CHAPTER 3

SIMULATION SOFTWARE MINUTIAE

While computers are still away from the supreme ideal of being able to think for themselves as artificially intelligent machines, the brute force developed as a result of its conjugation with software, has enhanced their ability to calculate and process information at phenomenal speeds. This ability powers research in every field of human endeavour. Software plays an important role in various research fields. In fact, appropriate software has ability to change the way in which information, or more precisely, knowledge is used in a particular research. The decision to use particular software to aid analysis in a particular research is generally influenced by a number of factors, such as the field of research and the researcher's preferred approach to data analysis which has as its basis certain epistemological and ontological assumptions. A plethora of software types is available to be used at various stages of research. For example, system software such as Microsoft Windows, Linux-Fedora, Mac OSX; application software such as Microsoft Word, Excel, PowerPoint; and simulator software suiting a particular application. This chapter highlights the importance of simulation software, its role in general and the minutiae of various simulation software used for network securities, with special reference to NS-2.
3.1. SIMULATION IN GENERAL

Simulation plays an important part in modern technology. It has constantly been applied in various fields of science and engineering for diverse uses. Simulation is used to study and predict behaviour of complex set ups so as to encourage research, which otherwise would not have been feasible due to infrastructure or cost restrictions. Computer aided simulation may be used to model theoretical and practical objects as well as processes on a computer which helps in their study and analysis. Computer simulation can be used to assist the modelling and analysis in many areas of sciences viz. physics, chemistry, biology, biotechnology, etc. and even in finance and social science. From technological point of view, the application areas are civil engineering, electrical engineering, mechanical engineering and computer engineering etc. Presently application of simulation technology into networking area such as network traffic simulation and simulation of different network types is relatively novel.

3.2. NETWORK SIMULATOR AND ITS ROLE IN RESEARCH

Network simulator is a subset of simulator programs that are used to predict the behaviour of a network, without installing the actual network. Thus this piece of software imitates the working of a computer network. Network simulators are used to model real world networks so that its features can be changed and results be analyzed. Using this software, users can customize their network simulation in accordance to specific needs for analysis. Since the process of model modification is relatively cheap using a network simulator, therefore, a wide variety of scenarios can be analyzed at low cost. In these simulators, various computer networks are modelled with devices, traffic etc. and their performance can be analyzed.
3.2.1. General View

In general, a network simulation, more specifically a computer-assisted network simulation may be visualized as a software program that helps in creating a model to study the behavioral aspect of networks. It is applied in simulation of various networking algorithms or systems by using software engineering. The model is built either by using some network entities using certain mathematical formulas or alternatively it may be based on capturing certain parameters of the network physically. The model thus developed is then used to analyze various aspects of the network, thereby allowing different modifications and study of networks under various conditions.

Network simulator may vary from a simple to a complex form. However, minimally a network simulator must have the capability to represent a network, different types of links, define types of traffic among them. A more complicated system may allow the user to specify details about protocols, provide scalability, call different subroutines, provide a visual animation of simulated network and utilize the resources from other software too. Some of notable network simulation software implemented by various researchers includes NS-2, OPNET, NetSim. Generally text-based applications may provide a less user-friendly but allow more advanced forms of customization. Network simulators may be classified as commercial software and open-source software. A commercial network simulation is one in which the source code of its software or the affiliated packages are made available by paying some reasonable cost. It means that the users of this class of software have to pay to get authorization rights for the license so as to use this software or order specific packages according to their specific usage requirements. One of common commercial network simulator is the OPNET.

On the contrary, the open source network simulator is very open and any individual or an organization can contribute to it and find bugs in it.
Generally in open source software the interface is also open for further improvement. Since network technologies are developing at a very fast pace, different organizations participating in the process always require open platforms which should be scalable enough to include different efforts and different packages in the simulations of the whole network. Thus the network simulation tools must have this feature to be able to incorporate latest technology, allow different future new packages to be included and run transparently without harming existing components or packages. Thus the negative impact of some packages will have no or little impact to the other modules or packages. Open source network simulators are very flexible and reflect the latest developments of new technologies in a faster way than commercial network simulators [185-186]. Although the open source network simulators lack systematic and complete documentation, which makes it use difficult, still it offers numbers of advantages such as scalability and allowing tailor-made solutions.

3.2.2. Role of Network Simulators in research

Since research comprises various issues not limiting to trade, education, security, scalability and a wide range of cross-cutting issues, network simulators serve a variety of needs. In the field of networking mistakes can cost a lot of money and time to identify the faults and to rectify them. Compared to the cost and time involved in setting up an entire test bed containing multiple networked computers, routers and data links, network simulators are relatively fast and inexpensive. Thus network simulators are used by people from different areas such as academic researchers, industrial developers, and quality assurance personnel to design, simulate, verify, and analyze the performance of different networks protocols. They are also used to evaluate the effect of the different parameters on the protocols being studied.
A network simulator comprises of a wide range of networking technologies, protocols and basic building blocks which enable a user to build complex networks in a controlled and reproducible environment. A user can design different topologies employing building blocks such as nodes, hubs, routers, switches etc. This enables the user to create test scenarios which are difficult and in some cases practically not viable to create the test patterns and their evaluation. Expensive networks which are difficult to emulate using actual hardware may be easily created using network simulators. Network simulation tools can be used to address various issues such as high cost of monitoring equipment, maintaining the appropriate levels of hardware and software, having an appropriate mix of equipment available and ability to scale lab facilities. With this tool a researcher can create number of complex networks and may compare working or network parameters without involving much time, thereby proving it an economical too. A typical network simulator can simulate various types of Local Area Network (LAN) and Wide Area Network (WAN) technologies like Ethernet, token rings, TCP, ATM, IP etc. so as to allow testing and analysis of various standard protocols and strategies as well as devising novel protocols. For students and learners, a network simulator is an important tool as it allows them to explore a wider range of network alternatives than it can support in a laboratory. Not only this, simulators help design of hierarchical networks using fundamental types of nodes like computers, hubs, bridges, routers, switches, links, mobile units etc. The simulation results may vary from practical results obtained from actual network, but, they are close enough to give a meaningful insight into functioning of a network and variations in its parameters.

Performance of a network (wired or wireless) can be analyzed by three methods:

1. Analytical Modelling: This involves mathematical models which involve certain calculations, tools like calculus, queuing networks, probability etc.
2. Computer Assisted Simulation: In this technique the physical behaviour of the network is realized using probability, statistics and queuing theory.
3. Real Time Physical Tests: In this process actual network is created and different parameters are analyzed. These tests are tedious and could be applied to small networks only. However, for a network that includes hundreds of nodes, real time physical testing becomes expensive as well as cumbersome.

Thus Computer Assisted Simulation is the only feasible approach for the quantitative analysis of large networks, which can be classified into two types: Discrete Event Simulation or Continuous Simulation. The Discrete Event Simulation involves discrete representation of time so that it is quantized and the state of the system changes only at a particular event. On the other hand, a continuous simulation involves continuous progression of time during simulation. Computer Simulations are generally Discrete Event Simulations. There are several popular network simulators: Network Simulator - 2 (NS-2), QualNet, NetSim, GloMoSim, and OPNET. Network Simulator – 2 is the most popular one [187].

3.2.3. Literature study of network simulators in research

The field of wireless networks also depends on the use of certain simulation software for its analysis and research. A number of software may be employed for performance evaluation and analysis [179]. During the course of literature study some indications on appropriate software for carrying out our research were found.

NS-2 has been studied as a tool for analysis of various schemes and comparison between different existing protocols [100, 136, 145, 148, 151, 153, 160, 162, 165, 179-184].

It has been utilised to fix a flaw in the disassociation mechanism of the WPA protocol, to present an analysis on different routing protocols employed in wireless networks and to present observations regarding the
behaviour of these protocols [100, 145, 148, 151, 153, 160, 164, 184]. Majumder. K et al. have carried out a simulation based performance analysis of DSDV and DSR protocols in the hybrid networking environment using NS-2 [160, 164].

NS-2 is utilised in a security scheme, CDM which, attempts to fix a flaw in the disassociation mechanism of the WPA protocol [100]. Enhanced-AODV (E-AODV) protocol has been covered during the literature study, which addresses the problem in MANETs and uses Network Simulator-2 (NS-2) [148]. NS-2 has been used to present an analysis on routing protocols like AODV, DSR, and DSDV employed in these wireless networks [145].

NS-2 network simulator has been used to first implement an evolving graph based routing protocol, and then to use it as a benchmark while comparing, some popular ad hoc routing protocols viz. AODV, DSR, OLSR and DSDV [180]. The development process models implemented in Network Simulator for point multi point topologies, design of beyond 3G networks, set of libraries written for NS-2 called Multi InteRfAce Cross Layer Extension (MIRACLE) have also been covered [181-182]. Underwater networking and its protocol design has also been studied [183]. The tool used for the design and testing of this protocol is also Network Simulator. Harris A. F et al have presented the design and implementation of an interface and a channel model for underwater acoustic networks in the NS-2 network simulator.

Apart, several other areas of networking have been studied, such as, the IEEE 1609.4 Standard for Wireless Access in Vehicular Environments (WAVE) that has been utilized to enhance the performance of vehicular networks with multi-channel operations. In this study implementation of the IEEE 1609.4 protocol and an evaluation study of 1609.4-based VANETs is done using the NS-2 simulator [136].
Wireless networking applications in real-time control systems have also been studied [125]. In this study two applications for Wireless Networked Control Systems (WiNCS) are covered and comparison of various performance criteria, both from networking as well as control point of view are observed. For this purpose, an emulation tool called PiccSIM, abbreviated from Platform for integrated communications and control design, simulation, implementation and modelling, has been employed. PiccSIM is a design and evaluation platform for networked control systems, where different networks can be emulated using the real-time extension of Network Simulator. Use of various other software such as QualNet, Glomosim, LanTraffic etc. have also been studied, typically in the research area of wireless protocols but their use was limited for certain applications only [151].

Qualnet also been used along with NS-2 to carry out the simulations, which are utilized to present observations regarding the behaviour of protocols used in large-scale mobile ad hoc networks (MANETs) [151]. A simulation experiment has also been studied which is performed by using Glomosim simulator to study the effects of three types of attacks, viz; Routing Table Poisoning Attack, Routing Loop Attack and Misrouting Attack on AODV protocol [134]. LanTraffic as network monitoring tool has been studied which may be utilized to analyze the effect of encryption techniques (WEP-40 and WEP-104) on throughput in IEEE 802.11g for TCP and UDP protocols [85].

3.3. NS-2 AS A NETWORK SIMULATOR

NS-2 is a discrete event packet level simulator targeted at wired and wireless networking research. It was developed at University of California, Berkeley to analyze the performance of networks. NS-2 is an object oriented simulator written in C++ with an OTcl interpreter as the front end and is a widely used tool to simulate the behavior of wired and wireless networks. It
supports a class hierarchy in C++ (also known as compiled hierarchy) and a similar class hierarchy in OTcl interpreter (also known as interpreted hierarchy). The two hierarchies are closely related. Network Simulator 2 (NS-2) is one of the widely used simulators by the researchers in computer communication networks to simulate and study network performance of new technologies for communication [188].

3.3.1 Development of NS-2

NS-2 is an open-source event-driven simulator designed specifically for exploring research in computer communication networks. Since its inception in 1989, NS-2 has been a simulation tool of keen interest for academia, industry and research. NS-2 is a result of substantial contributions from different experts in the field of networking. Among these are the University of California, Berkeley and Cornell University, Ithaca (New York) who developed the REAL (REalistic And Large) network simulator. REAL was originally implemented for study of the dynamic behaviour of flow and congestion control schemes in packet-switched data networks. It provides users with a way of specifying such networks and to simulate their behaviour. REAL is written in C and it provides around 30 modules that exactly emulate the actions of several well-known flow control protocols (such as TCP), and 5 research scheduling disciplines [189]. NS began as a variant of the REAL network simulator in 1989.

However, in 1995 development of NS was supported by Defense Advanced Research Projects Agency (DARPA) through the Virtual Inter Network Test-bed (VINT) project. The VINT project aimed at creating a network simulator that will initiate the study of different protocols for communication networking. As a result the first version of NS, known as NS-1, was developed at Lawrence Berkeley National Laboratory (LBNL) in the 1997 by Steve McCanne, Sally Floyd, Kevin Fall, and other contributors. This was also known as the LBNL Network Simulator, as it was derived
from the earlier simulator, REAL by S. Keshav [190]. The core of this simulator was written in C++, with Tcl-based scripting of simulation scenarios.

Since then NS has included substantial contributions from other researchers, including wireless code from the University of California, Berkeley-Daedelus and Carnegie Mellon University- Monarch projects and Sun Microsystems [191-192]. In 1996-97, work on NS version 2 (NS-2) was initiated based on a refactoring by Steve McCanne [193]. Now use of Tcl was replaced by MIT's Object Tcl (OTcl), an object-oriented dialect of Tcl. The core of NS-2 is also written in C++, but the C++ simulation objects are linked to shadow objects in OTcl and variables can be linked between both language realms. After the conclusion of the VINT project, NS-2 continued to be funded during the 2001-04 timeframe by the DARPA through Simulation Augmented by Measurement and Analysis for Networks (SAMAN-2001) project and through National Science Foundation (NSF) with Collaborative Simulation for Education and Research (CONSER-2002) project, both in collaboration with other researchers.

Due to its flexibility and modular nature, NS-2 has gained constant popularity in the networking research community over the past few years. Although, since it’s very beginning several revolutions and revisions have marked the growing maturity of the tool; having been under constant investigation and enhancement for years, currently NS-2 has evolved substantially and contains modules for numerous network components such as routing, transport layer protocol, application, etc. To investigate network performance, researchers can simply use an easy-to-use scripting language to configure a network, and observe results generated by NS-2. Undoubtedly, NS-2 has become the most widely used open source network simulator, and one of the most widely used network simulators [194].
3.4. ARCHITECTURE OF NETWORK SIMULATOR-2

The NS-2 architecture is composed of five parts, namely Event Scheduler, Network Components, TclCL, OTcl Library and Tool Command Language (Tcl) scripting language [187]. These components have been depicted in Fig. 3.1 and are discussed as follows:

3.4.1 Event Scheduler

The event scheduler runs by selecting the earliest event, executing it to completion and returning to execute the next event. There are presently four event schedulers available in the network simulator. Each scheduler is implemented using a different data structure: a simple linked list, heap, calendar (default) and real time.

3.4.2 Network Components

Network components include nodes, classifiers and links as described follows:-

Nodes: Node is a basic component in a network. In NS-2, node can be created using the following primitive:

```tcl
set ns [new Simulator] $ns node
```

The procedure “node” creates a node out of simple classifier objects. The node itself is a standalone class in OTcl and most of its components are separate TclObjects.

Classifiers: In NS-2 classifier are objects that are used to check at a specific portion of receiving packet so as to examine various packet-fields such as destination address, source address, etc. so as to send the packet to an outgoing interface. Several classifier objects, each looking at a specific portion of the packets forward the packet through the node are employed in NS-2, for example Address Classifier, Multicast Classifier, Multipath Classifier, Hash Classifier and Replicator.
**Link:** The class “Link” is used to provide point to point connectivity between two nodes. It is a standalone class in OTcl. The class “SimpleLink” provides a unidirectional link from one node to another. It is also known as a “simplex link”. Its syntax is:

```tcl
set ns [new Simulator]
$ns simplex-link <node0> <node1> <bandwidth> <delay> <queue_type>
```

Similarly class “DuplexLink” provides point to point connectivity between two nodes with a bi-directional link from one node to another and vice versa. Syntax of duplex-link is:

```tcl
set ns [new Simulator]
$ns duplex-link <node0> <node1> <bandwidth> <delay> <queue_type>
```

![Figure 3.1: Architecture of Network Simulator-2](image-url)
3.4.3 TclCL

The Tool command language with Classes (TclCL) is the TCL/C++ interface between OTcl and C++ as shown in Fig. 3.2.

![Diagram of TclCL]

**Figure 3.2: TclCL as an interface between OTcl and C++**

The code to interface OTcl interpreter and C++ resides in a separate directory called tclcl in ns-allinone package. Rest of the simulator code resides in a directory called ns-2.34. There are many classes defined in the directory tclcl. The main classes that are used in ns-2.34 are as mentioned below:

Class Tcl: It contains the methods that C++ code will use to access the OTcl interpreter.

Class TclObject: This is a base class for all simulator objects that are also mirrored in the compiled hierarchy.

Class TclClass: This class defines the interpreted class hierarchy and the methods to permit the user to instantiate TclObjects.

Class TclCommand: It defines simple global interpreter commands.
Class EmbeddedTcl: This class contains the methods to load higher level built in commands that simplify the procedure of configuring simulation environments.

Class InstVar: Various methods to access C++ member variables as OTcl instance variables are contained in this class.

### 3.4.4 OTcl Library

Object Oriented Tool command language (OTcl) is an extension to Tool Command Language/Tool Kit with object oriented programming. It was developed at Massachusetts Institute of Technology. Hence it is also known as MIT OTcl. OTcl library in NS-2 contains the implementation code of various components of NS-2 such as event scheduler and networks components.

### 3.4.5 Tool Command Language (Tcl) scripting language

Tcl is a scripting language created by John Ousterhout in 1988 at the University of California, Berkeley. It is a string-based command language. The language has only a few fundamental constructs and relatively little syntax, which makes it easy to learn. Tcl is designed to assemble various software building blocks into applications. Tcl is interpreted when the application runs. The interpreter makes it easy to build and refine an application in interactive manner. Tcl is a platform independent language which allows a fast development and is flexible for integration.

Various commands used to write simple Tcl programs are discussed under:

1. `set a 10`

`set` command is used to assign a value to a variable. For example the above command assigns the value 10 to variable ‘a’.
2. \texttt{set x $a}

\$ is employed when the assigned value is to be used. For example in the above command value assigned to ‘a’ is assigned to x.

3. \texttt{set x [expr $a - $b]}

\texttt{expr} command is used to execute any mathematical operation. For example the above command executes the difference of values assigned to variables ‘a’ and ‘b’ which is assigned to x.

4. \texttt{# command starts a comment and the tcl interpreter does not executes this line.}

An example program of tcl script is provided as under

Example 1: Simple program to calculate factorial of a number

\begin{verbatim}
1: proc factorial {a} {
2:     puts [exec clear]; # clear screen
3:     puts "This function will display factorial of numbers from 1 to $a";
4:     for {set k 1} {$k < $a} {incr k} {
5:         set m 1; # initialize m
6:         for {set l 1} {$l <= $k} {incr l} {
7:             set m [expr ($m * $l)]; # calculation
8:             puts " Factorial of $k is $m"; # display result
9:         } ;# loop closed
10:     } ;# first loop closed
11:     set a 15; # input
12:     factorial $a; # function call
\end{verbatim}

To execute the above script in NS-2 the tcl file is located in a folder. The file is named as factorial.tcl and is placed in the folder vinaypractice created in ns-2.34 folder of NS-2 as shown in Fig. 3.3.
This is executed using the ./ns command for which steps shown in Fig. 3.4 may be carried out and the result is as shown in the same figure:
Example 2: Program to create NAM and a trace file.

```
1: set val(chan)          Channel/WirelessChannel ;#Channel Type
2: set val(prop)          Propagation/TwoRayGround ;# radio-propagation model
3: set val(netif)         Phy/WirelessPhy ;# network interface type
4: set val(mac)           Mac/802_11 ;# MAC type
5: set val(ifq)           Queue/DropTail/PriQueue ;# interface queue type
6: set val(ll)            LL ;# link layer type
7: set val(ant)           Antenna/OmniAntenna ;# antenna model
8: set val(ifqlen)        50 ;# max packet in ifq
9: set val(nn)            2 ;# number of mobilenodes
10: set val(rp)           DSDV ;# routing protocol
11: #set val(rp)           DSR ;# routing protocol
12: set val(x)         500
13: set val(y)         500

14: # Initialize Global Variables
15: set ns_        [new Simulator]
16: set tracefd     [open wireless_mitf.tr w]
17: $ns_ trace-all $tracefd
18: $ns_ namtrace [open wireless_mitf.nam w]
19: $ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
20: # set up topography object
21: set topo       [new Topography]
22: $topo load_flatgrid $val(x) $val(y)
23: # Create God
24: create-god $val(nn)
25: # New API to config node:
26: # 1. Create channel (or multiple-channels);
27: # 2. Specify channel in node-config (instead of channelType);
28: # 3. Create nodes for simulations.
29: # Create channel #1 and #2
30: set chan_1_ [new $val(chan)]
31: set chan_2_ [new $val(chan)]
32: # Create node(0) "attached" to channel #1
33: # configure node, please note the change below.
34: $ns_ node-config -adhocRouting $val(rp) \\
35: -llType $val(ll) \\
36: -macType $val(mac) \\
37: -ifqType $val(ifq) \\
38: -ifqlen $val(ifqlen) \\
39: -antType $val(ant) \\
40: -propType $val(prop) \\
41: -phyType $val(netif) \\
42: -topoInstance $topo \\
43: -agentTrace ON \\
44: -routerTrace ON \\
45: -macTrace ON \\
```
Simulation Software Minutiae

55:     -movementTrace OFF \
56:     -channel $chan_1_
57:
58:     set node_0 [$ns_node]
59:
60:     # node_1 can also be created with the same configuration, or with a different
61:     # channel specified.
62:     # Uncomment below two lines will create node_1 with a different channel.
63:     # $ns_node-config \
64:     #   -channel $chan_2_
65:     set node_1 [$ns_node]
66:
67:     $node_0 random-motion 0
68:     $node_1 random-motion 0
69:
70:     for {set i 0} {$i < $val(nn)} {incr i} {
71:       $ns_initial_node_pos $node_($i) 20
72:     }
73:
74:     #
75:     # Provide initial (X,Y, for now Z=0) co-ordinates for mobilnodes
76:     #
77:     $node_0 set X_ 5.0
78:     $node_0 set Y_ 2.0
79:     $node_0 set Z_ 0.0
80:
81:     $node_1 set X_ 8.0
82:     $node_1 set Y_ 5.0
83:     $node_1 set Z_ 0.0
84:
85:     #
86:     # Now produce some simple node movements
87:     # Node_1 starts to move towards node_0
88:     #
89:     $ns at 3.0 "$node_1 setdest 50.0 40.0 25.0"
90:     $ns at 3.0 "$node_0 setdest 48.0 38.0 5.0"
91:
92:     # Node_1 then starts to move away from node_0
93:     $ns at 20.0 "$node_1 setdest 490.0 480.0 30.0"
94:
95:     # Setup traffic flow between nodes
96:     # TCP connections between node_0 and node_1
97:
98:     set tcp [new Agent/TCP]
99:     $tcp set class_ 2
100:    set sink [new Agent/TCPSink]
101:    $ns attach-agent $node_0 $tcp
102:    $ns attach-agent $node_1 $sink
103:    $ns connect $tcp $sink
104:    set ftp [new Application/FTP]
105:    $ftp attach-agent $tcp
106:    $ns at 3.0 "$ftp start"
107:
108:     #
109:     # Tell nodes when the simulation ends
The above program when run in NS-2 creates number of files from which information of different types can be extracted using some useful NS-2 tools which are discussed as under.

### 3.5. USEFUL NS-2 TOOLS

There are several tools which are used by the NS-2 simulator for animating or plotting the results generated as a result of a simulation which support precise observation and effective analysis. These include NAM, Xgraph, AWK files.

#### 3.5.1 NAM

NAM is a Tcl/TK based animation tool for viewing network simulation traces and real world packet traces. It supports topology layout, packet level animation, and various data inspection tools. NAM began at Lawrence Berkley National Laboratory [190]. It has evolved substantially over the past few years. The NAM development effort was an ongoing collaboration with the VINT project. Currently, it is being developed as an open source project.

When program in Example 2 is run it creates a wireless_mitf.nam file which could be used to visualize the how does the network simulated in the
program runs. However to start NAM certain commands are required which are given as in Fig. 3.5 below.

![Figure 3.5: Execution of NAM for visualization](image)

The steps in above figure open the NAM window as shown in Fig. 3.6. Clicking on file and choosing the correct location helps finding the file wireless-mitf.nam as shown in Fig. 3.6.

![Figure 3.6: Location of NAM file](image)

Clicking on open when the file wireless_mitf.nam appears in “File name:” opens the animation of the simulated network. The snapshots of the simulation in action are provided in Fig. 3.7(a) and Fig. 3.7(b).
3.5.2 Xgraph

Xgraph was introduced by MPagan in 1985 [195] Enhanced by CHein, ARubin, TRegovich, and others, and continues to be used and supported by
several organizations and projects. Xgraph is a general purpose x-y data plotter with interactive buttons for panning, zooming, printing, and selecting display options. It can be used to plot data from any number of files on the same graph and can handle unlimited data-set sizes and any number of data files.

The Xgraph program draws a graph on an X display for the given data which is read either from data files or from some standard input, if no files are specified. It can display up to 64 independent data sets using different colours and/or line styles for each set. It annotates the graph with a title, axis labels, grid lines, tick marks, grid labels and legend. There are options to control the appearance of several components of the graph. A data set consists of an ordered list of points of the form “directive X Y”. For directive “draw”, a line will be drawn between the previous point and the current point. Specifying a “move” directive tells Xgraph not to draw a line between the points. “draw” is the default directive. The name of a data set can be specified by enclosing the name in double quotes.

Xgraph also presents three control buttons in the upper left corner of each window: Hardcopy, Close and About as shown in Fig. 3.8. Xgraph accepts a large number of options most of which can be specified either on the command line, in the user’s .Xdefaults or .Xresources file; or in the data files themselves.

Xgraph produces wysiwyg PostScript, PDF, PPTX and ODP output for printing hard-copies, storing, sharing plotted results and importing graphs directly into word-processors for creating documentation, reports, and viewgraphs.

Xgraph includes the ability to specify plotting colours for multi-colour plots, as well as line-thickness. It has the ability to use any column of a multi-column file as ordinate and abscissa axis. It also supports automatic
resizing of its window and interactively zooming into any region of a graph by simply dragging a box around the region with a mouse.

Xgraph for the sample program 2 is provided in Fig. 3.8. This graph utilizes the AWK script to calculate the throughput and the syntax for using this command is

./xgraph file_name.xgr

Figure 3.8: Xgraph to calculate throughput for sample program 2

:bg <colour> (Background): This command specifies background colour of the Xgraph window.

The syntax for using this command is

./xgraph –bg white file_name.xgr

The xgraph generated using this specification is shown in Fig. 3.9 below
Figure 3.9: Xgraph with white background for sample program 2

-lw width (LineWidth) : This command specifies the width of the data lines in pixels. The default is zero. The syntax for using this command is

./xgraph –lw 3 file_name.xgr

The xgraph generated using this specification is shown in Fig. 3.10 below

Figure 3.10: Xgraph with thick line width for sample program 2
Multiple specifications can be executed simultaneously in the `xgraph` command. For example, the background changed to white and the line width changed to three can be executed in the same command as follows:

```
./xgraph –bg white –lw 3 file_name.xgr
```

The resultant XGraph generated is shown in Fig. 3.11.

![Figure 3.11: Xgraph with white background as well thick line width for sample program 2](image)

`-zg <colour>` (Zero Colour): This is the colour used to draw the zero grid line. For example, when the colour of the zero line is desired as green, it can be executed using the following command and the corresponding XGraph is shown in Fig. 3.12.

```
./xgraph –zg green file_name.xgr
```
Figure 3.12: Xgraph with green colored zero grid for sample program 2

\texttt{zg <width>} (Zero Width): Specifies width, in pixels, of the zero grid line. The following command will execute the xgraph with a zero line of green colour having width 3 pixels as shown in Fig. 3.13.

\texttt{./xgraph \textasciitilde zg green \textasciitilde zw 3 file\textunderscore name.xgr}

Figure 3.13: Xgraph with thick green coloured zero grid for sample program 2
Units on X and Y axis can also be specified in a Xgraph.

\[ -x \text{ <unitname>} \text{(XUnitText): This is the unit name for the X axis. Its default is "X".} \]

\[ -y \text{ <unitname>} \text{(YUnitText): This is the unit name for the Y axis. Its default is "Y".} \]

For example if the unit of measurement on X axis is seconds then it can be implemented in xgraph using the following command as shown in Fig. 3.14.

\[ ./\text{xgraph} \ -x \text{ SECONDS file\_name.xgr} \]

---

**Figure 3.14: Xgraph with units on X axis for sample program 2**

\[ -t \text{ <string>} \text{(TitleText): Title of the plot. This string is centred at the top of the graph. Fig. 3.15 shows the Xgraph when following example command is run in NS-2.} \]

\[ ./\text{xgraph} \ -t \text{ ‘THIS IS THE XGRAPH PLOT FOR SAMPLE PROGRAM’ file\_name.xgr} \]
Figure 3.15: Xgraph with title text for sample program 2

–fg <colour> (Foreground Colour)

This command specifies the colour used to draw all text and the normal grid lines in the window. It should be used with a precaution that white foreground should not be used on a white background. For example, to obtain a white foreground when background is black following command may be used:-

./xgraph –fg white –bg black file_name.xgr

The Xgraph would appear to be as shown in Fig. 3.16.
Figure 3.16: Xgraph with white foreground and black background for sample program 2

Any other colour other than white can also be specified. For example yellow foreground on black background can be plotted using the command ./xgraph –fg yellow –bg black file_name.xgr.

The resultant Xgraph now changes as shown in Fig. 3.17.

Figure 3.17: Xgraph with yellow foreground and black background for sample program 2
Likewise number of other options can be used with Xgraph such as:-

-bar (BarGraph): Specifies that vertical bars should be drawn from the data points to a base point which can be specified with -brb.

Usually, the -nl flag is used with this option. The point itself is located at the center of the bar.

-brw <width> (BarWidth): This specifies the width of bars in a bar graph. The amount is specified in the user’s units. By default, a bar one pixel wide is drawn.

-lnx (LogX): Specifies a logarithmic X axis. In this command the grid labels represent powers of ten.

-lny (LogY): Specifies a logarithmic Y axis. For this command also, the grid labels represent powers of ten.

3.5.3 AWK

The AWK utility is an interpreted programming language typically used as a data extraction and reporting tool. AWK is a programming language designed to process text files. AWK can be used in NS-2 to process NS-2 trace files.

This programming language was created at Bell Labs in the 1970s, to do simple data processing for ordinary data generated on a day-to-day basis. The name AWK is derived from the family names of its authors – Alfred Aho, Peter Weinberger, and Brian Kernighan. AWK (pronounced as ‘auk’; the name of the bird which acts as an emblem of the language) is a language based on the principle of pattern-action processing. When written in all lowercase letters, as awk, it refers to the Unix program that runs scripts written in the AWK programming language [196]. In 1985 authors of AWK expanded the language, by adding user-defined functions. The language is described in the book “The AWK Programming Language”, published in 1988, and its implementation was made available in releases of UNIX System. To avoid confusion with the incompatible older version, this version
was sometimes called "new awk" or nawk. This implementation was released under a free software license in 1996, and is still maintained by Brian Kernighan. AWK script can be successfully employed to process NS-2 trace files to produce results and graphs along with Xgraph utility [197-198].

Program 3: AWK file to generate throughput

```awk
BEGIN {
    recvdSize = 0
    startTime = 400
    stopTime = 0
}

{ 
    event = $1
    time = $2
    node_id = $3
    pkt_size = $8
    level = $4

    # Store start time
    if (level == "AGT" && event == "s" && pkt_size >= 512) {
        if (time < startTime) {
            startTime = time
        }
    }

    # Update total received packets' size and store packets arrival time
    if (level == "AGT" && event == "r" && pkt_size >= 512) {
        if (time > stopTime) {
```
Simulation Software Minutiae

Program 4: Awk script to generate packet delivery fran

```
BEGIN {
    sendLine = 0;
    recvLine = 0;
    fowardLine = 0;
}

$0 ~/^s.* AGT/ { sendLine ++ ; }
$0 ~/^r.* AGT/ { recvLine ++ ; }
$0 ~/^f.* RTR/ { fowardLine ++ ; }

END {
    printf "cbr s:%d r:%d, r/s Ratio:%.4f, f:%d \n", sendLine, recvLine, (recvLine/sendLine),fowardLine;
}
```
Program 5: AWK Script for calculating the Send, Received, Dropped Packets, Received Packets, Packet Delivery Ratio and Average end to End Delay

1: BEGIN {
2:  seqno = -1;
3:  droppedPackets = 0;
4:  receivedPackets = 0;
5:  count = 0;
6: }
7: {
8:  #packet delivery ratio
9:  if($(4 == "AGT" && $1 == "s" && seqno < $6)) {
10:     seqno = $6;
11: } else if(($4 == "AGT") && ($1 == "$s")) {
12:     receivedPackets++;
13: } else if ($(1 == "D" && $7 == "tcp" && $8 > 512) {
14:     droppedPackets++;
15: }
16:  #end-to-end delay
17:  if($(4 == "AGT" && $1 == "s") { 
18:     start_time[$6] = $2;
19: } else if(($7 == "tcp") && ($1 == "$s") { 
20:     end_time[$6] = $2;
21: } else if($1 == "D" && $7 == "tcp") {
23: }
24: }
25: }
26: END {
27: for(i=0; i<=seqno; i++) {
28:     if(end_time[i] > ; 0) {
29:         delay[i] = end_time[i] - start_time[i];
30:         count++;
31:     } else {
32:         delay[i] = -1;
33:     }
34: }
35: }
36: }
37: for(i=0; i<count; i++) {
38:     if(delay[i] > 0) {
39:         n_to_n_delay = n_to_n_delay + delay[i];
40:     }
41: }
42: n_to_n_delay = n_to_n_delay/count;
43: print "\n";
44: print "GeneratedPackets = " seqno+1;
45: print "ReceivedPackets = " receivedPackets;
46: print "Packet Delivery Ratio = " receivedPackets/(seqno+1)*100
47: "%";
48: print "Total Dropped Packets = " droppedPackets;
49: print "Average End-to-End Delay = " n_to_n_delay * 1000 " ms";
50: print "\n";
51: }