state with low throughput is known as congestive collapse. Networks use congestion control and congestion avoidance techniques to try to avoid congestion collapse. These include exponential back-off in protocols such as 802.11CSMA/CA and the original Ethernet, window reduction in TCP, and fair queueing in devices such as routers. Another method is to implement priority schemes, transmitting some packets with higher priority than others. Priority schemes help to alleviate the effects of congestion for services such as 802.1p.

The Leaky Bucket Algorithm is based on, and gets its name from, analogy of a bucket that has a hole in the bottom through which any water it contains will leak away at a constant rate, until or unless it is empty. Water can be added intermittently, i.e. in bursts, but if too much is added at once, or it is added at too high an average rate, the water will exceed the capacity of the bucket, which will overflow. Hence, this leaky bucket determines whether adding some amount of water would exceed or conform to a limit on the average rate at which water can be added, set by the leak rate, and a limit on how much water can be added in a burst, set by the depth of the bucket. The leaky bucket algorithm is used in packet switched computer networks and telecommunications.



Leaky bucket algorithm.

PROGRAM OUTLINES: -

- 1). Start with include and define buffer size. i.e. 10.
- 2). By using the struct and printf command design the leaky bucket algorithm by using proper for loop.

Code: -

```

#include<stdio.h>

#include<stdlib.h>

#define NOF_PACKETS 10

Int rand(int a)

{

Int rn=(random() %10)%a;

Return rn==0?1:rn;

}
  
```

```
Int main()

#include<stdio.h>for(i=0;i<NOF_PACKETS;++i)

Packet_sz[i]=rand(6)*10;

For(i=0;NOF_PACKETS;++i)

Printf("\npacket[%d bytes\t",I,packet_sz[i]);

Printf("\nEnter the output rate:");

Scanf("%d",&o_rate);

Printf("Enter the Bucket Size:");

Scanf("%d",&b_size);

For (i=0;i<NOF_PACKETS;++i)

If ((packet_sz[i] +p_sz_rm)>b_size)

If(packet_sz[i]>b_size)/*compare the packet size with bucket size.

Printf("\nIncoming packet size (%dbtes)is Greater than bucket capacity(%dbytes)-PACKET

REJECTED ",packet_sz[i],b_size);

Else

Printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!);

Else

Capacity exceeded-PACKETS REJECTED!!");

Else

{
```

```
P_sz_rm+=packet_sz[i];

Printf("\n\nIncoming Packet size:%d,Packet_sz[i];

Printf("\nBytes remaining to Transmit: %d",p_sz_rm);

P_time=rand(4)*10;

Printf("\nTime left for transmission:%d units",p_time);

For(clk=10;clk<=p_time;clk+=10)

{

Sleep(1);

If (p_sz_rm)

{ if (p_sz_rm<=o_rate)/packet size remaining comparing with output rate*/

Op=p_sz_rm,p_sz_rm=0;

Else

Op=o_rate,p_sz_rm=o_rate;

Printf("\nPacket of size %d Transmitted",op);

Printf("-----Bytes remaining to transmit: %d",P_sz_rm);

}

else

Printf("\nTime left for transmission :%d units",p_time-clk);

Printf("\nNo packet to transmit!!);

} } } }
```