UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

TELECOMMUNICATIONS ENGINEERING

Xiang Chao

WIRELESS NETWORK STUDY AND ANALYSIS USING NS2 SIMULATOR

Master's thesis for the degree of Master of Science in Technology submitted for inspection in Vaasa, 14th of May, 2008.

Supervisor D.Sc. (Tech.) Mohammed Salem Elmusrati

Instructor D.Sc. (Tech.) Mohammed Salem Elmusrati

TABLE OF CONTENTS

ABBREVIATIONS AND SYMBOLS

AODV	Ad-hoc On-demand Distance Vector
API	Application Programming Interface
ARP	Address Resolution Protocol
ARQ	Automatic Repeat-Request
CBR	Constant Bit Rate
CONSER	Collaborative Simulation for Education and Research
DARPA	Defense Advanced Research Projects Agency
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
ECN	Explicit Congestion Notification
FTP	File Transfer Protocol
HUT	Helsinki University of Technology
ICSI	International Computer Science Institute
ICIR	ICSI Networking Group
ISI	Information Sciences Institute
LBL	Lawrence Berkeley National Laboratory
LMNR	Local Multiple Next Hop Routing Protocol
MAC	Media Access Control
MANET	Mobile Ad-hoc Network
NAM	Network Animator
NS2	Network Simulation Version 2
NSE	Network Simulation Emulation
OTCL	Object Oriented Extension of TCL
PARC	Palo Alto Research Center
RED	Random Early Detection
RERR	Route Error

RREP	Route Reply
RREQ	Route Request
SAMAN	Simulation Augmented by Measurement and Analysis for Network
TCL	Tool Command Language
TCLCL	TCL with Classes
TCP	Transmission Control Protocol
TEG	Telecommunication Engineering Group
TORA/IMPE	Temporally-Ordered Routing Algorithm
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
USC	University of South Carolina
UWB	Ultra-Wideband
VBR	Variable Bit Rate
VINT	Virtual Inter Network Testbed
WiNCS	Wireless Networked Control System
WSN	Wireless Sensor Network

SYMBOLS

P_t	Transmission Power
P_r	Received Power
G_t	Transmission Antenna Gain
G_r	Received Antenna Gain
h_t	Transmission Antenna Height
h_r	Received Antenna Height
λ	Wavelength
d	Distance
L	System Loss
χ	Lognormal Distribution

UNIVERSITY OF VAASA

Faculty of technology

Author: Xiang Chao

Topic of the Thesis: Wireless Network Study and Analysis using NS2 Simulator

Supervisor: Mohammed Salem Elmusrati

Instructor: Mohammed Salem Elmusrati

Degree: Master of Science in Technology

Department: Department of Computer Science

Degree Program: Degree Program in Information Technology

Major of Subject: Telecommunication Engineering

Year of Entering the University: 2006

Year of Completing the Thesis: 2008 **Pages:** 100

ABSTRACT:

NS2 (Network Simulation version 2) is a well-known generic network simulator. Unlike other expensive simulation software, it is free and based on open source. It is widely used to simulate and emulate communication networks. Furthermore, it has a rich library of network and protocol objects, which almost involve most of the aspects of network technology. This makes NS2 the most favorable simulation software which is widely used in academic research. On the other hand, the results of the simulation are validated by many research centers. For this reason many published articles about network technology show their results by using NS2 simulation. Additionally, act an excellent instruction tool NS2 is widely utilized in education. Nowadays, NS2 becomes more and more popular in scientific research and education.

Nevertheless, NS2 is quite difficult to handle for a beginner. Some reasons are: the content of NS2 is very huge; the official NS manual is not updated regularly and a lot of relative knowledge and tools are involved to operate NS2 efficiently.

NS2 will be one of the main tools in the research activities of the Telecommunication Engineering Group (TEG). Hence, the main target of this thesis is to study NS2 deeply and to show how to construct an emulation environment by using NS2 and MATLAB. Different simulators are given to demonstrate how to proceed with NS2. This thesis will be one reference for TEG researches for the applications of NS2.

KEYWORDS: NS2, Simulation, Integration, Emulation

1. INTRODUCTION

NS (version 2) is a discrete event simulator forcing on network research. NS2 provides substantial support for simulation of Transmission Control Protocol (TCP), routing, and multicast protocols over wired and wireless (local and satellite) networks. It can also implement such behaviors like File Transfer Protocol (FTP), Telnet, Web, Constant Bit Rate (CBR) and Variable Bit Rate (VBR), router queue management mechanism such as Drop Tail, Random Early Detection (RED) and Class-Based Queueing (CBQ), routing algorithms such as Dijkstra, and more (Chung & Claypool A 2003). The multicasting and some of the Media Access Control (MAC) layer protocols can also be simulated by NS2.

1.1. The History of NS2

NS development began in 1989 as a variant of the REAL network simulator. By 1995, NS has been supported by the Defense Advanced Research Projects Agency (DARPA), the Virtual Inter Network Testbed (VINT) project at Lawrence Berkeley National Laboratory (LBL), Palo Alto Research Center (PARC), University of California, Berkeley (UCB), and the Information Sciences Institute of the University of Southern California (USC/ISI) (NS2 wikipedia 2008).

NS is now developed in collaboration between a number of different researchers and institutes, including Simulation Augmented by Measurement and Analysis for Network (SAMAN), Collaborative Simulation for Education and Research (CONSER), and the ICSI Networking Group (ICIR). Long-running contributions have also come from Sun Microsystems and the UCB Daedelus and Carnegie Mellon University’s Monarch projects, cited by the NS homepage for wireless code additions. (NS2 wikipedia 2008.)

The latest version of NS2 is ns-2.33. For documentation on recent changes, see the NS Change History (Information Sciences Institute B 2006).

The development of the 3rd Generation of NS has begun development on July 1, 2006 and will take four years (NS2 wikipedia 2008).

1.2. Basic Operation Flow of Using NS2

NS2 is an Object-oriented Tool Command Language (OTCL) script interpreter that has a simulation event scheduler, network component object libraries and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object) as shown in Figure 1.1. (Chung & Claypool A 2003).

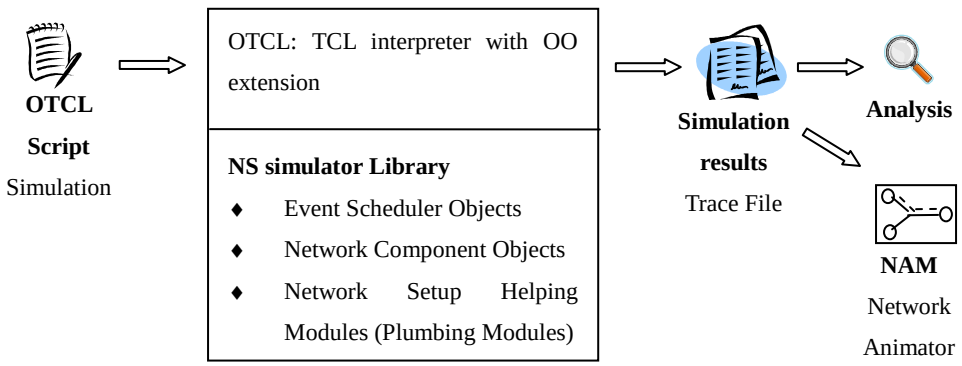

Figure 1.1: Simplified flow chart of using NS2 (Chung & Claypool A 2003.)

To use NS, the first step is to edit the program in OTCL script language. In order to setup and run a simulation network, a user should write an OTCL script that initiates an event scheduler, sets up the network topology by using the network objects and the

plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler. The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object (Chung & Claypool A 2003). This sounds like a complicated job, but the plumbing OTCL modules actually make the job very easy. The power of NS comes from this plumbing.

After the NS2 is running, a trace file is generated automatically, which contains the entire event schedule during the simulation. The trace file makes the result analysis of the simulation possible and the user can observe the entire communication process via the special tool called Network Animator (NAM).

The format of the trace file and the method to analyze it will be introduced later.

1.3. Assistant Tools in NS2

1.3.1. NAM

NAM began at Lawrence Berkeley National Laboratory (LBL). It has evolved vigorously over the past few years. The NAM development effort was an ongoing collaboration with the Virtual Inter Network Testbed (VINT) project. Currently, it is being developed at Information Sciences Institute (ISI) as part of the Simulation Augmented by Measurement and Analysis for Network (SAMAN) and Collaborative Simulation for Education and Research (CONSER) projects. (Buchheim 2002).

NAM is a Tool Command Language (TCL) based animation tool for viewing network simulation traces and real world packet trace (Buchheim 2002). The first step to use NAM is to produce the trace file. The trace file should contain topology information, e.g., nodes, links, as well as packet traces. Usually, the trace file is generated by NS2. During an NS simulation, a user can produce topology configurations, layout information, and packet traces using tracing events in NS.

When the trace file is generated, it 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 the time of the first packet in the trace file. Through its user interface, NAM provides control over many aspects of animation. (Buchheim 2002).

More information about NAM will be given later.

1.3.2. Trace File

The trace file format depends on the simulated network whether it is wired or wireless as explained next.

- **Wired Case**

After the simulation a trace file will be created to record the process of all the events during the simulation. The wired network trace file usually looks like Table 1:

Table 1: Model of trace file

state	time	From node	To node	type	size	flag	fid	Src addr	Dst addr	Seq num	id
+	.959779	2	3	tcp	1040	-------	1	0.0	3.0	63	379
r	1.959925	2	0	ack	40	-------	1	3.0	0.0	54	374
+	1.959925	0	2	tcp	1040	-------	1	0.0	3.0	64	382
-	1.959925	0	2	tcp	1040	-------	1	0.0	3.0	64	382
r	1.962	1	2	cbr	1000	-------	2	1.0	3.1	231	380
+	1.962	2	3	cbr	1000	-------	2	1.0	3.1	231	380
d	1.962	2	3	cbr	1000	-------	2	1.0	3.1	231	380

Now there are 7 trace entries in Table 1. It is clear that there are three enque operations mean join into the waiting queue list (indicated by “+” in the first column), one deque operations which mean leave from the waiting queue list (indicated by “-”), two receive events (indicated by “r”), and one drop event (indicated by “d”) (this had better be a trace fragment, or some packets would have just vanished!).

The simulated time (in seconds) at which each event occurred is listed in the second column. The third and fourth columns indicate between which two nodes the tracing happens. The fifth field is a descriptive name for the type of packet. The sixth field is the packet’s size, encoded in its IP header.

Characters from the seventh to the tenth field represent special flag bits which may be enabled. Presently only one such bit exists (explicit congestion notification, or ECN) (Harding 2005). In this example, Explicit Congestion Notification (ECN) is not used.

The next field gives the IP flow identifier field as defined for IP version 6.1. The two subsequent fields indicate the packet's source and destination node addresses, respectively. The following field indicates the sequence number. The last field is a unique packet identifier. Each new packet created in the simulation is assigned for a new, unique identifier.

For the first recode:

+	1.959779	2	3	tcp	1040	-------	1	0.0	3.0	63	379

It means it is a TCP packet whose size is 1040 bytes; deliver from node 2 to node 3 at time 1.959779(s). The TCP connection in this case is noted as field 1. The other data present that: the source address of the packet is 0.0 and the direction address of it is 3.0. The sequence number of the packet is 63 and the packet ID of it is 379. They are important data to analyze the simulation as well as to demo on NAM file.

- **Wireless Case**

In wireless case, the trace files have some different formats, the specific explain can be found in "~ns/trace/cmu-trace.cc", the instance shown below is an Ad-hoc On-demand Distance Vector (AODV) case:

s -t 0.001529932 -Hs 26 -Hd -2 -Ni 26 -Nx 585.08 -Ny 960.45 -Nz 0.00 -Ne 200.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 26.255 -Id -1.255 -It IMEP -Il 44 -If 0 -Ii 0 -Iv 1 -P aodvt -Pt 0x1 -Ph 1 -Pd 26 -Pds 2 -Pl 4.000000 -Pc HELLO
r -t 0.003927946 -Hs 1 -Hd -2 -Ni 1 -Nx 400.00 -Ny 200.00 -Nz 0.00 -Ne 199.999842 -Nl RTR -Nw --- -Ma 0 -Md 6000000 -Ms ffff0008 -Mt 0 -Is 6.255 -Id -1.255 -It IMEP -Il 44 -If 0 -Ii 0 -Iv 1 -P aodvt -Pt 0x1 -Ph 1 -Pd 6 -Pds 2 -Pl 4.000000 -Pc HELLO

(1) Event type: in the previous example, the first filed means the event type. There are four styles: s (send), r (receive), d (drop), f (forward).

(2) General flag: the second field is begun with “-t”, means the time of the event.

(3) Next hop information: this filed presents the information of the next hop, leaded by “-H”: -Hs (Hop source Node ID), -Hd (Hop destination Node ID).

(4) Node Property: this field denotes the Properties of the nodes, such as the node ID, trace level, leaded by “-N”. –Ni (Node ID), -Nx, -Ny, -Nz (Node Coordinates), -Ne (Node Energy Level), -Nl (Network trace Level: AGT, RTR, MAC, etc.), -Nw (drop reason).

(5) IP Level Packet Information: leaded by “-I”. -Is (source address, source port num), -Id (destination address, dest port number), -It (packet type), -If (flow ID), -Ii (unique ID), -Iv (TTL value).
(6)MAC Level Packet Information: leaded by “-M”. -Ma (duration), -Md (Destination Ethernet Address), -Ms (Source Ethernet Address), -Mt (Ethernet Type).

(7) Packet Specific Information: presents the types of the route protocol type. In AODV case this field is leaded by “-P aodv”. -Pt (type), -Ph(Hop Count), -Pb (Broadcast ID),-Pd (Destination) -Pds (Destination Sequence Number), -Ps (Source), -Pss (Source Sequence Number), -Pl (Lifetime), -Pc (Operation: REQUEST, REPLY, ERROR, HELLO).

There are many different types of trace files in the practical case such as: Address Resolution Protocol (ARP), CBR, TCP. The main difference of these trace file is in the field of “-P”. (Harding 2005.)

1.3.3. Xgraph and Gnuplot

Xgraph and Gnuplot are two plotter tools of NS2 used to show the results of the simulations.

Xgraph is a general purpose x-y data plotter, the operation of which is using the first column as the X-axis data, and Y-axis is decided by the second column then plot the graph. It will be discussed in more details in Chapter 4.

Gnuplot is one kind of command-driven interactive function plotting program. It is a program with a fairly long history, dating back to 1986 (Gnuplot 2008). Its function is to generate two or three-dimensional plots of data, which is utilized to analyze the data and functions.

Nowadays, Gnupolt is widely used on UNIX, Linux and Windows platform. The operation methods are almost similar on different platforms. Below the paper will introduce some simple examples of how to apply Gnuplot in Linux.

Run the command Gnuplot, shown as the Figure 1.2.

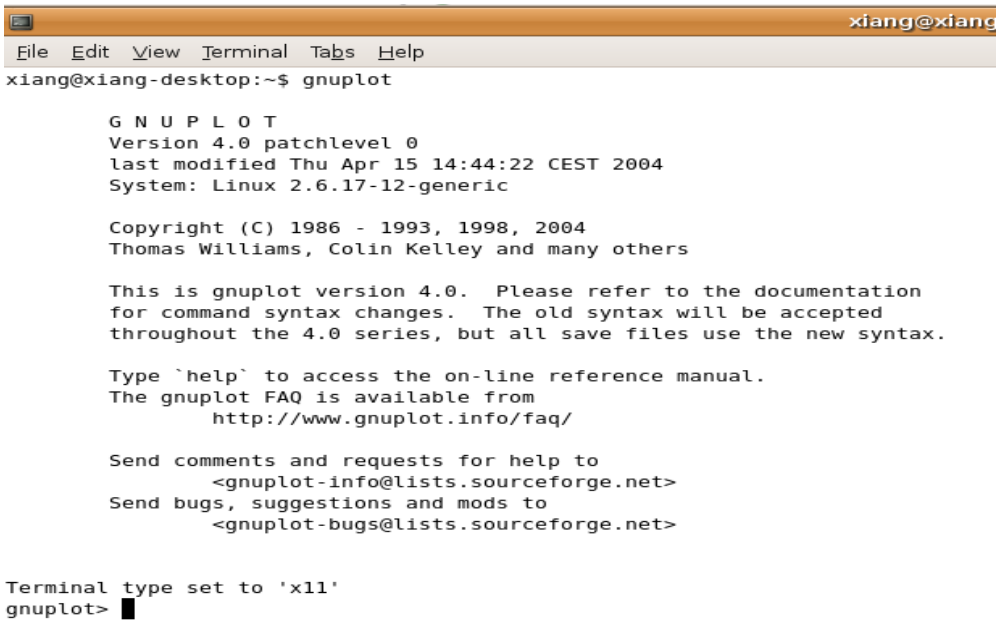

Figure 1.2: Interface of Gnupolt.

Type “plot sin(x)”, then obtain the graph show as the Figure 1.3.

:

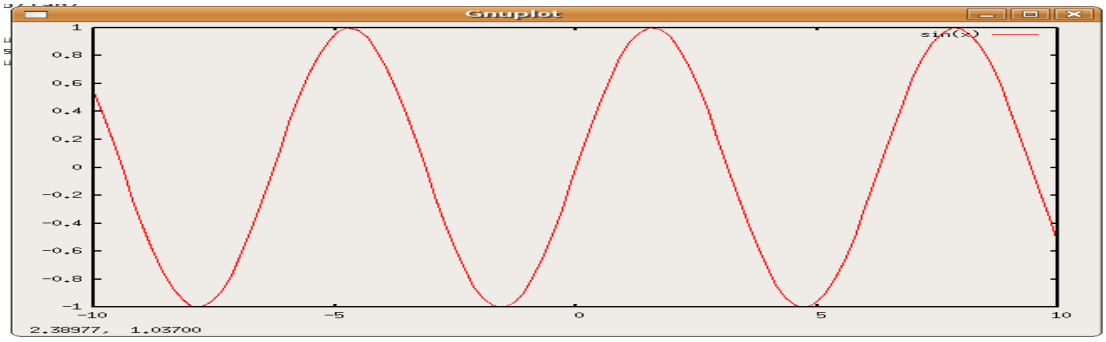

Figure 1.3: sin(x) plot in Gnupolt

To set the interval of x-axis, one can use the command: gnuplot>set xtics -10,1,10; gnuplot>plot sin(x). It means marking on x-axis from -10 to 10 and the unit of the marking is 1. It is the same method to reset y-axis.

Figure 1.4: Reset the interval of x-axis

The command “set grid” and “unset grid” used to set or cancel the grid in the x-y plane.

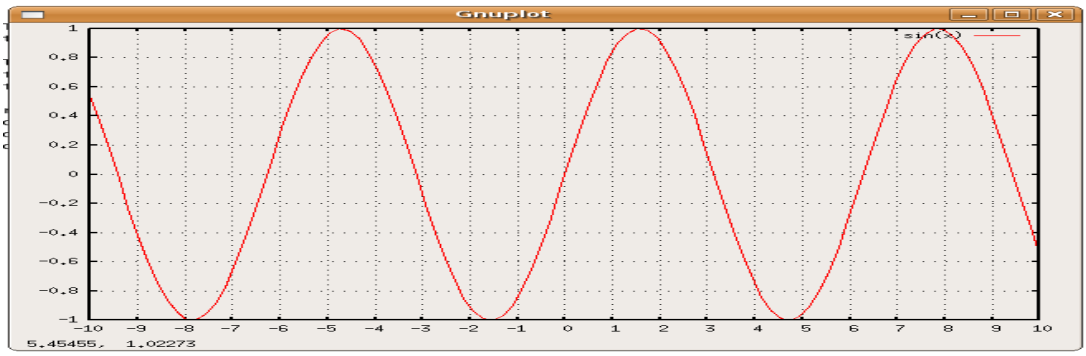

Figure 1.5: Set grid of the plot

Sometimes logarithms are necessary to analyze the results. The command of setting the coordinates system transformation is: set logscale <axis> <base> (axes can be x,y,z or combination of them; default base is 10).

For some complicated case, a substituted software TraceGraph is recommended. It is easy to operate and analyze the trace file result. Just like the NS2, it is also a free software to the public, which can be obtained form the official website http://140.116.72.80/~smallko/ns2/setup.htm (Ke 2004). The TraceGraph can run under Windows, Linux, UNIX and MAC OS systems. It can be downloaded from http://www.tracegraph.com./ (Malek 2007).

Although the TraceGraph can help user to analyze the entire trace file based on different network types, for the beginners using AWK to analyze trace files is recommend. AWK is a programming language that gets its name from the 3 people who invented it (Aho, Weinberger, and Kernighan). The users can learn more technology about the data analysis via using AWK program.

2. INSTALLATION OF NS2

The Network Simulator (NS-2) is developed for several kinds of UNIX (FreeBSD, Linux, SunOS and Solaris), so it is smoothest when installed on the UNIX platform (Information Sciences Institute A 2006). NS also can be built and run under Windows. Normal scenarios should run on any ordinary machine, but very large scenarios benefit from large amounts of memory size (e.g. 1GB) (Information Sciences Institute A 2006).

Several available packets support the Simulator, such as: Tool Command Language (TCL/TK), Object Oriented Extension of TCL (OTCL), TCL with Classes (TCLCL) and so on. TK is an open source, cross-platform widget toolkit, that is, a library of basic elements for building a graphical user interface. Since the components depend on each other, they should be built in the listed order. The software packets of the NS2 also conclude some relative tools: NAM and Xgraph.

There are two kinds of ways to install it: one way is unpacking the pieces packets in proper order and then install them manually; the other way is getting everything at once by the allinone installation packet. The latest installation method is recommended, because of its convenience for beginners. If the manual way is necessary, the website below can be referenced:
http://www.isi.edu/nsnam/ns/ns-build.html#pieces.

In this chapter the installation of the allinone under both Linux and Windows will be introduced in details. The latest version of the NS2 for Linux is 2.32, and 2.29 for Windows.

2.1. Installation under Linux with Ns-allinone

1. Download the ns-allinone-2.32.tar.gz from:
   http://www.isi.edu/nsnam/ns/ns-build.html#allinone

2. Assume current directory is /home/nsuser/

3. Use the tar command to decompress the file: tar xzvf ns-allinone- 2.32.tar.gz

4. Change the current directory as ns-allinone-2.32: cd ns-allinone-2.32.

5. Run command: ./install

The NS system will be installed automatically. After the successful installation, it shows the following output (as Figure 2.1):

```
Ns-allinone package has been installed successfully.
Here are the installation places:
tcl8.4.15:      /home/wlan/NS2/NS/ns-allinone-2.32/{bin,include,lib}
tk8.4.15:               /home/wlan/NS2/NS/ns-allinone-2.32/{bin,include,lib}
otcl:           /home/wlan/NS2/NS/ns-allinone-2.32/otcl-1.13
tclcl:          /home/wlan/NS2/NS/ns-allinone-2.32/tclcl-1.19
ns:             /home/wlan/NS2/NS/ns-allinone-2.32/ns-2.32/ns
xgraph: /home/wlan/NS2/NS/ns-allinone-2.32/xgraph-12.1
gt-itm:   /home/wlan/NS2/NS/ns-allinone-2.32/itm, edriver, sgb2alt, sgb2ns, sgb2comns, sgb2hierns

---------------------------------------------------------------------------

Please put /home/wlan/NS2/NS/ns-allinone-2.32/bin:/home/wlan/NS2/NS/ns-allinone-2.32/tcl8.4.15/unix:/home/
5/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/wlan/NS2/NS/ns-allinone-2.32/otcl-1.13, /home/wlan/NS2/NS/ns-allinone-2.32/lib,
    into your LD_LIBRARY_PATH environment variable.
    If it complains about X libraries, add path to your X libraries
    into LD_LIBRARY_PATH.
    If you are using csh, you can set it like:
                setenv LD_LIBRARY_PATH <paths>
    If you are using sh, you can set it like:
                export LD_LIBRARY_PATH=<paths>

(2) You MUST put /home/wlan/NS2/NS/ns-allinone-2.32/tcl8.4.15/library into your TCL_LIBRARY environmental
    variable. Otherwise ns/nam will complain during startup.


After these steps, you can now run the ns validation suite with
cd ns-2.32; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list archive
for related posts.

wlan@linux-h1rg:~/NS2/NS/ns-allinone-2.32>
```

Figure 2.1: Reset the parameters after the successful installation

In Figure 2.1., NS reminds user to set 3 parameters: PATH, LD_LIBRARY_PATH and TCL_LIBRARY. The command below should be added to the .bashrc file:

#export NS_HOME=/home/wlan/NS2/ns-allinone-2.32/

#export PATH=$NS_HOME/tcl8.4.5/unix:$NS_HOME/tk8.4.5/unix:$NS_HOME/bin:$PATH

#export LD_LIBRARY_PATH=$NS_HOME/tcl8.4.5/unix:$NS_HOME/tk8.4.5/unix:\

#$NS_HOME/otcl-1.8:$NS_HOME/lib:$LD_LIBRARY_PATH

#export TCL_LIBRARY=$NS_HOME/tcl8.4.5/library

6. Run ./validate to make sure whether the NS2 has been installed correctly.

2.2 Installation NS2 on Windows with Cygwin

Cygwin is a Linux-like environment for Windows. It consists of two parts: (Cygwin homepage 2008).

- A DLL (cygwin1.dll) which acts as a Linux Application Programming Interface (API) emulation layer providing substantial Linux API functionality.

- A collection of tools which provide Linux looking and feeling.

The Cygwin DLL currently works with all recent, commercially released x86 32 bit and 64 bit versions of Windows, with the exception of Windows CE. (Cygwin homepage 2008.)

2.2.1 Installation of Cygwin

1. Download Cygwin, from http://www.cygwin.com/ (Cygwin homepage 2008).

2. Select the bottom of “Install or update now”

3. Install from Internet, as the Figure 2.2.:

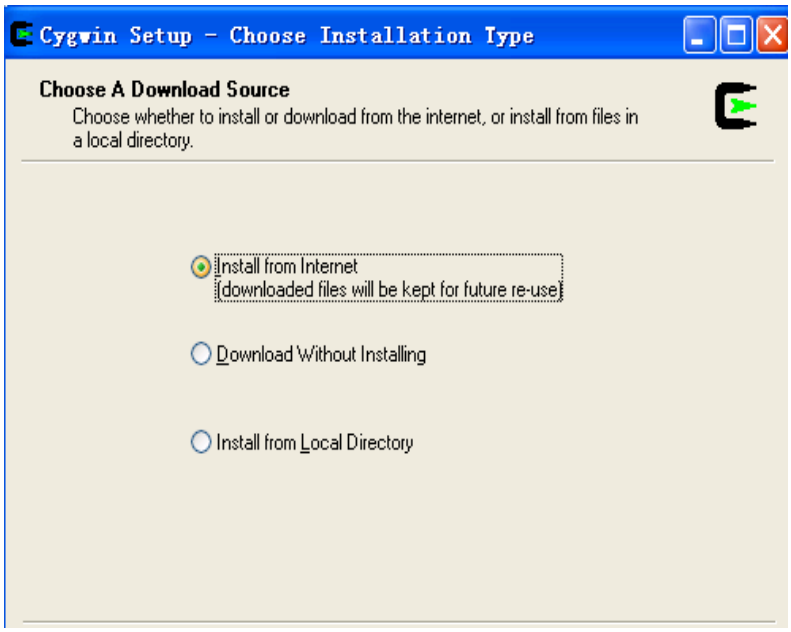

Figure 2.2: Interface of Cygwin installation

4. Install the Cygwin to the default directory: c:\cygwin

5. Select the method of network connection: Direct Connection

6. Add some necessary package for the NS2: gcc, gcc-core, gcc, g++, gawk, gzip, make, patch, perl, w32api, tar, xorg-x11-base, xorg-x11-bin, xorg-x11-bin-dlls, xorg-x11-devel, xorg-x11-libs-data, xorg-x11-etc,x-startup-scripts.

7. After the successful installation the new file “home” will be generated in the current directory c:\cygwin.(Cygwin homepage 2008.)

More details about the Cygwin installation can be found in (Information Sciences Institute 2005).

2.2.2 Installation of Ns-allinone under Cygwin

1. The Visual C++ is necessary for installation NS2 on the Windows plat, so please make sure which has been installed.

2. Download the ns-allinone-2.29.tar.gz to c:\cygwin\home\Administrator
   http://www.isi.edu/nsnam/dist/ns-allinone-2.29.2.tar.gz/

3. Find the icon of cygwin on the Desktop and type: `tar xvfs ns-allinone-2.29.tar.gz`

4. Modify some settings after the decompression: (Ke 2004).

   - Modify the makefile.vc located in otcl-1.11: annotate STATIC_TCLTK=1.

   - Modify the makefile.win located in tclcl-1.17:
     The location of the file is:
     C:\cygwin\home\Administrator\ns-allinone-2.29\tclcl-1.17\conf\makefile.win
     Edit: MSVDIR=C:\Program Files\Microsoft Visual Studio\VC98（the current directory of VC++）
     LOCAL_SRC=C:\cygwin\home\Administrator\ns-allinone-2.29; annotate STATIC_LIB=1
     Reset the values: TK_VER=83，TCL_VER = 83，TCL_SUFFIX = 8.4.11，TK_SUFFIX = 8.4.11 ， OTCL_DIR = $(LOCAL_SRC)\otcl-1.11 ，TCLCL_DIR = $(LOCAL_SRC)\tclcl-1.17。

   - Modify the file makefile.win in ns-2.29:
     The location of the target file is:

C:\cygwin\home\Administrator\ns-allinone-2.29\ ns-2.29\conf\makefile.win

Edit: MSVDIR=C:\Program Files\Microsoft Visual Studio\VC98（the current directory of VC++）

LOCAL_SRC=C:\cygwin\home\Administrator\ns-allinone-2.29 ， annotate STATIC_LIB=1

Reset the values: TK_VER=83，TCL_VER = 83，TCL_SUFFIX = 8.4.11，TK_SUFFIX = 8.4.11 ， OTCL_DIR = $(LOCAL_SRC)\otcl-1.11 ，TCLCL_DIR = $(LOCAL_SRC)\tclcl-1.17. (Ke 2004.)

- Replace .relid”as .relid’in the below files to avoid the bug mentioned in the link: http://ns-2.blogspot.com/2006/05/pr...2-allinone.html/

  C:\cygwin\home\Administrator\ns-allinone-2.29\tcl8.4.11\unixconfigure

  C:\cygwin\home\Administrator\ns-allinone-2.29\tcl8.4.11\unix\tcl.m4

  C:\cygwin\home\Administrator\ns-allinone-2.29\tk8.4.11\unix\configure

  C:\cygwin\home\Administrator\ns-allinone-2.29\tk8.4.11\unix\tcl.m4

  C:\cygwin\home\Administrator\ns-allinone-2.29\otcl-1.11\configure

5. Type cd ns-allinone-2.29 to change the current directory to ns-allinone-2.29. Then run command: `./install.`

6. Finally, some path parameters should be added to the .bashrc file, which is just like the final process of NS2 installation on the Linux platform:

#export NS_HOME=/home/wlan/NS2/ns-allinone-2.32/

#export PATH=$NS_HOME/tcl8.4.5/unix:$NS_HOME/tk8.4.5/unix:$NS_HOME/bin:$PATH

#export LD_LIBRARY_PATH=$NS_HOME/tcl8.4.5/unix:$NS_HOME/tk8.4.5/ unix:\

#$NS_HOME/otcl-1.8:$NS_HOME/lib:$LD_LIBRARY_PATH

#export TCL_LIBRARY=$NS_HOME/tcl8.4.5/library

7. Run ./validate on the startxwin.bat whose address is c:\cygwin\usr\X11R6\bin to make sure whether the NS2 has been installed correctly. The process will take some time.

The information for installation different version of ns-allinone can be seen in (Nilsson 1998: 56-61).

In windows, there are two possibilities to install NS2: using Cygwin and using VC++. Using Cygwin will make the installation almost the same as that for Linux; using VC++ is a variation and not recommended by the NS2 development group.

3. BASIC CONCEPTS OF NS2

3.1. Two Languages Implemented NS2

The programs in NS2 are briefly written in C++ and OTCL (TCL script language with Object-oriented extensions). These two languages are connected with each other via TCLCL class in NS2. Two relative classes realize the facility: one in C++; the other in OTCL. Therefore, both two structures are contained in NS (Xu, Pang & Zhao 2003: 45). The main functions of the facilities are realized in C++; Otcl mainly support the interface faced to the user. To the C++ programmer, object-oriented programming in OTCL may feel unfamiliar at first. Here are some of the differences to help to orient. (OTCL Tutorial 1995.)

- Instead of a single class declaration in C++, write multiple definitions in OTCL. Each method definition (with instproc) adds a method to a class. Each instance variable definition (with set or via instvar in a method body) adds an instance variable to an object (OTCL Tutorial 1995).

- Instead of a constructor in C++, write an init instproc in OTCL. Instead of a destructor in C++, write a destroy instproc in OTCL (OTCL Tutorial 1995). Unlike constructors and destructors, init and destroy methods do not combine with base classes automatically. They should be combined explicitly with next.

- Unlike C++, OTCL methods are always called through the object. The name self, which is equivalent to this in C++, may be used inside the method bodies. Unlike C++, OTCL methods are always virtual.

- Instead of calling shadowed methods by naming the method explicitly as in C++, call them with next. Next searches further up the inheritance graph to find shadowed methods automatically. It allows methods to be combined without naming dependencies.

- Avoid using static methods and variables, since there is no exact analogue in OTCL. Place shared variables on the class object and access them from methods by using $class. This behavior will then be inherited. For inherited methods on classes, program with meta-classes. If inheritance is not needed, use proc methods on the class object. (OTCL Tutorial 1995.)

The basic model of NS2 implementation is shown as Figure 3.1.:

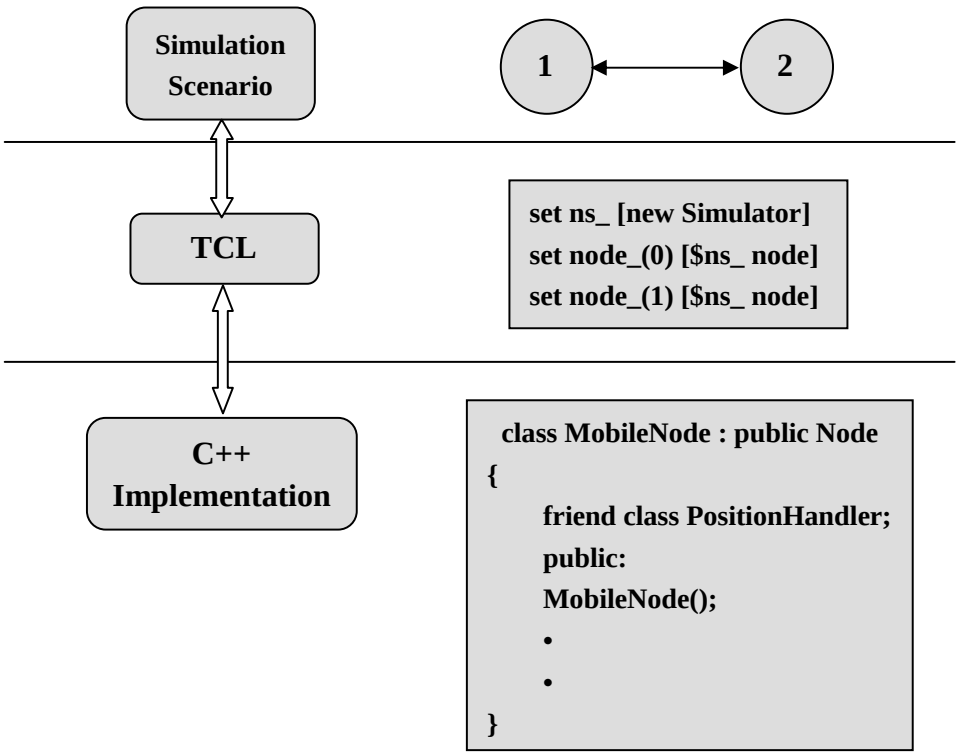

Figure 3.1: Architecture of NS2 implementation (Wang 2004: 4.)

It is easy to read from Figure 3.1, that C++ is hard to modify and adjust. However, sometimes changing the model and re–run the program is also quite important. Although TCL script is used to simulate varying parameters or configurations slightly, which means it can avoid the drawback of C++ easily. Maybe it will take a longer time to run the program sometimes.

For this reason, in NS the classes in C++ and in Otcl have some relationship. Most inheritance characters of the facilities belong to both sides. When an element is created in Otcl, it will be automatically created in C++ at mean time, in order to operate and control each other easily.

3.2. OTCL Variable and Express Method

Add a symbol “$” in front of the name of variable, such as, $a, $b. All the separate symbols in the function or program always are the curly brackets {}. Variable binding with the command set, like: set $a 5. Time is specified as a real value, optionally suffixed by a 'm' to express time in milli-seconds, 'n' to express time in nano-seconds, or 'p' to express time in pico-seconds. The default time is expressed in seconds. For example: $object set timevar 1500e9p. Command [expr …] is utilized to obtain the calculated value, [expr $a + $b]. Notice that the square brackets are necessary.

3.3. NS2 Structure and Models

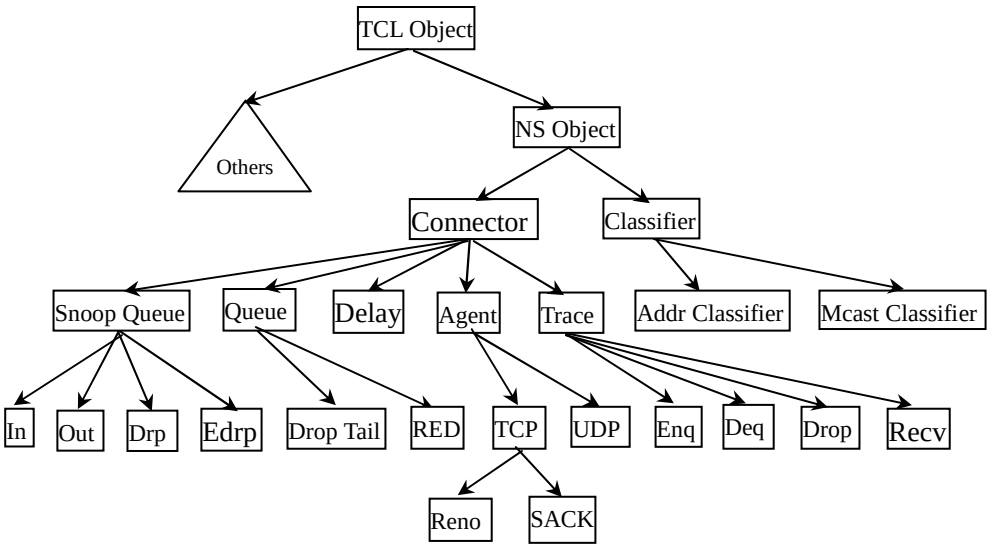

Figure 3.2: Classic hierarchy structure of NS2 (Xu, Pang & Zhao 2003: 17.)

From the hierarchy Figure 3.2, it can be seen that TCL Object is the base class for most of the other classes in the compiled hierarchies. There are two classes which can create the NS Object according to the number of the output interface: Connector (only one output) and Classfier (more than one output). In NS2, all the processes of the simulation are defined and controlled by a TCL class called Simulator, which offers a series ports for the simulation running, including the port for “event scheduler”.

Relative TCL command: set ns [new Simulator]; #establish a new simulation. $ns halt; #stop the scheduler. $ns run; #begin the scheduler. $ns at <time> <event>; #at <time> do the <event>.

3.4. Node

A node is an important structure of the Topology. In this section the methods of creating and controlling nodes will be introduced.

3.4.1. Creating and Structure of Node

The Node itself is a standalone class in OTCL. However, most of the components of the node are themselves TCLObjects. In this way, the method of creating a node is very simple: call node directly in the class simulator. TCL command: set ns [new Simulator] $ns node

The typical structure of a unicast node is as shown in Figure 3.2.:

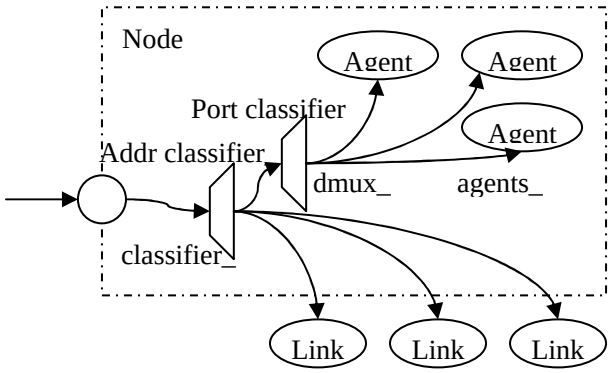

Figure 3.2: the structure of the node (Fall & Varadhan 2000: 41-42).

The structure of the node includes two TCL objects, which called address classifier and port classifier. Both are used to determine the destination address and the target agent of each node. By default, nodes in NS are constructed for unicast simulations. In order to enable a multicast simulation, the simulation should be created with an option “-multicast on”, e.g.: set ns [new Simulator -multicast on]. The structure of the multicast

node is shown in Figure 3.3.

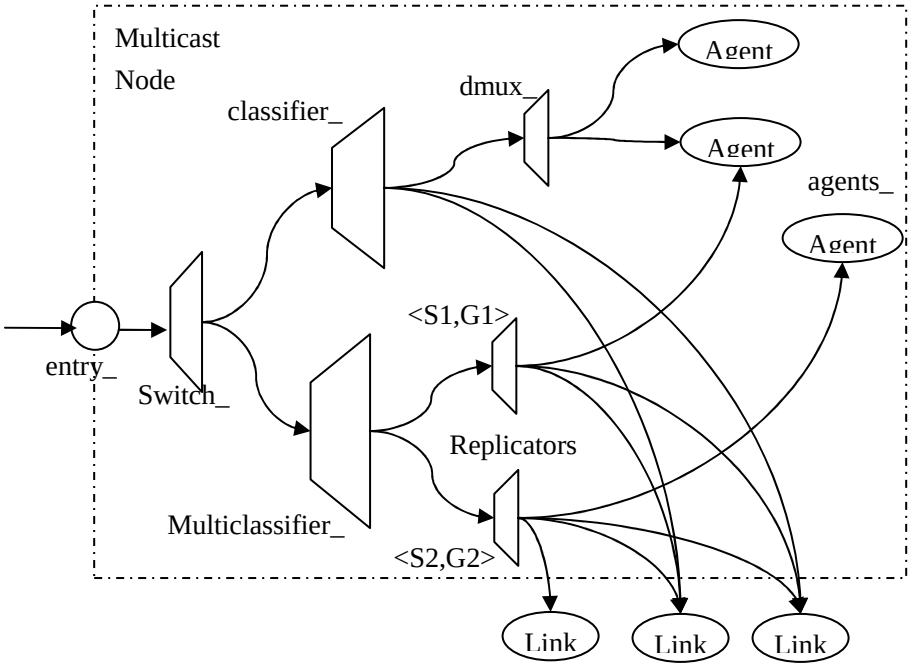

Figure 3.3.: Internal Structure of a Multicast Node (Fall & Varadhan 2000: 41-42).

3.4.2. The Node Configuration

The attributes of the node should be defined before the node is created. The attributes include the channel type, propagation model, routing protocol and decide whether switch the trace function of each layer (Agent, Router and MAC).

```
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)          AODV                    ;# ad-hoc routing protocol
set val(nn)          2                       ;# number of mobilenodes
```

```
# configure nodes
$ns_ node-config -adhocRouting $val(rp) \
                      -llType $val(ll) \
                      -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
```

3.5. Link

The method of creating a link between the nodes is: $ns duplexlink node1 node2 bandwidth delay queuetype. (Chung & Claypool B 2003.)

The purpose of the command is that two simplex links of specified bandwidth and delay, and connects the two specified nodes will be created. In NS, the output queue of a node is implemented as a part of a link; therefore users should specify the queuetype when they create links. In the common simulation case, DropTail queue is used. If the reader wants to use a RED queue, this can be done by the replacement of the word DropTail with RED. The NS implementation of a link is shown in a later section. Like a node, a link is a compound object and users can create its sub objects and connect them

and the nodes. Link source codes can be found in "ns2/tcl/libs/nslib.tcl" and "ns2/tcl/libs/ns link. tcl" files. One thing to note is that a user can insert error modules in a link component to simulate a lossy link (actually users can make and insert any network objects). Refer to the NS documentation to find out how to do this.

3.6 Agent

Agents represent endpoints where network-layer packets are constructed or consumed, and are used in the implementation of protocols at various layer. (Fall & Varadhan 2000: 71-75.)

There are several agents supported in NS2. Names of them can be seen in OTCL, now list some main agents (Fall & Varadhan 2000: 71-75):
TCP: a”Tahoe” TCP sender (cwnd=1 on any loss)
TCP/FullTcp: a more full-functioned TCP with 2-way traffic
TCPSink: a Reno or Tahoe TCP receiver
UDP: a basic User Datagram Protocol (UDP) agent
Null: a degenerate agent that discards packets used with udp agent

In the script, it is clear that the functions `“set tcp [new Agent/TCP]”, “set udp [new Agent/UDP]”, “set sink [new Agent/TCPSink]”, “set null [new Agent/Null]”` are used to create the new agent. The model of the creation is that: set <name> [new Agent/<agent name>]. Because, shown as the example, there are two packet flows: FTP and CBR, to separate them, two different flow_IDs are necessary. The commands “$tcp set fid_ 1”, “$udp set fid_ 2” are written for this task. The ID of the TCP flow is set by 1, UDP 2. (Fall & Varadhan 2000: 71-75.)

After the agents have been created the next step is to attach the nodes to the related agent. In this case, the tcp packets flow from node s1 to node d, so s1 is attached to Tcp agent and node d is attached to TCP Sink by the command “$ns attach-agent $s1 $tcp”, “$ns attach-agent $d $sink”. It means node d is the final destination of the connection. “$ns connect $tcp $sink” is used to establish the TCP connection. For the UDP connection it works similar.

Applications sit on top of transport agents in NS. They are hooked together and communicate with one agent via the applications programming interface (API). Through API, applications request services from the underlying transport agents. FTP and CBR are two most common application used in NS2. The structure of the application and the agent is shown as the figure 3.4. The attach-agent method is used to attach an application to an agent, as shown in the common example: “set ftp [new Application/FTP], $ftp attach-agent $tcp”.

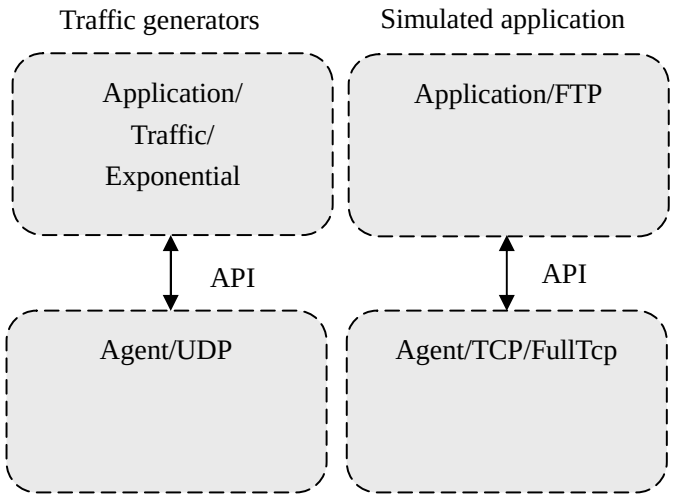

Figure 3.4: Application and the agent (Fall & Varadhan 2000: 93).

4. SIMULATIONS

In this chapter, we will introduce some simulations to demonstrate the main concepts of NS2.

4.1. Simulation of TCP Protocol

TCP (transmission control protocol) is one of the core protocols of the internet protocol suite, which is responsible for the transmission in the Internet traffic. Because of its reliability and suitability for the applications like file transfer and e-mail that sometimes the entire suite is referred to as "the TCP/IP protocol suite." Although TCP protocol is already widely developed, it continues to evolve.

In this chapter, the operation of TCP will be described at first. Then we present several NS scripts to illustrate the analysis of TCP through simulations.

4.1.1. Description of TCP

TCP has several characters: TCP service is obtained by both the sender and receiver creating end points. In TCP, the entire address of a source is called socket. It is organized hierarchically within a node. The user or process ID of the socket is called a port in TCP. The ID of port is included in the transport header for both source and destination, whereas the network and node IDs appear in the IP header. It is the reason that all sessions will normally have the same source and destination address in the IP header and only can be distinguished in the transport header if they are going from the given source host and to a given destination host (Bertesekas & Gallager 1992: 124). Thus all sessions between the same pair of host could be viewed as being multiplexed

together at the transport layer to a lower-layer session.

TCP provides reliable end-to-end transmission using sliding window Automatic Repeat-Request (ARQ). TCP allows the destination to control the flow of data from the source host. This is implemented by a 16-bit field called a window, which decides how many bytes beyond the request number can be accepted.

4.1.2. Tracing and Analyze by Examples

The explaining of the main TCL script of the example is shown in detail as below (the whole TCL script seen in APPENDIX I):

```
$ns duplex-link $s1 $r 2Mb 10ms DropTail
```

The purpose of the command is to set a duplex link between node s1 and node r, with 2Mbps bandwidth and 10ms delay. The link selects DropTail as the queue model.

```
$ns duplex-link $s2 $r 2Mb 10ms DropTail
```

The previous command is used to connect node s2 and node r with a duplex link. The bandwidth of the link is 2Mb, delay is 10ms and queue model is DropTail.

```
$ns duplex-link $r $d 1.7Mb 20ms DropTail
```

The purpose of this command is similar to the pervious ones. Connection between node r and direction node with a 1.7Mbps bandwidth duplex link is required. The delay is 20ms and the queue model is DropTail.

```
$ns queue-limit $r $d 10
```

The number of the packets waiting in queue is limited to 10 packets.

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

This command is used to set a TCP agent

```
$ns attach-agent $s1 $tcp
```

In this command, the agent TCP is attached to the node s1.

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

These two commands' functions are similar to the pervious two: set a sink agent and attach it to the direction node.

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

Now, connect the TCP agent to the sink agent.

```
$tcp set fid_ 1
```

In this command TCP agent is set as the first flow ID, which will be demonstrated in blue.

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

This command is used to establish an FTP application.

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

Then attach the application FTP to the agent TCP.

The main function of the script has been noted and explained step by step. Notice the follow scripts:

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

```
set sink [ new Agent/TCPSink]
$ns attach-agent $s1 $tcp
$ns attach-agent $d $sink
$ns connect $tcp $sink
```

The above code illustrates that in NS2, agents are firstly attached to a node via attach-agent. After that, the applications should be connected to the transport agent. After the TCL script is written, it has to be saved before it can be run with the command "./ns name.tcl". The trace file can be created automatically as well as the NAM file which is based on the trace file, shown as below:

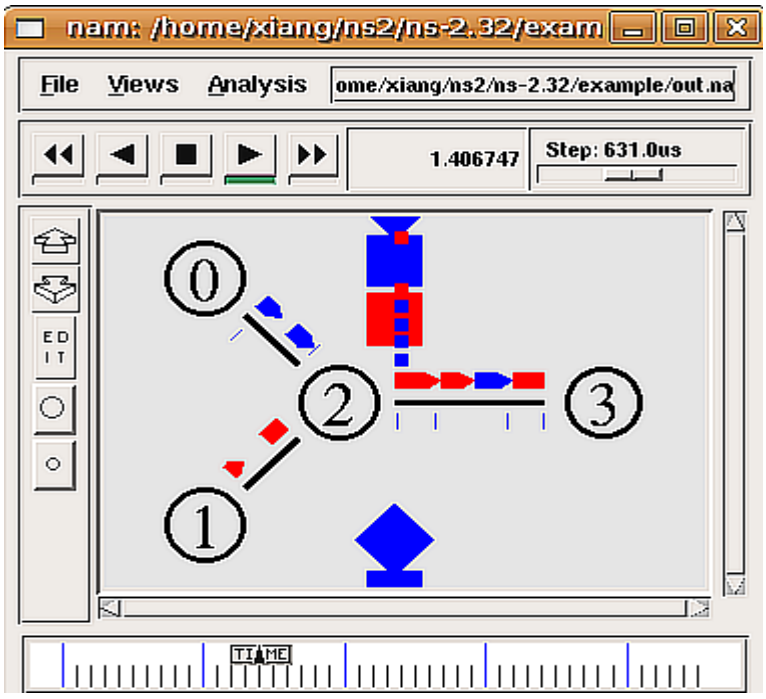

Figure 4.1.: Trace animator interface of NAM

In the figure, the blue arrows mean the ftp packets and the red ones are cbr packets. The destination of the packets is the node 3. The bigger squares are the loss packets and the smaller ones located upper are indications the packets in the waiting place.

To analyze the trace file efficiently, the AWK is necessary to be introduced. AWK is a general purpose programming language that is designed for processing text-based data, either in files or data streams. The name AWK is derived from the family names of its authors — Alfred Aho, Peter Weinberger, and Brian Kernighan. The initial purpose of AWK is to deal with the text file. And the foundation of this language is that if the data of the input line are matched with the requirement, the command will be executed. If not, it will deal with the next line automatically. (Fan 2005.)

A simple AWK command will be shown as below to analyze the delay in the example case.

CBR-delay and FTP-delay

The script of AWK shown in APPENDIX II computes the ftp and cbr packet delay, the graph plotted in Figure 4.2.

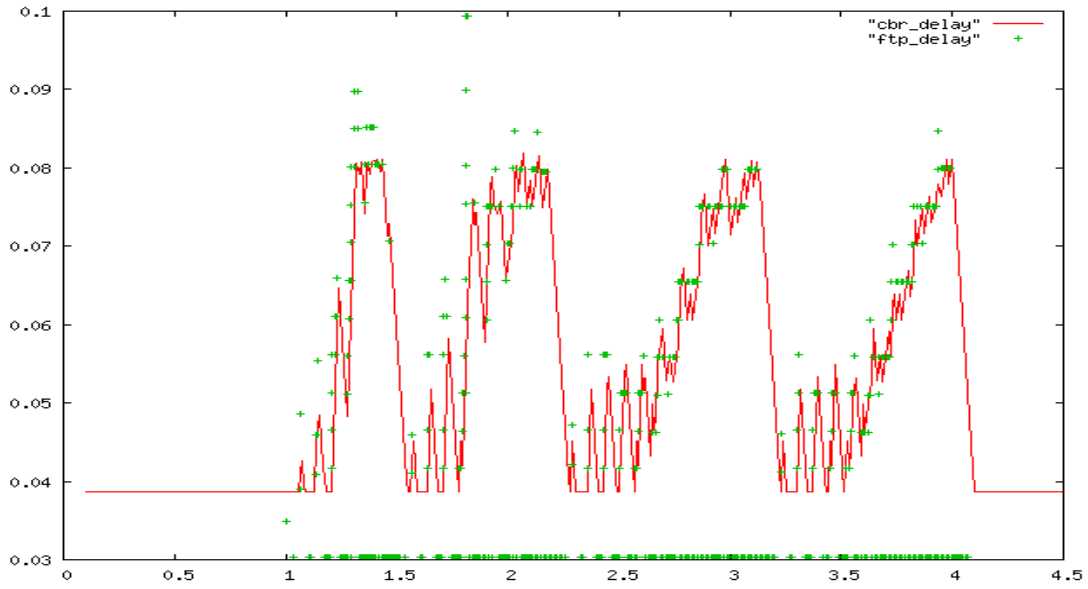

Figure 4.2.: FTP delay and CBR delay

After the program run, two files are created: “ftp_delay” and “cbr_delay”. Plot them with the program, Gnuplot. The graph is shown as the upper. In the graph, the cbr_delay is stable between 0.1s and 1.0s as well as 4.0s and later, because at that time the ftp application has not been started or ended. There is only CBR packet in the channel and no congestion happens. After 1.0s the ftp packets are transmitted. Some packets must wait in the queue, some got lost. That is the reason of the obviously delay happens during this period.

Jitter

Jitter is an unwanted time-variation of one or more signal characteristics in electronics and telecommunications. Jitter may be seen in characteristics such as the interval between successive pulses, or the amplitude, frequency, or phase of successive cycles (Jitter wikipedia 2008). Jitter is a significant factor in the design of almost all communications links. It is a delay variance based on the network estate. That means the larger the jitter, the more unstable is the network.

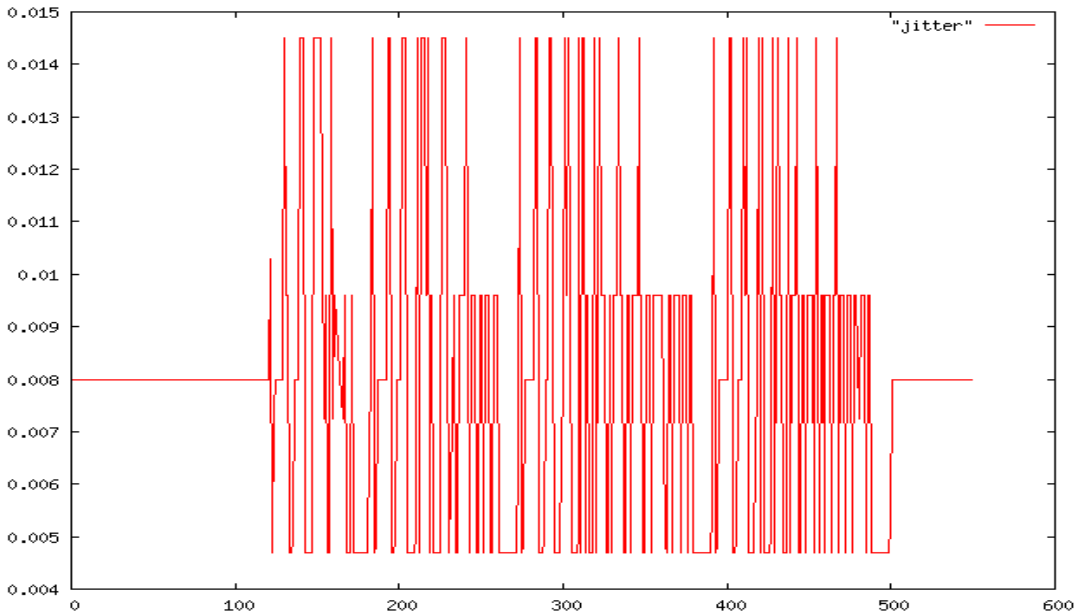

Figure 4.3.: Jitter of CBR packets

From the jitter plot, it is clear that the change of jitter is synchronous to the end to end delay. Because the reasons of the change are the same: the FTP packets join in the transmission. The whole AWK script is shown in APPENDIX IV.

Through AWK we can also compute some other characters in the system, such as loss, throughput and some others. In the previous example, we can also obtain that 550 CBR packets are sent and 8 of them are lost. The same, we can compute that 10 FTP packets are lost among 246 packets in all, which can be computed by APPENDIX III.

Now analyze another TCP model. All the nodes send FTP packets to node 0 via node 1 at a random and delay time internal from 0s to 7s also from 0s to 7s.

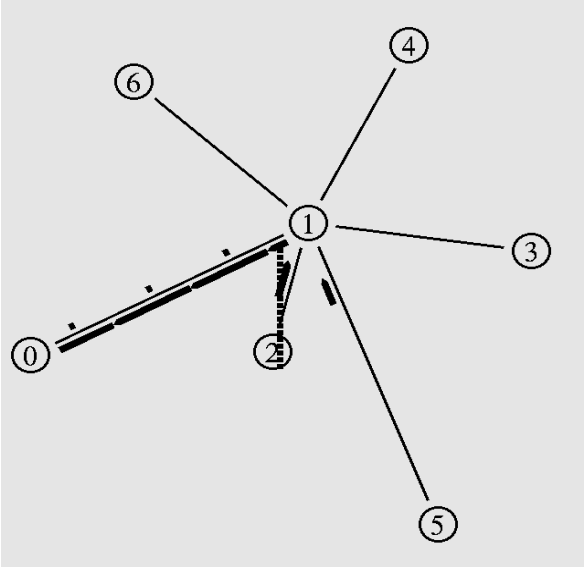

Figure 4.4.: Another tcp model

The script of the TCL is almost the same as the previous one. Note that the various of the random value should be defined at first: set rng [new RNG] (Altman & Jimene 2003: 76); $rng seed 0. set RVstart [new RandomVariable/Uniform]; $RVstart set min_ 0; $RVstart set max_ 7; $RVstart use-rng $rng. The function of start at a random time from

0s to 7s can be realized. The same for the delay set. The whole script will be given in the APPENDIX VII.

After the analysis we can obtain that there are 2940 packets sent and 93 of them are lost.

4.2. Simulation of Router Layer

The major task at the network layer is routing and flow control. In fact they have been utilized in the former example. At the network layer, the transmission of packets between adjacent nodes can be distinguished of one session from another as well as different packets within the same session (Bertesekas & Gallager 1992: 124). When a node receives a packet, the information contained in the packet determines the node how to forward it. Because the header of each packet contained identification numbers for both the source and destination even each site is accessed during the transmission.

In the virtual circuits, the path through the network is given and there is a certain set of sessions using each link. It is helpful to realize the link as being shared by a set of virtual channels distinguished by numbers. When a new session will be established, a path is set by assigning, on each link of the path, one unused virtual channel to that session. Each node also keeps a table mapping each busy incoming virtual channel on each link onto the corresponding outgoing virtual channel and link for the corresponding session.

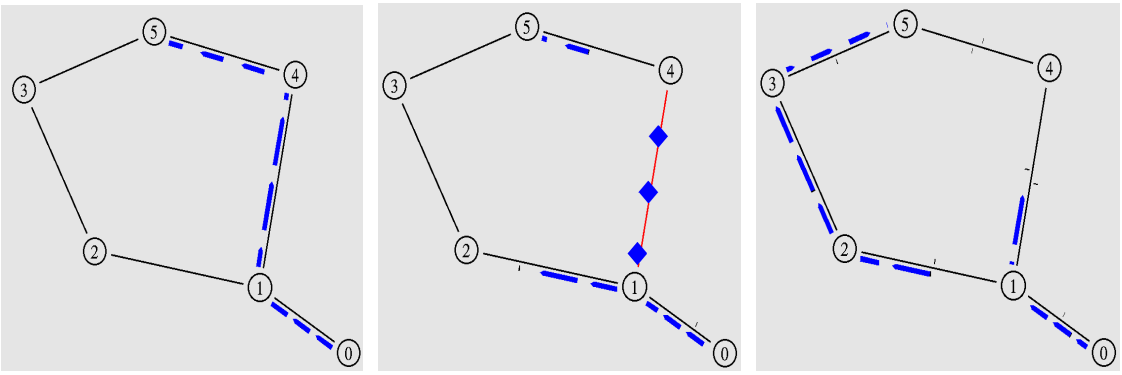

Figure 4.5.: Model of route selection

In the previous example, there are two different routes from source node 0 to destination node 5. The static routing, used by NS2, is the simpler one in which the shorter routing is chosen throughout the connection. The example simulates a disconnection between node 1 and node 4 from 1.0s to 3.0s. It is necessary to type:

```
$ns rtmodel-at 1.0 down $S(1) $S(4)
$ns rtmodel-at 3.0 up $S(1) $S(4)
```

In the example, a default route is chosen the route 0-1-4-5 for setting connections. In contrast to the static route, the Internet will find an alternative route when the original route disconnected. The operation in NS2 is used by adding the command: $ns rtproto DV (Fall & Varadhan 2000: 63).

In the previous example, the link 1-4 is down from 1.0s to 3.0s. In NAM file, it is clear that the link becomes red during its disconnection. And all the packets transmitted in the link are drop. Another TCP connection is established from node 0 to node 5. (APPENDIX VIII)

In the NAM trace, the result can be obtained that in the dynamic routing case, the signaling packets which are used to determine the path, not only at the beginning, but

also at the connectivity changes.

4.3. Simulation of Wireless Network

There are two structures for wireless communication between two hosts. The first is the centralized cellular network. In this case, the mobile is connected to the fixed base station, so that the communication between two nodes needs one or more base stations. Different scenarios can be considered as well, such as hard, soft and softer handover. The second method of the wireless is based on the ad-hoc network between two mobile nodes wish to communicate each other. Compared to the fixed base station, the ad-hoc networks have more limited range of a mobile terminal which means that mobile nodes do not need to be the source or the destination of the packets, but also to forward the packets between other mobiles. A cellular station has much larger communication range, however the advantage of the ad-hoc network is quickly deployable and without an existing infrastructure.

In cellular networks, the wireless part is restricted to the access to a network, and within it, the classical routing protocol can be utilized. Ad-hoc network in contrast rely on the special routing protocols. (Altman & Jimene 2003: 111-125.)

In ad-hoc networks the routing protocols are central. NS2 allows simulating the main existing routing as well as the transport and applications that use them. The current routing protocols used by NS2 are (Altman & Jimene 2003: 111-125).:

DSDV - Destination Sequenced Distance Vector
AODV - Ad-hoc on Demand Distance Vector
DSR - Dynamic Source Routing

TORA/IMPE - Temporally Ordered Routing Algorithm / Internet Mobile Ad-hoc Network (MANET) Encapsulation

4.3.1. The Routing Protocol Algorithm

(1) DSDV is a distance vector routing protocol. Each node has a routing table which indicates the destination. The destination is the next hop and the number of hops to the destination. Each entry in the routing table contains a sequence number. The sequence numbers are generally even if a link is present; otherwise, an odd number is used. The number is generated by the destination, and the emitter needs to send out the next update with this number. Routing information is distributed between nodes by sending full dumps infrequently and smaller incremental updates more frequently (Perkins & Bhagwat 2004: 236-238). If a router receives new information, then it uses the latest sequence number. If the sequence number is the same as the one already in the table, the route with the better metric is used. Stale entries are those entries that have not been updated for a while. Such entries as well as the routes using those nodes as next hops are deleted (Perkins & Bhagwat 2004: 236-238). If a node detected that a route to the destination has been broken, then its hop number is set to infinity and its sequence number is updated but an odd number assigned. (Altman & Jimene 2003: 111-125.)

(2) AODV is a distance vector type routing. It is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Additionally, AODV forms trees which connect multicast group members. The trees are composed of the group members and the nodes needed to connect the members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes.(Belding 2007.)

The protocol use different messages to discover and maintain links: Route Requests (RREQs), Route Replies (RREPs) and Route Errors (RERRs). These messages are typed via UDP, and normal IP header processing applies.

When a source node desires a route to a destination for which it does not already have a route, it broadcasts a RREQ packet across the network. Nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the route tables. In addition to the source node's IP address, current sequence number, and broadcast ID, the RREQ also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the RREQ may send a RREP if it is either the destination or if it has a route to the destination with corresponding sequence number greater than or equal to that contained in the RREQ. If this is the case, it unicasts a RREP back to the source. Otherwise, it rebroadcasts the RREQ. Nodes keep track of the RREQ's source IP address and broadcast ID. If they receive a RREQ which they have already processed, they discard the RREQ and do not forward it. (Belding 2007.)

When the RREP propagates back to the source, the nodes set up forward pointers to the destination. Once the source node receives the RREP, it may begin to forward data packets to the destination. If the source later receives a RREP containing a greater sequence number or contains the same sequence number with a smaller hop count, it may update its routing information for that destination and begin to use the better route. (Belding 2007.)

Nodes, part of an active route, may offer connectivity information by broadcasting local “Hello” messages (special RREP messages) to its neighbors. If “Hello”

messages stop arriving from a neighbor beyond some time threshold, the connection is assumed to be lost.

As long as the route remains active, it will continue to be maintained. A route is considered active as long as there are data packets periodically traveling from the source to the destination along that path. Once the source stops sending data packets, the links will time out and eventually be deleted from the intermediate node routing tables. If a link break occurs while the route is active, the node upstream of the break propagates a RERR message to the source node to inform it of the now unreachable destination(s). After receiving the RERR, if the source node still desires the route, it can reinitiate route discovery.

AODV does not allow the handling of unidirectional links.

(3) DSR uses source routing instead of relying on the routing table at each intermediate device. A source requested to send a packet to the destination broadcast a RREQ packet. Nodes receive the RREQ packet and search in their route cache for a route to the destination. If a route can not be found, the RREQ will be transmitted further and the node will add its own address to the recorded hop sequence. The process will be lasted, till the destination can be found or a node with the route to the destination are reached. The route back can be computed based on the hop record. If the routes are not symmetric, DSR checks the route cache of the replying node. If a new route is found, it will be instead. Compared to AODV protocol, the unidirectional links handling is allowed in DSR. (DSR wikipedia 2006.)

(4) TORA is one protocol of the family of link reversal protocols. It may provide several routes between the source and the destination. There are three parts of the TORA: creating, maintaining and erasing routes. At each node a separate copy of

TORA is run. Therefore, TORA builds a directed acyclic graph rooted in the destination. It associates a height with each node in the network. Message flows from the higher heights to the lower heights. When a node has no downstream link it reverses the direction of one or more links. If a node can not find the route to the particular destination, it sets the corresponding local height to the maximum value. (TODA wikipedia 2005.)

4.3.2. Simulation of a Mobile Example

NS2 can simulate many kinds of communication networks. Next we demonstrate how to simulate a wireless network.

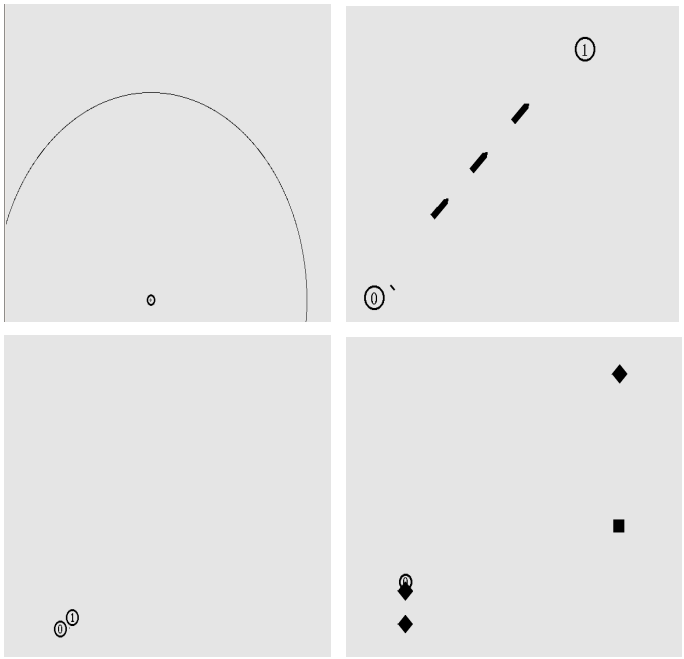

Figure 4.6.: Example in wireless case

One node moves to another when enter the certain range the path connected and the packets send to each other. When the node moves beyond the communication range, the packets are lost. In the wireless case, the signal power strength goes inverse ratio with

the rising distance. There are different fading formulas in different propagation models. The whole TCL script can be seen in APPENDIX IX.

Only when the received power is over the threshold the receiver node receives packets correctly. Default value of the threshold is: and the distance is 250 meters. In the NS script the receive value can the reset by the command: Phy/WirelessPhy set RXThresh_ (new value).

NS2 provides three propagation models: FreeSpace, TwoRayGround and Shadowing model.

In the FreeSpace model the received power represents:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi d)^2 L} \qquad (1)$$

Where P_t is Transmission Power; P_r is Received Power; G_t means Transmission Antenna Gain and G_r means Received Antenna Gain; λ means Wavelength; d is Distance; at last, L shows System Loss.

In the TwoRayGround model: if $d < \frac{4\pi h_t h_r}{\lambda}$, the receive power is equal to the FreeSpace case; else the receive power presents

$$P_r = \frac{P_t G_t G_r (h_t h_r)^2}{d^4 L} \qquad (2)$$

Where h_t is Transmission Antenna Height; h_r is Antenna Height for the received antenna.

In the Shadowing model, the distance d has been defined as 1. So the receive power is shown:

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 L} \chi \qquad (3)$$

Where χ indicates the lognormal distribute.

In NS2 the parameters in the formula are set as: λ = 3.0e8/freq; transmission power, Pt = 0.28183815; transmission antenna gain G_t = 1.0; received antenna gain G_r = 1.0; frequency freq = 914.0e6; loss sysLoss = 1.0; transmission antenna height h_t = 1.5; received antenna height h_r = 1.5.

In NS2, there is a tool used to compute the threshold value of the received power based on the different communication range. The tool is located in: ~ns/indep-utils.

.

Compile the file: g++ threshold.cc -o threshold at first. The command format of threshold presents: threshold -m <propagation-model> [other-options] distance. Obtain the new value of received power based on the new distance.

5. EMULATIONS

This chapter describes the emulation facility of NS. Emulation refers to the ability to introduce the simulator into a live network. Special objects within the simulator are capable of introducing live traffic into the simulator and injecting traffic from the simulator into the live network. (Fall & Varadhan 2000: 336-341.)

Because of the currently limited portability of emulation, it is compiled into Network Simulation Emulation (NSE) only. Before the emulation it is necessary to built firstly (build it with “make nse”). And make sure that -lnsl -ldl -lpcap\ are in lib of makefile.

5.1. Introduction of NSE

Network simulator emulator (NSE) is an extension to NS2, which provides basic utilities for reading and writing live packets from/to the live network. Figure 5.1 represents the emulation model implemented in NSE (Nethi, Pohjola, et al. 2007: 4).

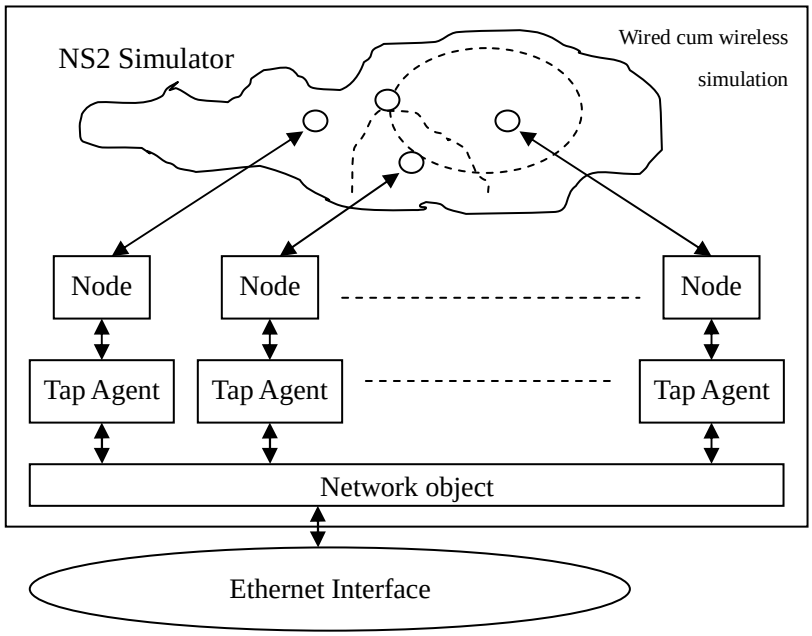

Figure 5.1.: Internal Flow Diagram of NSE (Nethi, Pohjola, et al. 2007: 4)

The emulation facility can be subdivided into two modes:

1. opaque mode – live data treated as opaque data packets
2. protocol mode – live data may be interpreted or generated by simulator

In opaque mode, the simulator treats network data as uninterpretable packets. In particular, real-world protocol fields are not directly used by the simulator. In opaque mode, live data packets may be dropped, delayed, re-ordered, or duplicated. Because no protocol processing is performed, protocol-specific traffic manipulation scenarios may not be performed (Fall & Varadhan 2000: 336-341).

In protocol mode, the simulator is able to interpret or generate live network data packets which contain arbitrary field assignments.

The components of the Network Simulator Emulator (NSE) are listed below:

Real-time scheduler: The real-time scheduler attempts to synchronize the execution of events in real-time. The function of the scheduler is used to introduce an *NS* simulated network into a real-world topology to experiment with easily-configured network topologies, like cross-traffic, etc. This only works for relatively slow network traffic data rates, as the simulator must be able to keep paces with the real-world packet arrival rate, and this synchronization is not presently enforced (Nethi, Pohjola, et al. 2007: 4).

TapAgent: This class is a simple class derived from the base Agent class. As such, it is able to generate simulator packets containing arbitrarily-assigned values within the NS common header. The tap agent handles the settings of the common header packet size field and the type field. The packet type field is PT_LIVE for packets injected into the simulator. Each tap agent can have at most one associated network object, although more than one tap agent may be instantiated on a single simulator node. It is also responsible for writing packets onto the network interface (Nethi, Pohjola, et al. 2007: 4).

Network objects: Network objects provide access to a live network (or to a trace file of captured network packets). There are several forms of network objects, depending on the protocol layer specified for access to the underlying network, in addition to the facilities provided by the host operating system. Use of some network objects requires special access privileges where noted. Generally, network objects provide an entry point into the live network at a particular protocol layer (e.g. link, raw IP, UDP, etc) and with a particular access mode (read-only, write-only, or read-write). Some network objects provide specialized facilities such as filtering or promiscuous access (i.e. the pcap/bpf network object) or group membership (i.e. UDP/IP multicast). The C++ class Network is provided as a base class from which specific network objects are derived.

Three network objects are currently supported in NSE: Pcap/BPF, raw IP, and UDP/IP.

Each is described below:

Pcap/BPF objects provide an extended interface to the LBNL packet capture library (libpcap) (Nethi, Pohjola, et al. 2007: 4). This library provides the ability to capture link-layer frames in a promiscuous fashion from network interface drivers (i.e. a copy is made for those programs making use of libpcap). It also provides the ability to read and write packet trace files in the “tcpdump” format. The extended interface provided by NS also allows for writing frames out to the network interface driver, provided the driver itself allows this action. Use of the library to capture or create live traffic may be protected; one generally requires at least read access to the system’s packet filter facility which may need to be arranged through a system administrator (Information Sciences Institute C 2006).

Raw IP objects provide raw access to the IP protocol, and allow the complete specification of IP packets (including header). The implementation makes use of a raw socket. In most UNIX systems, access to such sockets requires super-user privileges (Information Sciences Institute C 2006). In addition, the interface to raw sockets is not such a common standard than other types of sockets. The class Network/IP provides raw IP functionality plus a base class from which other network objects implementing higher-layer protocols are derived.

UDP/IP objects provide access to the system's UDP implementation along with support for IP multicast group membership operations. (Information Sciences Institute C 2006.)

5.2. Integration NS2 with Other Simulation Packages

The function of NSE is capable of introducing live traffic into the simulator and injecting traffic from the simulator into the live networks, which provide a possibility of evaluating the performance of communication protocols in real-time control systems and similarly test robustness and performance properties of control and data fusion algorithms in networked environments. Hence, it can be realized the integration NS2 with other packets under the NSE mode. (Fall & Varadhan 2000: 336-341.)

The traditional control theory is not suitable for the asynchronous systems, because it assumes constant sample times. Hence, it is necessary to develop a theory which can integrate wireless communication and control. NS2 can integrate with different simulator to realize the combination between controlling part and simulation part. In this thesis we select MATLAB as an example. Because MATLAB is also a widely employed research tools used in control system design and simulation.

The key features of the integration are: 1) support for powerful control design and implementation tools provided by MATLAB, Simulink and xPC Target enabling automatic code generation from Simulink models for real-time execution, 2) real-time control of a true or simulated process over a user-specified network, 3) capability to emulate any wired/wireless networks readily available in NS2, 4) easy-to-use network configuration tool and 5) the platform is accessible over the Internet, i.e. it supports remote experimenting. (Nethi, Pohjola, et al. 2005: 8.)

5.2.1. One Example of Integration

Now we take the example of ‘PiccSIM’, which is developed by the Communication and control Engineering Groups at Helsinki University of Technology (HUT), to illustrate

the integration NS2 with other packets.

PiccSIM is a platform for modeling, design, simulation and implementation of network control systems, and it integrates the control design tools available in MATLAB with NS2.

By using the PiccSIM platform, it is possible to integrate the network simulator with real processes and to only simulate the network while the control algorithms are executed on a real-time operating system, which controls real processes. The key idea is that a mobile node can measure the distances to its nearby static sensor nodes, and a temporal mobile node location can be computed by using at least three distance measurements (Nethi, Pohjola, et al. 2005: 8). Once the command center tracks the mobile node, it can guide it towards the reference path.

During the process of the research we have amended the exist demonstration of HUT:

In the HUT case, the sensor nodes are static by the uniform distributed. This means that the distance between each neighbor node can be controlled within the communication range easily. The system consists of the static wireless sensor nodes scattered in a grid distribution. The distances between each node are 200 meters and the distance measurement range is defined as 300 meters. If the distance between the mobile node and static nodes is less than 300 meters, the distance between them will be measured and transmitted to the computation center for estimating the position of the mobile node.

In our case, shown in APPENDIX X., the significant change is that all the static sensor nodes are randomly deployed in the network area which is helpful for the application in the real environment. The locations of the randomly static nodes are defined by the

localization method. As the scenario, all the static wireless sensor nodes are randomly deployed, a reference path is also randomly created and the mobile node should work follow the computed path. We simulate this scenario via the wireless networked control system (WiNCS). The objective of this work is to present the integrated control processing under the simulative Wireless Sensor Network (WSN) environment. Both the location of the sensor nodes and the mobile path are created by MATLAB, but are read to NS2 by the source command (Fall & Varadhan 2000: 336-341).

Based on these work, we can make the simulation case more closely approximately to the real situation. The results can be used to support the future development.

5.3 Architecture of Integration

The PiccSIM system typically consists of three computers (MoCoNet Server, RTOS xPC Target and NS2) and an I/O controller board (link to real process), but in our case the remote accessibility property is disregarded, so two computers are enough: one for NS2 and another for MATLAB. In Figure 5.2, they are connected via their own local area network (LAN).

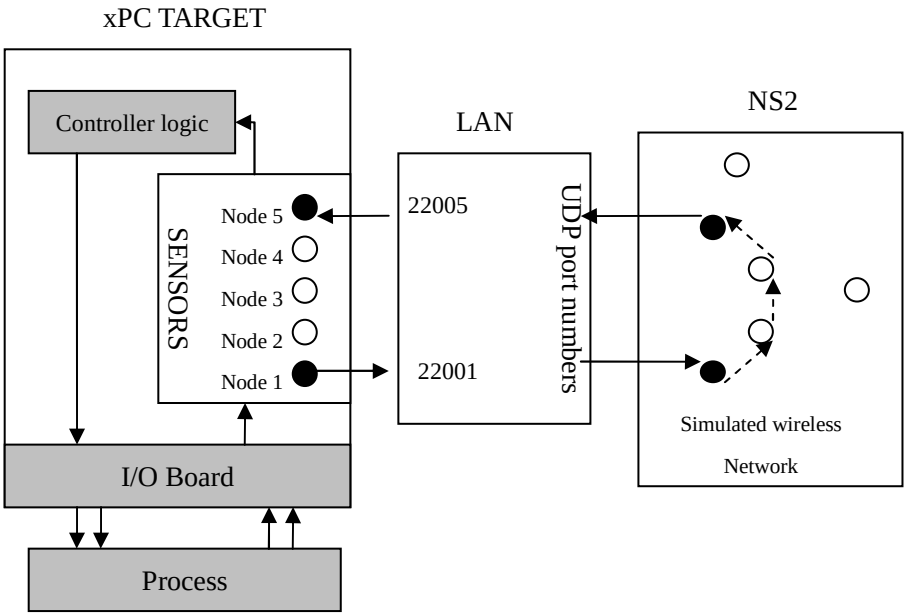

Figure 5.2.: Nodes in the xPC Target and the NS2 network are associated with UDP port numbers (Nethi, Pohjola, et al. 2007: 3)

The xPC target runs a real-time operating system. The main function of this computer is to measure and control the processes based on the user specified algorithms, which are made with a Simulink model in the MATLAB, storm where UDP or TCP packets are transmitted then the xPC transmits signals (i.e. UDP packets) to the network simulator.

The NS2 computer is using the emulation (NSE) to capture the User Datagram Protocol (UDP) packets and inject them into simulated wireless network model. Thus communication between any two nodes in the control system is passed through NS2. Figure 5.2 shows the connectivity mapping between xPC Target and NS2 nodes. UDP port numbers are used to tap and inject the packet to the corresponding node in NS2.

Figure 5.2 shows the components of the integration model. The process is measured and controlled with xPC Target computer equipped with an I/O controller board. One of the nodes in the network acts as the process controller that computes the control signal for

the process. The signal is transmitted over the network to the actuator in the process (Nethi, Pohjola, et al. 2007: 3). The sensor and actuator nodes in the real process and the simulated network are associated by their UDP port numbers, which makes it possible that the real process maps to the simulated network.

In Figure 5.2, the process of communication is presented clearly. Node 1 is the source and Node 5 is the destination. Node 1 on the xPC Target represents a sensor. It creates UDP packets of the signals measured either from a real case or a simulated process. Any packet generated by Node 1 is destined to Node 5 (a controller node) passes through the simulated network on NS2.

The NS2 computer uses packet filtering tools to capture the UDP packets and associates it to the correct node in the simulated network by the UDP destination port number. It then performs a mapping for the simulated destination node and the packet is injected into the simulated network in NS2. After a successful receipt, the packet is sent back to the xPC Target via the LAN.

In the NS2 part the key commands to connect two computers are:

```
set ns [new Simulator]
$ns use-scheduler RealTime
```

Create a new simulator and the real-time scheduler requires the following specification at the beginning of a simulation script.

```
set me [exec hostname]
```

The purpose of this command is to determine the name of the local system.

```
set pf1 [new Network/Pcap/Live]
```

This command creates an instance of the pcap network object for capturing live traffic.

```
$pf1 set promisc_ true
```

The function of this command is to tell the packet filter whether it should configure the undelying interface in promiscuous mode.

```
set intf [$pf1 open readonly]
```

The open call activates the packet filter, and may be specified as readonly, writeonly, or readwrite. Itreturns the name of the network interface the filter is associated with.

```
puts "pf1 configured on interface $intf"
set filt "(ip src host foobar) and (not ether broadcast)"
set nbytes [$pf1 filter $filt]
puts "filter compiled to $nbytes bytes"
```

The filter method is used to create a Berkeley Packet Filter (BPF)-compatible packet filter program which is loaded into the underlying BPF machinery. The filter method returns the number of bytes used by the filter predicate.

```
puts "drops: [$pf1 pdrops], pkts: [$pf1 pkts]"
```

The pdrops and pkts methods are available for statistics collection. They report the number of packets dropped by the filter due to buffer exhaustion and the total number of packets that arrived at the filter, respectively (*not* the number of packets accepted by the filter).

5.4. Related Protocol

Multi-path routing consists of finding routes between a source node and a destination node. Multi-path routing doesn't provide only schemes for an efficient resource management but also improves overall system performance (Nethi, Pohjola, et al. 2007: 3). Multi-path routing exploits path diversity and compensates the dynamic and unpredictable nature of sensor networks.

In this thesis, the popular Ad hoc On Demand Distance Vector routing protocol (AODV) is chosen to work as well as Local Multiple Next Hop Routing Protocol (LMNR) (Nethi, Pohjola, et al. 2005: 8). LMNR makes the network congested by reason of a lot of traffic generated. It tries to find multiple paths, and contrary to many multi-paths routing protocols, it uses single path to avoid synchronization of packets at the destination node (Nethi, Pohjola, et al. 2005: 6). The strongpoint of the LMNR scheme is that each source and the intermediate nodes give the liberty to choose from multiple local paths the destination, which allows local route selection making it adaptive to a dynamically changing environment.

5.4.1. Add New Route Protocol in NS2

In our case the protocol LMNR named AODVT. Although it is a multicast routing protocol developed from the AODV algorithm, it is still a new protocol in NS2. This section will introduce the method of how to add a new protocol in NS2. Here we select AODV as a sample to add a new protocol to the NS2 without changing the content. In the real time simulation the AODVT is provided from the HUT emulation case.

(1) Copy the aodv fold located in ~/ns-allinone-2.32/ns-2.32 to aodvt, and replaces all the "AODV" or "aodv" by "AODVT" and "aodvt" in aodv.cc and aodv.h. (Xu 2008.)

(2) Modify the packet.h file in the ~/ns-allinone-2.32/ns-2.32/common. Find the word of “aodv”, obtain that: name_[PT_AODV]= "AODV"; PT_AODV. Copy them and replace the aodv by aodvt: name_[PT_AODV]= "AODV";name_[PT_AODVT]= "AODVT"; PT_AODV, PT_AODVT.

(3) Modify the ns-lib.tcl file which is located in ~/ns-allinone-2.32/ns-2.32/tcl/lib. Search the term “AODV” and “aodv” in the file and duplicate it with “AODVT” and “aodvt” respectively.

(4) Edit ns-packet.tcl file just like the previous example. The file’s location is ~/ns-allinone-2.29/ns-2.29/tcl/lib.

(5) Modify the makefile in ~/ns-allinone-2.32/ns-2.32. Search the word of “aodv.o” and then obtain the sentence “aodv/aodv_logs.o aodv/aodv.o \; aodv/aodv_rtable.o aodv/aodv_rqueue.o \” Add the related command “aodvt/aodv_logs.o aodvt/aodv.o \; aodvt/aodv_rtable.o aodvt/aodv_rqueue.o \” after the command about aodv.

(6) After the previous steps a new route protocol named aodvt is almost installed, but the trace format needs still some modification. It is necessary to modify the files: cmu-trace.cc and cmu-trace.h, in ~/ns-allinone-2.32/ns-2.32/trace. Add the relative command according the aodv part: void CMUTrace::format_aodvt(Packet *p, int offset) {…}. (Xu 2008.)

(7) Do “make clean” in the terminal at the directory: ~/ns-allinone-2.32/ns-2.32, after that do “make”.

These were the main processes of adding a new route protocol in NS2. In conclusion the

best solution is to search the term “AODV” and “aodv” in all the NS2 files and replace it with “AODVT” and “aodvt” respectively.

5.5 Result Analysis

Due to the trace file in AODV protocol is very complicated; the result here is analyzed via the software “TraceGragh”, which is easier to operate than coding the AWK script. It can analyze most of parameters of the simulation, such as: end to end delay, throughput, jitter, etc. However, in the simple case the AWK language is still recommended for beginners to realize and to manage the basic concept of the computing process for the analysis result.

It is well known that the major problems arising from wireless networks are varying because of the time delay and packet losses in the communication process. Hence, this section analyzes the different results based on two different protocols: AODV and LMNR.

During the simulation we observe three parameters mainly in two different scenarios: End to End delay, Jitter and the number of the dropped packets. The last one means the level of the reasonable utilization in the network resource. Figure 5.3, Figure 5.4 and Figure 5.5 present the network properties difference between the AODV and LMNR during the simulation period.

AODV LMNR

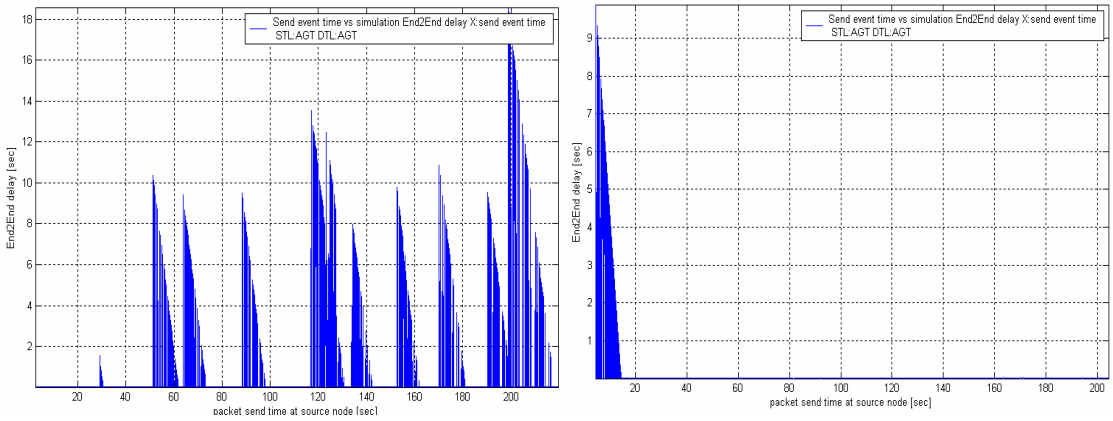

Figure 5.3.: End to End delay: AODV vs. LMNR

In Figure 5.3, it can be recognized that the End to End delay in AODV is much higher than in LMNR. From 50s to 200s the delays under AODV protocol vary frequently, while in LMNR during the same time period no delay appears which exactly explains the character of LMNR well. It makes the network congested because of a lot of the highly generated traffic. It tries to find multiple paths, and contrary to many multi-paths routing protocols. A single path is used to avoid synchronization of packets at the destination node (Nethi, Pohjola, Gao & Jantti 2005).

AODV LMNR

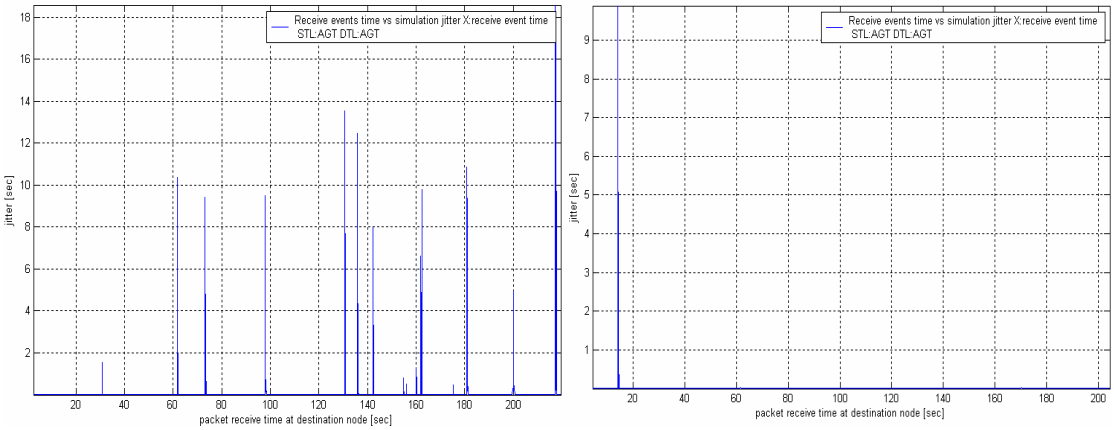

Figure 5.4.: Jitter: AODV vs. LMNR

The situation of Jitter shown as Figure 5.4 is almost synchronous to the situation of an End to End delay. Because a jitter is a delay variance based on the network estate. Figure 5.4 shows that the networks are based on LMNR protocol which is much more stable than the networks based on AODV protocol.

AODV LMNR

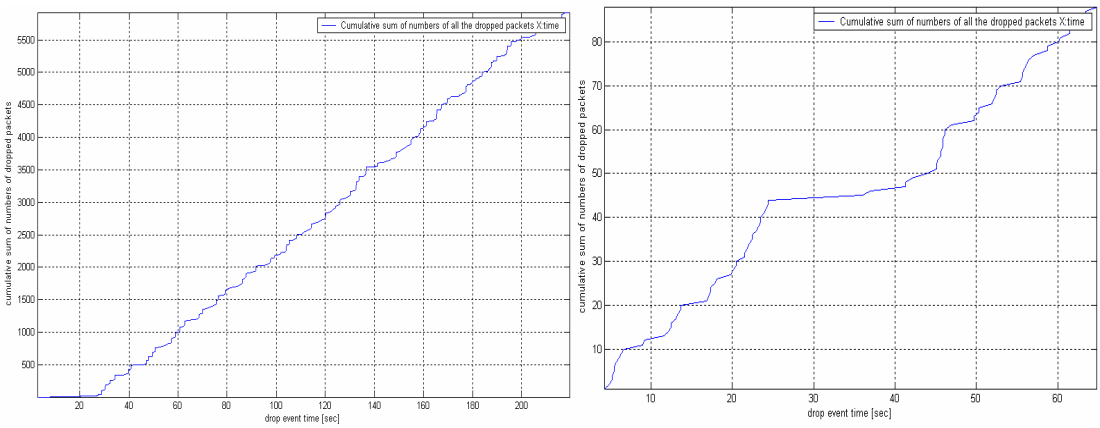

Figure 5.5.: The number of the dropped packets: AODV vs. LMNR

To prove this point, Figure 5.5 delivers the number of the dropped packets between two protocols. In LMNR case, there are only approximately 90 packets dropped within the simulation process compared to the huge amount (close to 6000) of the dropped packets under VODA protocol, and the drops only happen within the first 60 seconds during the simulation of LMNR caused by the high traffic. Nevertheless, the drops appear continuously during the whole process of the AODV simulation.

In a nutshell, the new protocol LMNR is much more adaptive for a dynamic and unpredictable nature of sensor networks.

6. LIMITATIONS OF NS2

Although NS2 is widely used for simulation of different network systems, and many experiments have leveraged it to examine protocols and distributed systems. However, it has some limitations. The purpose of this chapter is to discuss several limitations of NS2.

On one hand, NS2 integrates the TCL scripting language into the tool, and therefore offers similar flexibility and determinism in the simulation. The main limitation with NS2 is that it is exclusively a simulation framework; a protocol authored for NS2 must be re-implemented to test in deployment (Demmer, Levis, et al. 2005: 2). Suppose that there is an environment that runs identical application code in simulation and in deployment, a particular implementation can be examined in simulation as well on actual hardware, and multiple implementations can be compared in terms of code complexity, size, and execution time.

On the other hand, due to the simulations in NS2 are very detailed for the packet data, during the simulation process huge amount of packets are created, results that it can not simulate the large-scale networks, special for Wireless Sensor Networks (WSN) scenarios.

A simulator model of a real-world system is necessarily a simplification of the real-world system itself. Now we describe some of the limitations of the simulation model embodied in the current release of NS2:

- **TCP**

The simulator model for one-way TCP is described previously. There is no dynamic window advertisement, segment and ACK number computations are in units of packets, and there is no SYN/FIN connection establishment/teardown (Information Sciences Institute 2002).

- **Two-Way TCP (FullTCP)**

The simulator model for two-way TCP is described in Section 17.3 on the NS Manual (Fall & Varadhan 2000: 170-172). It is very similar to a 4.x BSD TCP, except there is no dynamic window advertisement, no 2MSL-wait or persist states, no urgent data, and no RESET segments. Recently, SACK, Newreno, and Tahoe functionality have been added to FullTCP.

Limitations to FullTCP: There is not a complete validation test which suites for FullTCP. For example, BugFix_ does not work correctly for FullTCP. The test for BugFix_, "ns test-suite-simple-full.tcl tahoe4", has been commented out from test-suite-simple-full.tcl. (Information Sciences Institute 2002.)

Except the previous mentioned limitations, one of the biggest drawbacks to NS2 is too difficult to master for the beginners. There are several reasons result that: firstly, the content of NS2 is very huge, the official NS manual can not update regularly, which makes the beginner hard to understand; secondly, a lot of relative knowledge and tools are involved to operate NS2 efficiently.

Furthermore, as an open source software NS2 is not exploited by the same company or person. That means version format maybe developed very quickly and sometimes even

without unification, while the documentation is often limited and out of date with the current release of the simulator. The consecution of the NS manual is also not very well. Else the code consistency is lacking at times in the code base and across releases.

However, the complete set of some paid simulators modules provides more features than NS2, and they therefore will be more attractive to network operators

Finally, there is a lack of tools to describe simulation scenarios and analyze or visualize simulation trace files. These tools are often written with scripting languages. The lack of generalized analysis tools may lead to that, that different people measure different values for the same metric names.

Fortunately, most current limitations can be overcome by consulting the highly dynamic newsgroups and browsing the source code.

.

7. CONCLUSIONS

NS2 is widely used to simulate and emulate telecommunication networks. And with its rich libraries of network and protocol objects, it can simulate most aspects of network technology. The results of the simulation are validated, which makes NS2 to be one of the most favorable simulation software which is widely used in education and research.

This thesis gives a particular description of the NS2 installation under different platforms. Furthermore, the structures and main operation principles are also presented in detail.

Many useful scenarios for the simulations have been presented, which can help the NS2 users to familiarize with the method of TCL script coding and analyze the results. It also illustrates how to construct an emulation environment using NS2 and MATLAB. This joining of NS2 and MATLAB can considerably enhance the application of NS2 for real system simulations. Different examples are given to demonstrate how to proceed with NS2 and MATLAB.

Although NS2 is a very strong network simulation tool, it has many limitations and disadvantages were discussed in Chapter 6.

In the future work we will implement NS2 for simulations of different communication scenarios, such as: Universal Mobile Telecommunications System (UMTS), Ultra-Wideband (UWB), Worldwide Interoperability for Microwave Access (WiMAX) and Satellite Network communications.

BIBLIOGRAPHIES

Altman, Eitan & Tania. Jimene (2003). NS Simulation for Beginners, 76, 111-125, Univ. de Los Andes, Merida, Venezuela and ESSI, Sophia-Antipolis, France.

Belding-Royer, Elizabeth (2007). AODV Description [online] [cited 2008-4-12]. Available from Internet: <URL: http://moment.cs.ucsb.edu/AODV/aodv.html>.

Bertesekas, Dimitri & Robert Gallager (1992). Data Networks second edition, 124. Prentice Hall, Upper Saddle River, NJ 07458 1992.

Buchheim, Tim (2002). Nam: Network Animator [online] [cited 2008-1-24]. Available from Internet: <URL: http://www.isi.edu/nsnam/nam/>.

Chung, Jae & Mark Claypool A (2003). NS2 overview [online] [cited 2007-10-14]. Available from Internet: <URL: http://nile.wpi.edu/NS/overview.html>.

Chung, Jae & Mark Claypool B (2003). Simple Simulation Example [online] [cited 2008-3-23]. Available from Internet:
<URL: http://nile.wpi.edu/NS/simple_ns.html>.

Cygwin homepage (2008) [online] [cited 2008-1-16]. Available from Internet:
<URL: http://www.cygwin.com/>.

Demmer, Michael, Philip Levis, August Joki, Eric Brewer & David Culler (2005). TYTHON: A DYNAMIC SIMULATION ENVIRONMENT FOR SENSOR NETWORKS, 2. University of California, Berkeley Computer Science Division Berkeley, CA 94720.

DSR wikipedia (2006) [online] [cited 2008-4-12]. Available from Internet: <URL: http://en.wikipedia.org/wiki/Dynamic_Source_Routing>.

Fall, Kevin & Kannan Varadhan (2000). The ns Manual, 40-41, 63, 72-75, 93, 170-172, 336-341. UC Berkeley, LBL, USC/ISI, and Xerox PARC.

Fan, Qiang (2005). AWK manual [online] [cited 2007-11-10]. Available from Internet: <URL: http://fanqiang.chinaunix.net/program/other/2005-09-07/3621.shtml>.

Gnuplot homepage (2008) [online] [cited 2008-3-23]. Available from Internet: <URL: http://en.wikipedia.org/wiki/Gnuplot>.

Harding, Chris (2005). NS-2 Trace Formats [online] [cited 2008-1-24]. Available from Internet: <URL: http://k-lug.org/~griswold/NS2/ns2-trace-formats.html>.

Information Sciences Institute, (2002). Ns Limitations, [online] [cited 2008-4-6]. Available from Internet: <URL: http://www.isi.edu/nsnam/ns/ns-limitations.html>.

Information Sciences Institute, (2005). Building ns-2 on Cygwin [online] [cited 2007-11-15]. Available from Internet: <URL: http://www.isi.edu/nsnam/ns/ns-cygwin-old.html>.

Information Sciences Institute A (2006). The Network Simulator – ns-2 [online] [cited 2007-10-14]. Available from Internet: <URL: http://www.isi.edu/nsnam/ns/>.

Information Sciences Institute B, (2006). NS: change log [online] [cited 2007-11-29]. Available from Internet: <URL: http://www.isi.edu/nsnam/ns/CHANGES.html>.

Information Sciences Institute C, (2006). Network Emulation with the NS Simulator [online] [cited 2008-4-20]. Available from Internet: <URL: http://www.isi.edu/nsnam/ns/ns-emulation.html>.

Jitter wikipedia (2008) [online] [cited 2008-3-30]. Available from Internet: <URL: http://en.wikipedia.org/wiki/Jitter>.

Ke, Zhiheng (2004).Winxp + Cygwin + ns-allinone-2.29.2 setup [online] [cited 2008-2-5]. Available from Internet: <URL: http://140.116.72.80/~smallko/ns2/setup.htm>.

Malek, Jaroslaw (2007). Trace graph - Network Simulator NS-2 trace files analyzer [online] [cited 2008-4-15]. Available from Internet: <URL:http://www.tracegraph.com./>.

NS2 wikipedia (2008) [online] [cited 2007-11-28]. Available from Internet: <URL: http://en.wikipedia.org/wiki/Ns2>.

Nethi, Shekar, Mikael Pohjola, Lasse Eriksson & Riku Jantti (2007). “Platform for Emulating Networked Control Systems in Laboratory Environments”, to appear in Proc. IEEE Inter-national Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2007), Helsinki, Finland.

Nethi, Shekar, Chao Gao, Riku Jäntti & Mikael Pohjola (2007), Localized Multiple Next-hop Routing Protocol, appear in Proc.7th international conference on ITS telecommunication (ITST 2007), Paris, France.

Nilsson, Johan (1998). Real-time control systems with delays, 56-61. Ph.D. dissertation, Lund Institute of Technology, 1998.

OTCL Tutorial (1995) [online] [cited 2007-10-13]. Available from Internet: <URL: http://www.openmash.org/developers/docs/otcl-doc/doc/tutorial.html>.

Perkins, Charles & Pravin Bhagwat (2004). Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, 236-238. Computer Science Department University of Maryland College Park, MD 20742.

TODA wikipedia (2005) [online] [cited 2008-4-12]. Available from Internet: <URL: http://en.wikipedia.org/wiki/TORA>.

Wang, Jianping (2004). ns-2 Tutorial, 4. Multimedia Networking Group, the Department of Computer Science, UVA.

Xu, Leiming, Bo Pang & Yao Zhao (2003). NS and Network Simulation, 17-45. Posts & Telecommunications Press: ISBN 7-115-11867-1/TN.2213.

Xu, Leiming (2001). How to Add a New Protocol in NS2 [online] [cited 2008-4-23], Available from Internet: <URL: http://netarchlab.tsinghua.edu.cn/~zm/presentation/ns-extend-xlming.ppt>.

APPENDIX I

```
#creat a new ns file
set ns [new Simulator]

#set the defferent color for defferent application
$ns color 1 Blue
$ns color 2 Red

#open a new nam file named out.nam and save the process in it
set nf [open out.nam w]
$ns namtrace-all $nf

#open a tr file to save the process and save the process in it
set nd [open out.tr w]
$ns trace-all $nd

#finish function
proc finish {} {
        global ns nf nd
        $ns flush-trace
        #close the file
         close $nf
        close $nd
        #show the nam file
         exec nam out.nam &
        exit 0
}

#set the nodes s1:id0 s2:id1 r:id2 d:id3
set s1 [$ns node]
set s2 [$ns node]
set r [$ns node]
set d [$ns node]

#set the links connection from the source nodes to node r with bandwidth: 2Mbps,
#delay: 10ms, queue model:DropTail
$ns duplex-link $s1 $r 2Mb 10ms DropTail
$ns duplex-link $s2 $r 2Mb 10ms DropTail

#connect node r to the direction node d with bandwidth: 1.7Mbps, delay: 10ms, queue
```

```
#model:DropTail
$ns duplex-link $r $d 1.7Mb 20ms DropTail

# set Queue Limit:10 packets in the link r - d
$ns queue-limit $r $d 10

#observe the queue change between r - d
$ns duplex-link-op $r $d queuePos 0.5

#set tcp connection
set tcp [new Agent/TCP]
$ns attach-agent $s1 $tcp
set sink [ new Agent/TCPSink]
$ns attach-agent $d $sink
$ns connect $tcp $sink

#plot the tcp in blue
$tcp set fid_ 1

#establish FTP on tcp
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#set udp connection
set udp [new Agent/UDP]
$ns attach-agent $s2 $udp
set null [new Agent/Null]
$ns attach-agent $d $null
$ns connect $udp $null

#plot the udp in red
$udp set fid_ 2

#establish cbr on udp
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR

#set the size of packet 1K bytes
$cbr set packet_size_ 1000
```

```
#rate of cbr 1M bps
$cbr set rate_ 1mb
$cbr set random_ false

#set the time when FTP & CBR start and stop
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#detach the tcp connection
$ns at 4.5 "#ns detach-agent $s1 $tcp"
$ns at 4.5 "$ns detach-agent $d $sink"

#call the finish function
$ns at 5.0 "finish"

#run it
$ns run
```

APPENDIX II

```
BEGIN {

# beginning,set the record the highest packet ID
      highest_packet_id = 0;
}

{     action = $1;
      time = $2;
      flow_id = $8;
      packet_id = $12;

# record the highest current ID
      if ( packet_id > highest_packet_id )
      highest_packet_id = packet_id;

# record the flow ID and time
      if ( start_time[packet_id] == 0 )
      start_time[packet_id] = time;
      flow_num[packet_id]=flow_id;

# record the obtain time of all the packet
      if ( action == "r" ) {
      end_time[packet_id] = time;
      } else {

# the time of the loss packet is -1
```

```
        end_time[packet_id] = -1;
        }
}

END {

# calculate end-to-end delay
        for ( packet_id = 0; packet_id <= highest_packet_id; packet_id++ ) {
        packet_duration = end_time[packet_id] - start_time[packet_id];

# save the result
        if (packet_duration > 0) {
        if (flow_num[packet_id]==1){
        printf("%f %f\n", start_time[packet_id], packet_duration) > "ftp_delay";
        } else {
        printf("%f %f\n", start_time[packet_id], packet_duration) > "cbr_delay";
                        }
                }
        }
        }
```

APPENDIX III

```
BEGIN {
    fsDrops = 0;
    numFs = 0;
}

{
    action = $1;
    time = $2;
    node_1 = $3;
    node_2 = $4;
    type = $5;
    flow_id = $8;
    node_1_address = $9;
    node_2_address = $10;
    seq_no =$11;
    packet_id = $12;
if ( node_1==1&&node_2==2&&action=="+" )
                numFs++;
            if ( flow_id == 2&&action == "d" )
              fsDrops++;
}

END {
        printf ("number of packets sent:%d lost:%d\n",numFs,fsDrops);
}
```

APPENDIX IV

```
BEGIN {
    old_time = 0;
    old_seq_no = 0;
    i = 0;
}

{
    action = $1;
    time = $2;
    node_1 = $3;
    node_2 = $4;
    type = $5;
    flow_id = $8;
    node_1_address = $9;
    node_2_address = $10;
    seq_no =$11;
    packet_id = $12;

    if ( node_1==2&&node_2==3&&type=="cbr"&&action=="r" ) {
                dif = seq_no - old_seq_no;

    if ( dif == 0 )    {
        dif = 1;
    }

    jitter[i] = (time - old_time)/dif;
```

```
    seq[i] = seq_no;
    i = i + 1;
    old_seq_no = seq_no;
    old_time = time;
    }
}

END {
        for ( j = 1; j < i ; j++ ) {
        printf ("%d\t%f\n",seq[j],jitter[j]);
     }
}
```

APPENDIX V

```
BEGIN {
    init = 0;
    i = 0;
}

{
    action = $1;
    time = $2;
    node_1 = $3;
    node_2 = $4;
    type = $5;
    pktsize = $6;
    flow_id = $8;
    node_1_address = $9;
    node_2_address = $10;
    seq_no =$11;
    packet_id = $12;

    if ( action=="r"&&node_1==2&&node_2==3&&flow_id=="2" ) {
                pkt_byte_sum[i+1]=pkt_byte_sum[i]+pktsize;

    if ( init == 0 ) {
                start_time=time;
               init = 1;
     }
```

```
        end_time[i] = time;
        i = i + 1;
        }
}

END {
            printf ("%.2f\t%.2f\n",end_time[0],0);
            for (j = 1; j < i; j++){
            th = pkt_byte_sum[j]/(end_time[j] - start_time)*8/1000;
            printf("%.2f\t%.2f\n",end_time[j],th);
            }
        printf("%.2f\t%.2f\n",end_time[i-1],0);
}
```

APPENDIX VI

```
BEGIN {
    fsDrops = 0;
    numFs = 0;
  }

{
    action = $1;
    time = $2;
    node_1 = $3;
    node_2 = $4;
    type = $5;
    flow_id = $8;
    node_1_address = $9;
    node_2_address = $10;
    seq_no =$11;
    packet_id = $12;

    if ( node_1==1&&node_2==2&&action=="+" )
                numFs++;

    if ( flow_id == 2&&action == "d" )
              fsDrops++;
}

END {
          printf ("number of packets sent:%d lost:%d\n",numFs,fsDrops);          }
```

APPENDIX VII

```
#creat a new ns file
set ns [new Simulator]

#open a new nam file named out and save the process in it
set nf [open many.nam w]
$ns namtrace-all $nf

#open a tr file and save the process
set tf [open many.tr w]
$ns trace-all $tf

set windowVsTime [open win w]
set param [open parameters w]

#finish function
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #close the file
          close $nf
          close $tf
        #show the nam file
          exec nam many.nam &
          exit 0
}
```

```
#set the nodes
set n0 [$ns node]
set n1 [$ns node]

#r - d bandwidth 1.7Mbps; delay:10ms; queue model:DropTail
$ns duplex-link $n0 $n1 0.7Mb 20ms DropTail

set NumSrc 5
set Duration 10

#set the other nodes
for {set j 1} {$j <= $NumSrc} {incr j} {
set S($j) [$ns node]}

#creat a random generator
set rng [new RNG]
$rng seed 0

set RVdly [new RandomVariable/Uniform]
$RVdly set min_ 1
$RVdly set max_ 5
$RVdly use-rng $rng

set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng
```

```
#set random delay for the nodes
for {set i 1} {$i <= $NumSrc} {incr i} {
set dly($i) [expr [$RVdly value]]
set startT($i) [expr [$RVstart value]]
puts $param "dly($i) $dly($i) ms"
puts $param "startT($i) $startT($i) sec"}

for {set j 1} {$j <= $NumSrc} {incr j} {
$ns duplex-link $S($j) $n1 10Mb $dly($j)ms DropTail
$ns queue-limit $S($j) $n1 100}

#observe the queue change between r - d
$ns duplex-link-op $n1 $n0 queuePos 0.5
$ns queue-limit $n1 $n0 10

#set tcp source
for {set j 1} {$j <= $NumSrc} {incr j} {
set tcp_src($j) [new Agent/TCP/Reno]}

#set tcp destination
for {set j 1} {$j <= $NumSrc} {incr j} {
set tcp_snk($j) [new Agent/TCPSink]}


#the beginning of tcp is S($i)
for {set j 1} {$j <= $NumSrc} {incr j} {
$ns attach-agent $S($j) $tcp_src($j)
$ns attach-agent $n0 $tcp_snk($j)
```

```
$ns connect $tcp_src($j) $tcp_snk($j)}

#establish FTP on tcp
for {set j 1} {$j <= $NumSrc} {incr j} {
set ftp($j) [$tcp_src($j) attach-source FTP]}
for {set j 1} {$j <= $NumSrc} {incr j} {
$tcp_src($j) set packetSize_ 552}
for {set i 1} {$i <= $NumSrc} {incr i} {
$ns at $startT($i) "$ftp($i) start"
$ns at $Duration "$ftp($i) stop"}

#call the finish function
$ns at [expr $Duration] "finish"

#run it
$ns run
```

APPENDIX VIII

```
#creat a new ns file
set ns [new Simulator]

#open a new nam file named out and save the process
set nf [open uni.nam w]
$ns namtrace-all $nf

#open a tr file to save the process
set tf [open uni.tr w]
$ns trace-all $tf

#finish function
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #close the file
          close $nf
          close $tf
        #show the nam file
          exec nam uni.nam &
          exit 0
}

$ns color 1 blue
$ns color 2 red
```

```
$ns rtproto DV

set node 5

#set the nodes
for {set j 0} {$j <= $node} {incr j} {
set S($j) [$ns node]}
$ns duplex-link $S(0) $S(1) 0.3Mb 10ms DropTail
$ns duplex-link $S(1) $S(2) 0.3Mb 10ms DropTail
$ns duplex-link $S(2) $S(3) 0.3Mb 10ms DropTail
$ns duplex-link $S(1) $S(4) 0.3Mb 10ms DropTail
$ns duplex-link $S(3) $S(5) 0.5Mb 10ms DropTail
$ns duplex-link $S(4) $S(5) 0.5Mb 10ms DropTail

set tcp [new Agent/TCP/Newreno]
$ns attach-agent $S(0) $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $S(5) $sink
$ns connect $tcp $sink
$tcp set fid_ 1

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

$ns rtmodel-at 1.0 down $S(1) $S(4)
$ns rtmodel-at 3.0 up $S(1) $S(4)
```

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

#call the finish function
$ns at 5.0 "finish"

#run it
$ns run
```

APPENDIX IX

```
#set the parameters for the wireless channel
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)             2                            ;# number of mobilenodes
set ns_         [new Simulator]
set nd [open out1.tr w]
$ns_ trace-all $nd

set nf [open out1.nam w]
$ns_ namtrace-all-wireless $nf 100 100

#define the finish function
proc finish {} {
global ns nf nd
$ns_ flush-trace
close $nf
close $nd
exec nam out1.nam &
exit 0
```

```
}

set topo [new Topography]
$topo load_flatgrid 100 100
create-god $val(nn)

# Configure nodes
$ns_ node-config -adhocRouting $val(rp) \
                                 -llType $val(ll) \
                                 -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

# Provide initial (X,Y, for now Z=0) co-ordinates for node_(0) and node_(1)
for {set i 0} {$i < $val(nn) } {incr i} {
                        set node_($i) [$ns_ node ]
                        $node_($i) random-motion 0              ;# disable random motion
            }
```

```
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 90.0
$node_(1) set Y_ 85.0
$node_(1) set Z_ 0.0

# Node_(1) starts to move towards node_(0)
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
$ns_ at 10.0 "$node_(0) setdest 20.0 18.0 1.0"

# Node_(1) then starts to move away from node_(0)
$ns_ at 100.0 "$node_(1) setdest 49.0 48.0 15.0"

# 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"

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
```

```
        $ns_ at 150.0 "$node_($i) reset";
}

$ns_ at 150.0001 "stop"
$ns_ at 150.0002 "puts \"NS EXITING...\" ; $ns_ halt"
#stop function
proc stop {} {
        global ns_ nf nd
        $ns_ flush-trace
        close $nf
        close $nd
        exec nam out1.nam &
        exit 0
}

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

APPENDIX X

```
# Define options
set opt(chan)           Channel/WirelessChannel
set opt(prop)           Propagation/FreeSpace
set opt(netif)          Phy/WirelessPhy
set opt(mac)            Mac/Simple
set opt(ifq)            Queue/DropTail/PriQueue
set opt(ll)             LL
set opt(ant)            Antenna/OmniAntenna
set opt(x)              500     ;# X dimension of the topography
set opt(y)              500     ;# Y dimension of the topography
set opt(ifqlen)         100      ;# max packet in ifq
set opt(seed)           0.0
set opt(tr)             Simple.tr       ;# trace file
set opt(nm)             Simple.nam ;#nam file
set opt(adhocRouting)   AODV ;#Routing table
set opt(nn)             27       ;# how many nodes are simulated
set opt(stop)           500      ;# simulation time

# set the source file path
set opt(mobility)       "/home/simulation/simulation_files/mobility080303.txt"
set opt(location)       "/home/simulation/simulation_files/NodesPosition"
set opt(mnode)          "/home/simulation/simulation_files/mobile_node.txt"
set opt(CommRange)      "/home/simulation/simulation_files/CommRange.txt"

set ns_ [new Simulator];            # Intialize simulator
$ns_ use-scheduler RealTime;        # Real time schedular
```

```
set wtopo [new Topography]
# create trace object for ns and nam
set tracefd    [open $opt(tr) w]
$wtopo load_flatgrid $opt(x) $opt(y)
set namtrace [open $opt(nm) w]
$ns_ trace-all $tracefd
set tracefd    [open $opt(tr) w]
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)
$ns_ use-newtrace
set god_ [create-god $opt(nn)]
set chan_1_ [new $opt(chan)]

# Configure 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) \
                 -channel $chan_1_    \
                 -energyModel "EnergyModel"    \
                 -initialEnergy 100 \
                 -rxPower 0.3 \
                 -txPower 0.3 \
                 -topoInstance $wtopo \
                 -agentTrace ON\
```

```
            -routerTrace ON\
            -macTrace OFF

#set the communication range
$opt(netif) set RXThresh_ 1.20174e-09
set nn1 [expr $opt(nn)]

for {set i 0} {$i < $nn1} {incr i} {
    set node_($i) [$ns_ node]

     #disable random motion
    $node_($i) random-motion 0
     }

set n 0;

#read from the source file which has been defined
source $opt(location)
source $opt(mnode)

$node_(25) color "red"
$node_(25) shape "box"
$node_(25) set X_ 300.0
$node_(25) set Y_ 100.0
$node_(25) set Z_ 0.0

set c 22200
```

```
for {set i 25} {$i    <    27} {incr i 1} {

# Create a TCPTap Agent
set tap($i) [new Agent/Tap];
set ipnet($i) [new Network/IP];              # Create a Network agent
$ipnet($i) open writeonly
$tap($i) network $ipnet($i);                    # Connect network agent to tap agent
$ns_ attach-agent $node_($i) $tap($i);              # Attach agent to the node.
}

set k 0;

for {set i $i} {$i < [expr    $opt(nn)+27]} {incr i 1} {
set p [expr    $c+$k]

# Configure the Entry point
set tap($i) [new Agent/Tap];               # Create the TCPTap Agen
set bpf($i) [new Network/Pcap/Live];      # Create the bpf
set dev [$bpf($i) open readonly eth0]
$bpf($i) filter "src 130.233.125.158    and src port $p"
$tap($i) network $bpf($i);                      # Connect bpf to TCPTap Agent
$ns_ attach-agent $node_($k) $tap($i);              # Attach TCPTap Agent to the node
incr k;
}

source $opt(mobility)
source $opt(location)
```

```
for {set i 0} {$i < $opt(nn)} {incr i} {
    $ns_ initial_node_pos $node_($i) 20
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $opt(nn) } {incr i} {
     $ns_ at $opt(stop).000000001 "$node_($i) reset";
     }

# tell nam the simulation stop time
$ns_ at    $opt(stop).000000001 "$ns_ halt"

# "puts \"NS EXITING...\" ; $ns_ halt"
puts "Starting Simulation..."
$ns_ run
```