VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Junbo Cai

Improvement of Java-based NS-2 Visualizer

Information Technology

2011

VAASAN AMMATTIKOREAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme of Information Technology

ABSTRACT

Author	Junbo Cai
Title	Improvement of Java-based NS-2 Visualizer
Year	2010
Language	English
Pages	53
Supervisor	Gao Chao

NS-2 (Network Simulator version 2) is the most popular open-source network simulation program. NS-2 kernel is made in C++ but simulation scenario design is done in TCL (Tool Command Language). TCL is not a popular language for most networking researchers and engineers; therefore a user-graphic interface program is highly appreciated. We are aimed to design this software and denote it as NSSV (NS-2 Scenario Setup Visualizer).

In this project, my work is to continue the leading work done by Miss Huang Cheng to improve the functions (mobile network simulation) of such a program. The program was designed using Java language so that it is compatible in different operating systems. Due to the limit of time and workload, Huang Cheng did not implement the function of mobile network simulation setting. By carefully reviewing the source code, as well as based on my deep understanding of wireless network communications. I have implemented mobile network simulation scenario design by enabling traffic generating function and node mobility generating function.

In the new version of this program a user is able to set up a mobile network simulation easily, and the TCL scripts and scenarios can be generated with the NSSV program.

CONTENT

LIST of ABBREVIATIONS

AODV	Ad-hoc On-demand Distance Vector
API	Application Programming Interface
CBQ	Class Based Queuing
CBR	Constant Bit Rate
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
FTP	File Transfer Protocol
IDE	Integrated Development Environment
IQT	Interface Queue Type
ISI	Information Sciences Institute
GUI	Graphical User Interface
NS-2	Network Simulator ver.2
NSSV	NS-2 Scenario Setup Visualizer
PUMA	Protocol for Unified Multicasting through Announcements
TCL	Tool Command Language
TCP	Transmission Control Protocol
TORA	Temporally-Ordered Routing Algorithm
UDP	User Datagram Protocol
VBR	Variable Bit Rate
WLAN	Wireless Local Area Network

1. INTRODUCTION

Network Simulator (Version 2), is widely known as NS-2, which provides support for simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP (Transmission Control Protocol), UDP (User Datagram Protocol)).

NS2 is an object-oriented network simulator, which is essentially a discrete event simulator. NS2 has a virtual clock, all the simulation by the discrete event-driven. Nowadays, NS2 simulation can be used for a variety of communication networks. NS-2 has the strong functions and great modules, has achieved some of the simulation modules: Network Transfer Protocol module, e.g. TCP and UDP; Business Source Flow Generator module, e.g. FTP (File Transfer Protocol), Telnet, CBR (Constant Bite Rate) and VBR (Variable Bit Rate); Routing Queue Management module, e.g. Droptail, Red and CBQ (Class Based Queuing); Routing Arithmetic module, e.g. Dijkstra, WLAN (Wireless Local Area Network), Ad hoc Router, Mobile IP Wireless Network and Satellite Communication Network. NS-2 also has achieved multicast and some MAC sub-layer protocol for local network simulation.

However, NS2 was developed by C++ and OTCL languages. But OTCL language is not popular for most users.

This application was to design a Graphical User Interface to generating the NS codes for network simulation. We use Java as the platform for this software, so that the users can use this program on different operating systems.

1.1 Rationale of this project

NS-2 is the most popular open-source network simulation program, but simulation scenario design is done in TCL. TCL is not a popular language for most networking researchers and engineers. Therefore the existing program mentioned above is also made for most networking researchers and engineers as a user-graphic interface, but a program made in Java can work on any operating system that supports Java.

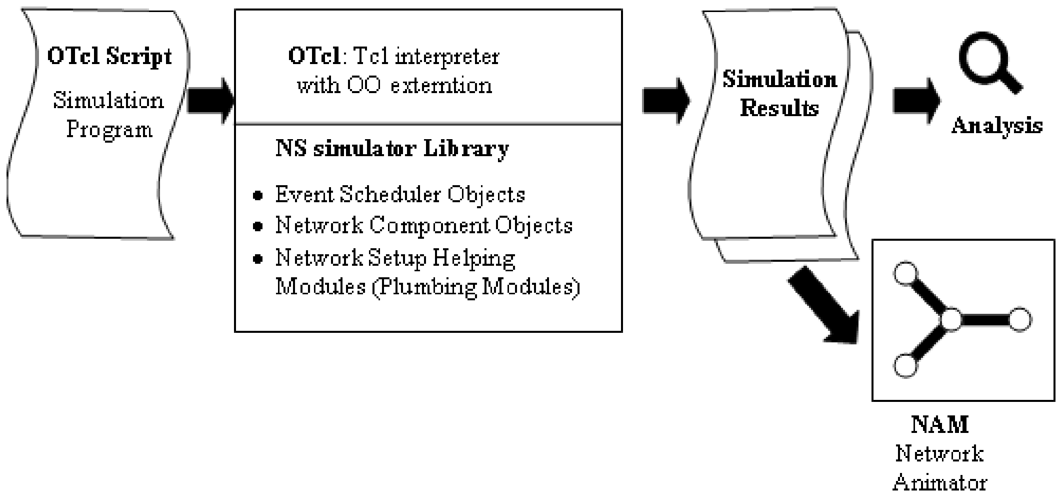
The purpose of the project is to develop a GUI (Graphical User Interface) that is used to generate the TCL scripts. Simple use friendly interface will allow user to generate the TCL scripts easily. My work is to continue the previous project done by Huang Cheng. And my task is to improve the connection pattern and node movement for the mobile/wireless simulation and to make an automatic traffic generation.

My thesis structure is started from the introduction of my work; see what is my work, why this work has to be done and how did I do it. The rest of the thesis is arranged as follows: In Chapter 2 I briefly introduce NS-2 background, explain how wireless network simulation works. In Chapter 3 I detailed describe what the function of this project is. In Chapter 4 I analyze and design the project by together software. In Chapter 5 I focus on implementation by codes and results. Finally in Chapter 6 I conclude my work.

2. BACKGROUND of NS-2

2.1 NS-2 working flow

A Simulation program in NS-2 is designed by OTcl script, using the NS simulator library (Event scheduler objects, Network component objects and Network setup helping modules) to compile and simulate by OTcl interpreter. NS2 outputs either text-based or animation-based simulation results. To analyze a particular behavior of the network, uses can extract a relevant record of simulation results to generate the network topology picture, or data visualization charts. Figure 1 shows the basic architecture of NS-2.



OTcl Script
Simulation Program
OTcl: Tcl interpreter with OO externtion
NS simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

Figure1: Basic architecture of NS-2 [4]

2.2 NS-2 Structures

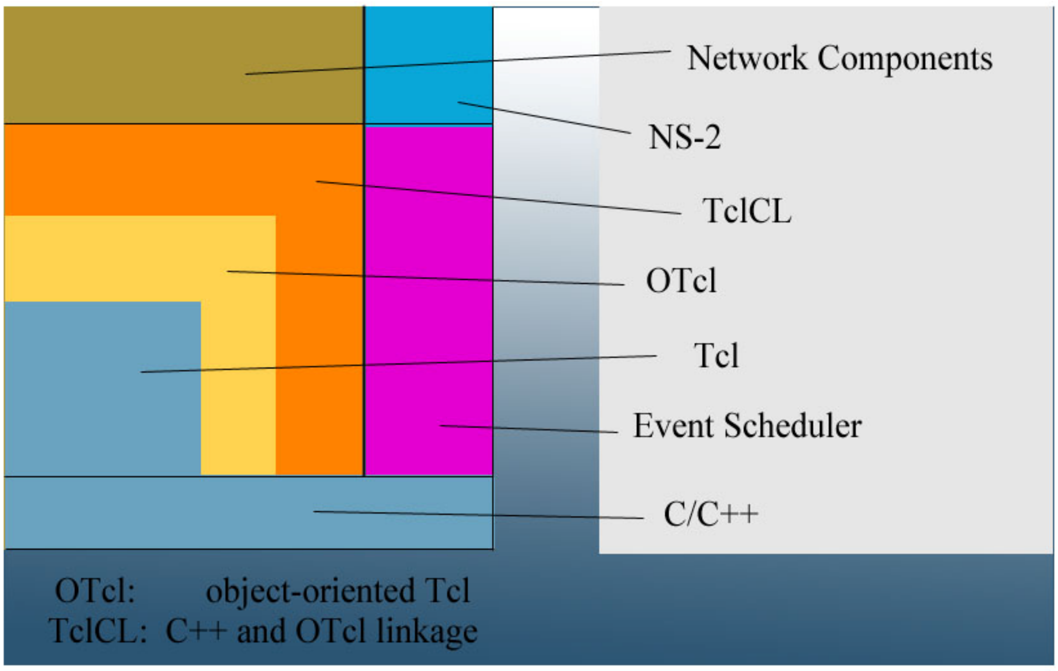


Figure 2: NS-2 Structure [4]

OTcl is the object-oriented extension which is established over the part of Tcl. Event scheduler and Network component these tow parts are written by C++. And C++ in the lowest layer means C++ is the core of NS2. TclCl is linkage between OTcl and C++. And TclCl is "Tcl with classes" which is an interface between C++ and Tcl.

In a word, when we simulate a new protocol, firstly we need to describe the protocol by C++ language. This includes deciding for protocol specification and processing for protocol version. Then we establish a simulator by Tcl script, and define the protocol for MAC layer, network layer and etc. which we need to simulate in this simulator. Finally it is to be analyzed its characteristics by simulating network activity.

2.3 NS contains OTcl and C++ languages

OTcl is short for Object Tcl, an extension to Tcl/Tk for object-oriented programming, used as a front-end to setup the simulator, configure objects and schedule events. Here is the function of OTcl code.

- Used to build the network structure and topology this is just the surface of your simulation;
- Easily to configure your network parameters;
- Not enough for research schemes and protocol architecture adaption.

C++ is most important and kernel part of the NS2, used for the creation of objects because of speed and efficiency. Here is the function of C++ scripts.

- To implement the kernel of the architecture of the protocol designs;
- From the packet flow view, the processes run on a single node;
- To change or "comment out" the existing protocols running in NS2;
- Details of your research scheme.

2.4 Simulation of Wired Network

Nodes and Links are demanded in wired network simulation for creating the topologies. Agent and traffic frame can be attached to the nodes. And all the nodes should be connected by link, the agent need to be connected as well.

2.4.1 Nodes

There are two important roles of a node in NS2. A node acts as a router and a host. As a router, it forwards packets to the connecting link based on a routing table. As a host, it delivers packets to the transport layer agent attached to the port specified in the packet header.

2.4.2 Links

A link is an OTcl object which connects two nodes and carries packets from the beginning node to the terminating node. There are three link types in NS2 which are Simplex-Link, Duplex-Link and Duplex-Intserv-Link. And there are some attributes of each link in the following. Table1 lists some attributes of each link.

Bandwidth	Link bandwidth in bits per second
Delay	Link propagation delay in seconds
Queue Type	Link uses queue types: DropTail, Fair Queuing (FQ), Stochastic Fair Queuing (SFQ), Deficit Round Robin (DRR), Random Early-Detection (RED)
Orientation	Where the packets flow
Monitor position	Where to captured the flow
Queue Limit	Control the flow

Table1.The Attributes of each link [2]

2.4.3 Agents

An agent is a program that gathers information or performs some other service without your immediate presence and on some regular schedule. The agent includes enough internal state to assign fields to a simulated packet before it is sent. There are some agents supported in NS2. They show in following. Table2 lists some agents and specifications above.

TCP	a “Tahoe” TCP sender
TCP/Reno	a “Reno” TCP sender
TCP/Newreno	a modified Reno TCP sender
TCP/Sack1	a SACK TCP sender
TCP/Fack	a “forward” SACK TCP sender
TCP/FullTcp	a more full-functioned TCP with 2-way traffic
TCP/Vegas	a “Vegas” TCP sender
TCP/Vegas/RBP	a Vegas TCP with “rate based pacing”
TCP/Vegas/RBP	a Reno TCP with “rate based pacing”
TCP/Asym	an experimental Tahoe TCP for asymmetric links
TCP/Reno/Asym	an experimental Reno TCP for asymmetric links
TCP/Newreno/Asym	an experimental Newreno TCP for asymmetric links
TCPSink	a Reno or Tahoe TCP receiver
TCPSink/DelAck	a TCP delayed-ACK receiver
TCPSink/Asym	an experimental TCP sink for asymmetric links
TCPSink/Sack1	a SACK TCP receiver
TCPSink/Sack1/DelAck	a delayed-ACK SACK TCP receiver
UDP	a basic UDP agent
RTP	an RTP sender and receiver
RTCP	an RTCP sender and receiver
LossMonitor	a packet sink which checks for losses

IVS/Source	an IVS source
IVS/Receiver	an IVS receiver
CtrMcast/Encap	a “centralized multicast” encapsulator
CtrMcast/Decap	a “centralized multicast” de-encapsulator
Message	a protocol to carry textual messages
Message/Prune	processed multicast routing prune messages
SRM	an SRM agent with non-adaptive timers
SRM/Adaptive	an SRM agent with adaptive timers
Tap	interfaces the simulator to a live network
Null	a degenerate agent which discards packets
rtProto/DV	distance-vector routing protocol agent

Table2. The detail of agent [2]

Agent states for each type. All the states will be used in my project. Table3 lists the states of each agent.

Flow ID	the flow identifier
Priority	the ID priority field
Flag	packet flags
Time to live	default is 32
Class	the node class field
Address	Address of the attached node
Port number	Port where the agent is attached
Window size	window size in bytes
Packet size	packets size in bytes
Destination address	where it is sending packets to
Destination port	where it is directing packets to

Table3. The states of each agent [2]

2.4.4 Traffic Generators

Sitting on top of a transport layer agent, an application informs the attached agent of user demand. Application can be classified into traffic generators (Traffic/CBR, Traffic/Exponential, Traffic/Pareto) and simulated applications (FTP and Telnet).

FTP (File Transfer Protocol)

An NS2 FTP module does not need an input file. It simply informs an attached sending transport layer agent of a file size in bytes. Upon receiving user demand, the agent creates packets which can accommodate the file and forwards them to a connected receiving transport layer agent through a low-level network.

Telnet

Telnet is an interactive client-sever text-based application. Telnet is not implemented based on a predefined schedule, since its data traffic is created in response to user demand. NS2 models a Telnet application in the same way as it does for traffic generators: sending a fixed size packet for every randomized interval.

Traffic/CBR (Constant Bit Rate)

A CBR traffic generator creates a fixed size payload burst for every fixed interval. The parameters of a CBR traffic can be seen in Table 4.

Inst_Var	Default_value	Description
packetSize	210	Application payload size in bytes
rate	488×10^3	Sending rate in bps

random	0 (false)	If true, introduce a random time to the inter-burst transmission interval.
maxpkts	16^7	Maximum number of application payload packet that CBR can send

Table4. Instruction Variables of Traffic/CBR [2]

Traffic/Exponential

An exponential on/off traffic generator acts as a CBR traffic generator during an ON interval and does not generate any payload during an OFF interval. ON and OFF periods are both exponentially distributed. The parameters of a Traffic/Exponent can be seen in Table 5.

Inst_Var	Default_value	Description
packetSize	210	Application payload size in bytes
rate	64×10^3	Sending rate in bps during an ON period
burst_time	0.5	Average ON period in seconds
idle_time	0.5	Average OFF period in seconds

Table5. Instruction Variables of Traffic/Exponential [2]

Traffic/Pareto

A pareto on/off traffic generator does the same as an exponential on/off generator but the ON and OFF periods conforms to a Pareto distribution. The parameters of a Traffic/Pareto can be seen in Table 6

Inst_Var	Default_value	Description
packsize	210	Application payload in bytes
rate	64×10^3	Sending rate in bps during an ON period
burst_time	0.5	Average ON period in seconds
idle_time	0.5	Average OFF period in seconds
shape	1.5	A “Shape” parameter of a Pareto distribution

Table6. Instruction Variables of Traffic/Pareto [2]

2.5 Simulation of Wireless Network

The components of mobile networking are Packet Headers, Mobile nodes, Wireless channels and Forwarding and routing. Simulation of wireless network needs to configure the mobile nodes, movement path, and scenarios.

2.5.1 Packet Headers

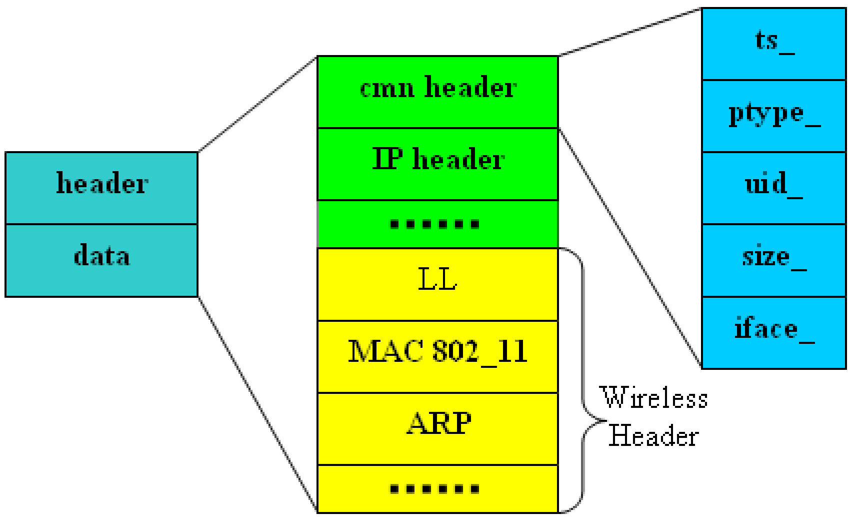


Figure3. Wireless Packet Format [5]

From the Figure3 we can know the main variables of wireless that are LL, MAC, Channel Type, Antenna Type and Interface Queue Type. So these variables will be used for wireless nodes in ns and configured in my project.

2.5.2 Mobile Nodes

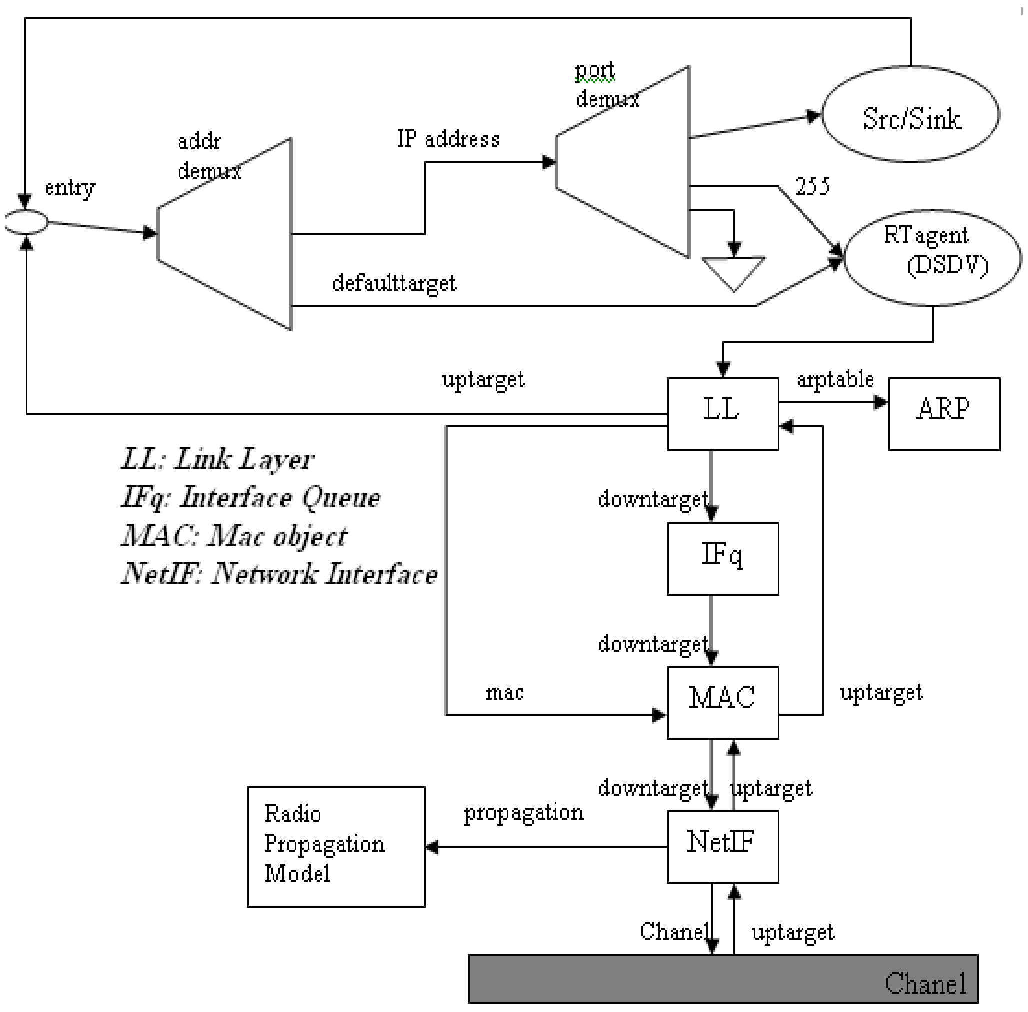


Figure4. Portrait of a Mobile Node [5]

Figure 4 is a mobile node processing under the CMU (Camegie Mellon University) Monarch (the name of the project) wireless extensions to NS-2. It shows the network components in the mobile node and the data path of sending and receiving packets.

The mobile node needs the parameters that are location (coordinates (x,y,z)) and movement (speed, direction, starting/ending location, time, and etc.).

2.5.2.1 Link Layer

Link Layer is the protocol layer with a separate ARP (Address Resolution Protocol) module. It handles the moving of data in and out in a network.

2.5.2.2 Interface Queue

Interface queue is a real time packet scheduler which gives priority to routing protocol packets.

2.5.2.3 MAC Layer

MacTdma and IEEE 802.11 Mac layer protocols are used in NS2. NS2 has used their implementation Distributed Coordination Function (DCF) from CMU.

2.5.2.4 Network Interfaces

NS uses Phy/WirelessPhy as wireless media interface to access the channel. Wireless media interface subject is collisions; the radio propagation model receives packets transmitted by other node interfaces to the channel.

2.5.2.5 Radio Propagation Model

NS-2 manual is given the math expression of these 3 models: TwoRayGround, Shadowing and FreeSpace.

2.5.2.6 Antenna

OmniAntenna and DirAntenna are used by mobile nodes as the antenna type. And antenna provides a good abstraction to wireless networking simulation.

OmniAntenna in NS-2 is a virtual Omnidirectianal antenna system.

DirAntenna in NS-2 is a virtual Directional Antenna system.

2.5.3 Wireless channel

The wireless channel simulates the transmission of packets at the physical layer. It is the receiver's responsibility to decide if it will accept the packets.

2.5.4 Forwarding and routing

The five different ad hoc routing protocols currently implemented for mobile networking in NS-2 are:

- *Destination-Sequenced Distance-Vector* (DSDV)
- *Dynamic Source Routing* (DSR)
- *Ad hoc On-demand Distance Vector* (AODV)
- *Temporally-Ordered Routing Algorithm* (TORA)
- *Protocol for Unified Multicasting Through Announcements* (PUMA)

In NS2 forwarding and routing function is archived by a classifier object. The NS nodes contain many different classifiers. Different classifiers have different tasks. When one extends the functionality of the node, more classifiers are added into the base node, and each of these blocks needs its own classifiers. Multiple classifier objects, each looks at a specific portion of the packet forward the packet through the node. The node contains the base routing module.

3. APPLICATION DESRIPTION

The main functionality of this program is the generation of the TCL code. And the purpose of this design is made the TCL to be used simply for the NS users.

3.1 Requirement Analysis

At this application there are two parts tasks. One was done by Huang Cheng. Now I am going to do another one. Here is a list of functions which the program must include in my application.

- Traffic generation has to be set manually with program Actually NS-2 contains an automatic traffic generation utility “cbrgen” which can generate traffic randomly by the given parameters
- Another improvement is for the mobile/wireless network simulation. NS-2 contains another utility “setdest”. With this utility, a random movement path for all the mobile nodes in the simulation can be randomly set up.

The program should have the following functions:

- All the implementation should cover all the parameter setting for both of “cbrgen” and “setdest” utilities.
- The program should generate a command to be executed as TCL file.

It is nice to have these functions now.

3.2 Accessing the functions

The function should be accessed simply in the normal way in which parameters are usually set in programs.

Firstly the program is an executable jar file, users need the java environment in own computer. There two parts need to pay attention. One is “cbrTraffic”, which can generate traffic randomly by “cbrgen” utility. In this case, cbrgen.tcl file should be run firstly in Linux Operating System, and then continue the next tcl file. On the other hand, that is “setdest” for creating a node-movement scenario randomly. In this part, this is the “./setdest” makefile, which should be run firstly in Linux Operating System. The rest can be generated by the given parameters, and then executed in Linux environment.

3.3 Users interface

There should be a main window that has simulation parameters, script content, network layout and five main tabs. Script content displays the script here. Main tabs are Node, Link, Agent, cbrtraffic and setdest.

Normally a wired simulation TCL file needs three parts in this program; they are Node, Link and Agent. And cbrTraffic is used optionally, it depends on using random traffic or not. But a wireless simulation TCL file does not need Link part.

When the script generation was done, saves as TCL file. And run it in Linux environment.

3.4 Analysis and Design

For software applications, the first and the most important thing are to analyze and know the logic of TCL before implementation. It is important to have the Java Swing knowledge since it benefits any program and makes the implementation easier.

3.4.1 UML (Unified Modeling Language) Diagrams

The UML design of this program is done with Borland Together 6.1.

3.4.1.1 Main Components

In the following diagram, it shows the main components in the main window.

We can see there is only one actor which is user self. User is able to use the program components setting the parameters for Link, Node, Select, Agent, cbrtraffic, setdest, Network Layout, Simulation Parameters as well as Script content. All of these cases apply for generating TCL code.

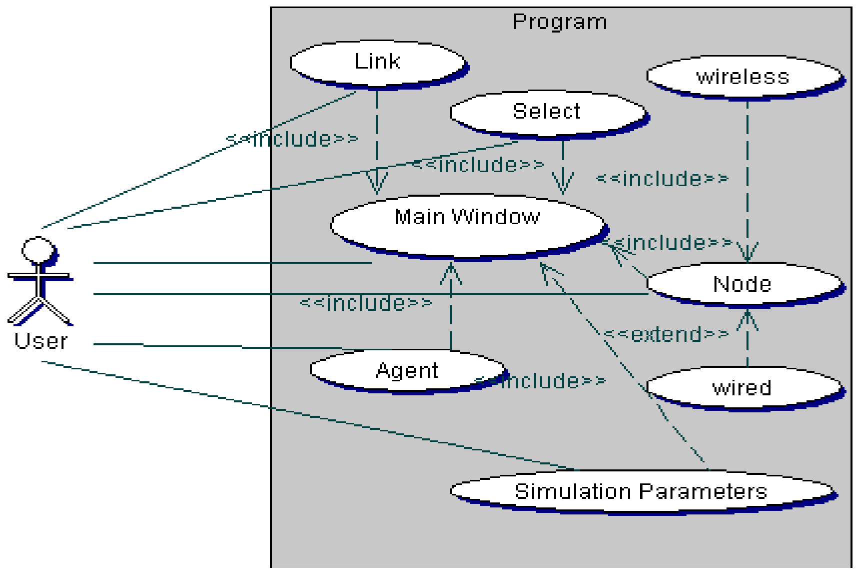


Figure5. Use Case diagram

3.4.1.2 Application Main Modules

Application is divided into three main packages: MainPackage, GUI, and NetworkComponents. DrawAction package is for selecting and drawing node or link. MainPackage package is for the main window function and the GUI package is for the entire NS frame. The package NetworkComponents is for all the components of NS2.

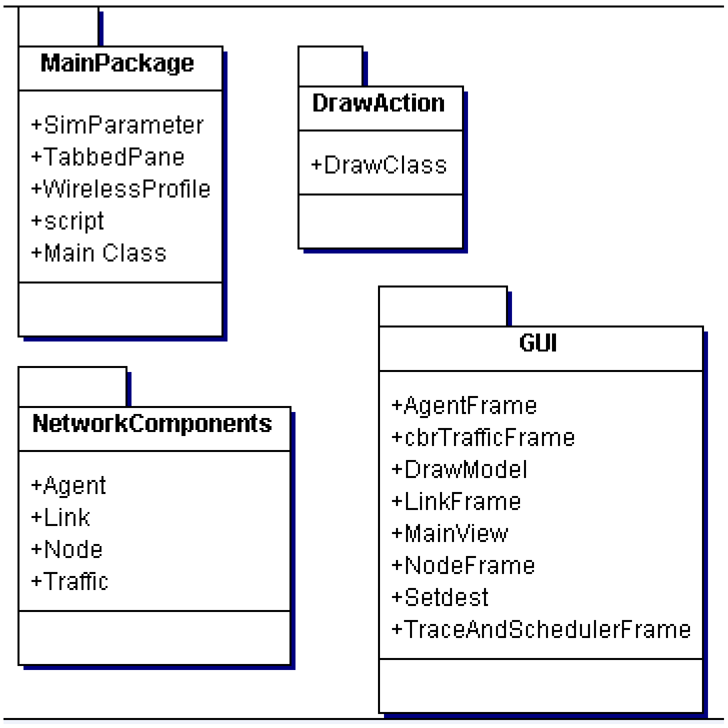


Figure6. Components diagram

3.4.1.3 Class Diagrams

Here we take a more detailed look at the class and roles of each class.

Figure 7 shows the class diagram in package GUI. There are eight classes in this package: MainView, NodeFrame, LinkFrame, AgentFrame, TraceAndSchedulerFrame, cbrTrafficFrame, Setdest, as well as

DrawModel. Most of these represent a window in the graphical user interface. The class called MainView is the window which shows the parameter field's layout. And the other classes NodeFrame, LinkFrame, AgentFrame, cbrTrafficFrame, cbrTrafficFrame and Setdest show their own component of simulation parameter. But TraceAndSchedulerFrame and DrawModel show the window of Script Content and Network Layout.

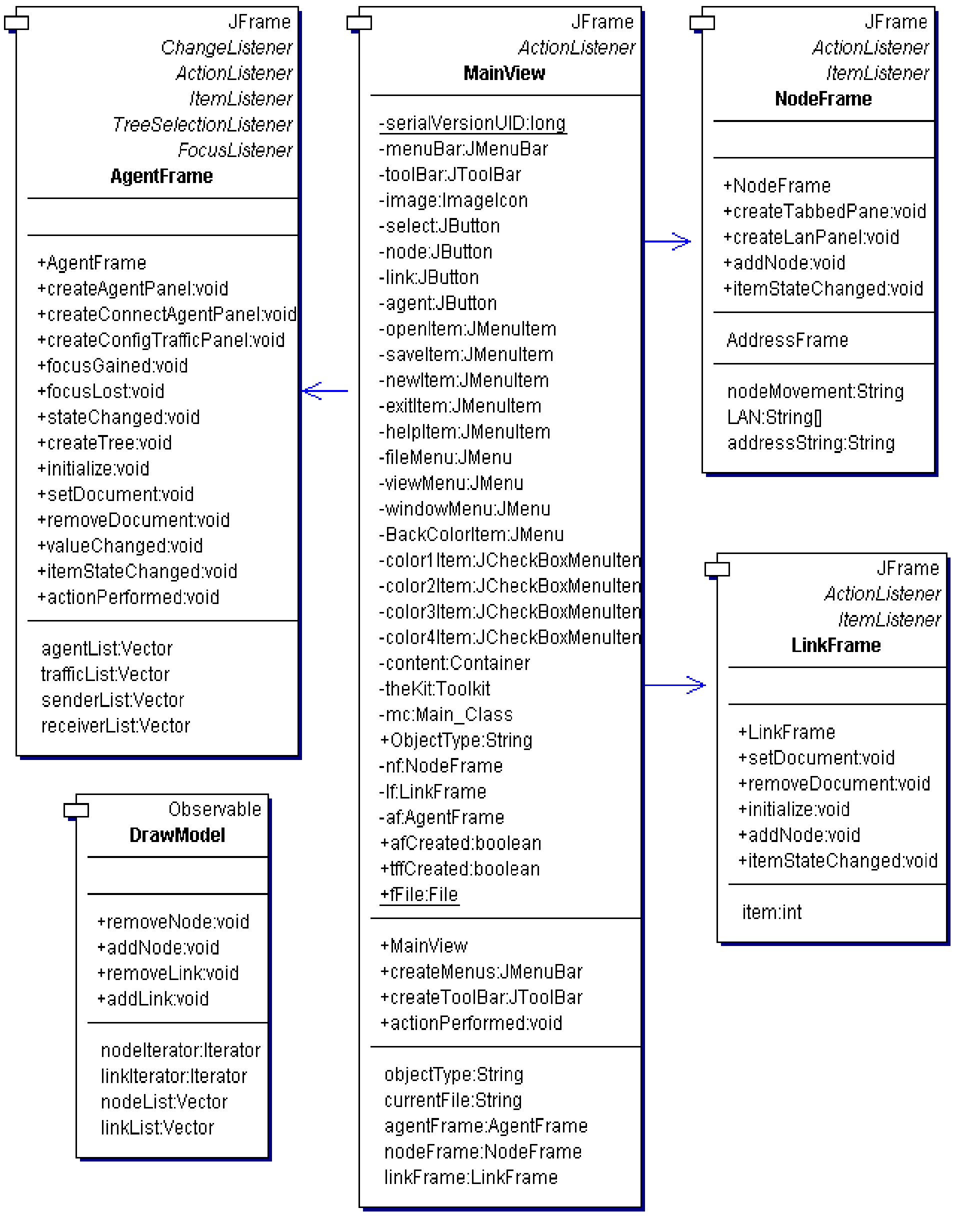


Figure7. Class diagram of GUI

Shown in Figure 8 is the class diagram in package MainPackage. There are five classes in this package. The class called Main_Class starts the program and it will launch the MainView window.

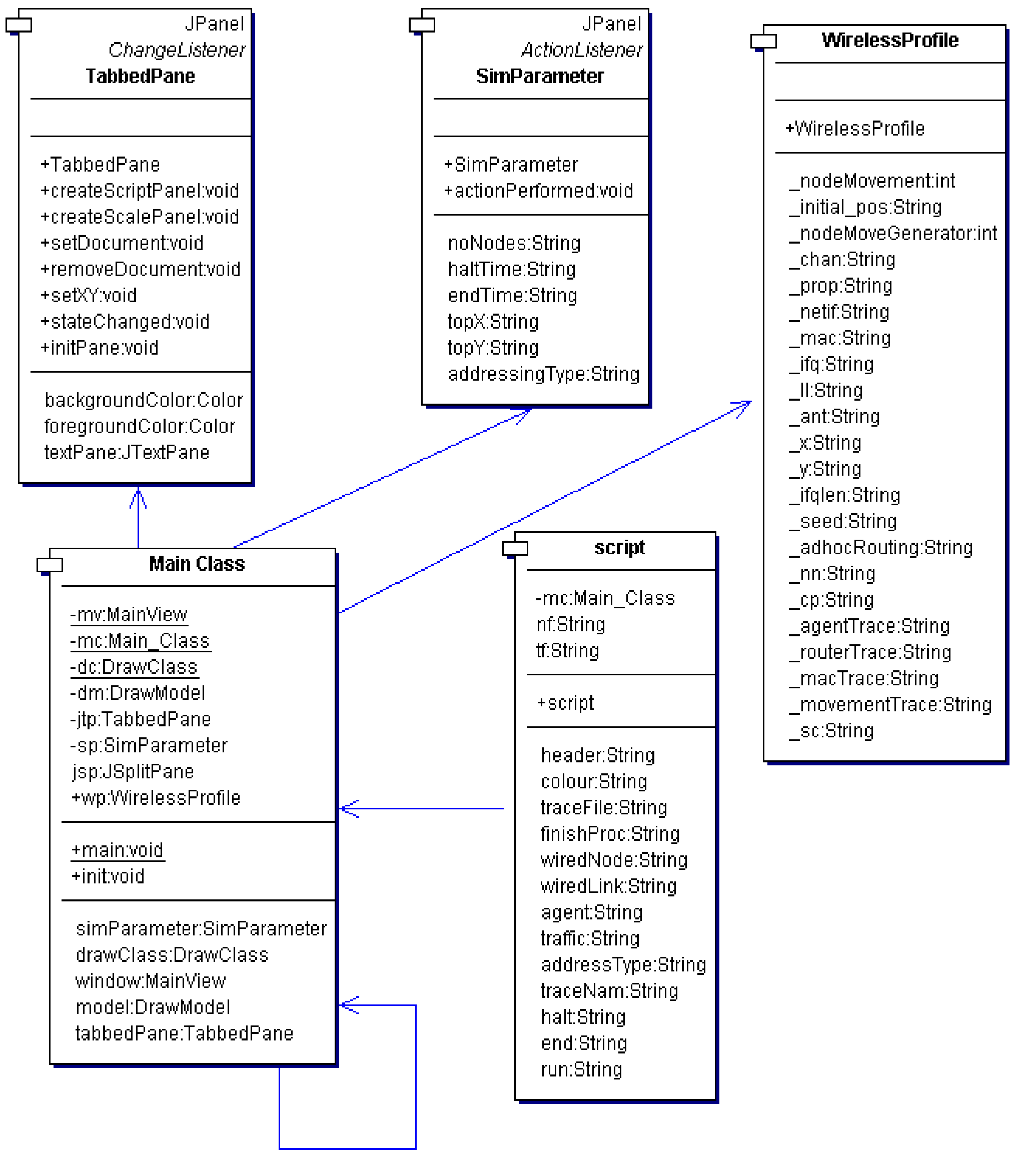


Figure8. Class diagram of MainPackage

NetworkComponent package has four classes. **Agent** class sets all the types of agent component parameters and lists all the variables. **Link** is a class which lists all the variables in link component. **Node** is also a class which lists all the variables in node component and sets the position of nodes. **Traffic** class initializes the traffic packages and set the parameters of traffic component.

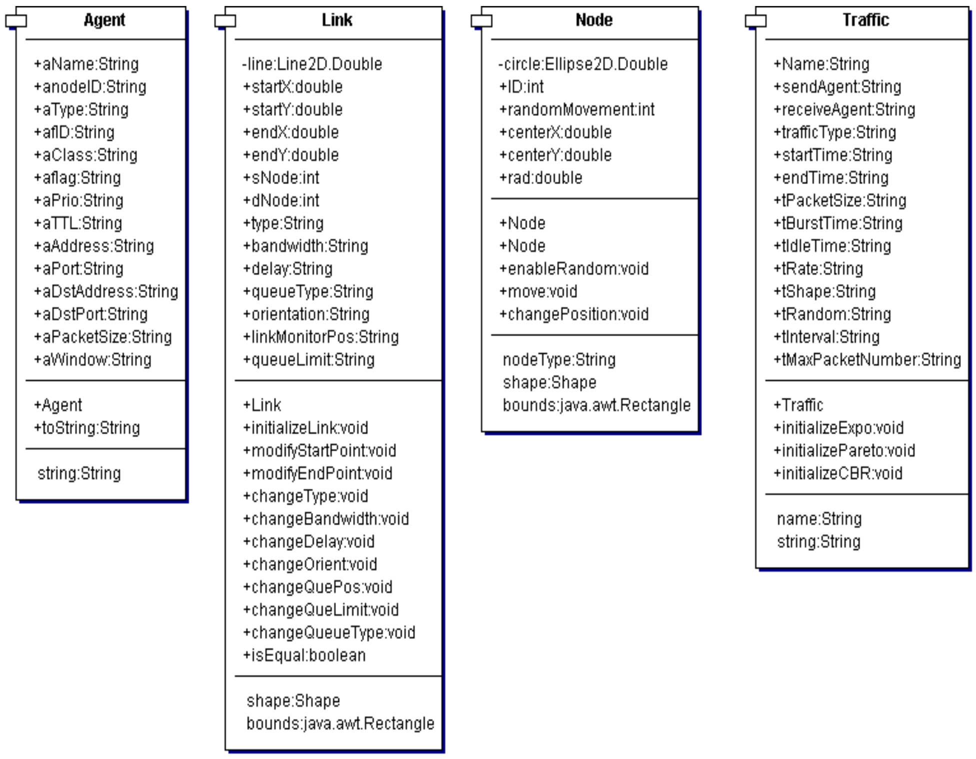


Figure9. Class diagram of NetworkComponent

3.5 A wireless example of Simulation TCL file

```
# Part 1: Node and protocol parameter setting
set val(chan)   Channel/WirelessChannel ;# channel type
set val(prop)   Propagation/TwoRayGround ;# radio-propagation model
set val(netif)  Phy/WirelessPhy ;# network interface type
set val(mac)     Mac/802_11 ;# MAC type
set val(ifq)     Queue/DropTail/PriQueue ;# interface queue type
set val(ll)       LL ;# link layer type
set val(ant)      Antenna/OmniAntenna ;# antenna type
set val(x)            600   ;# X dimension of the topography
set val(y)            600   ;# Y dimension of the topography
set val(ifqlen)        50          ;# max packet in ifq
set val(adhocRouting)   DSDV ;# Routing protocol
set val(nn)           3          ;# how many nodes are simulated
set val(cp)           "../cbr-3-test" ;# connection file
set val(sc)           "../scen-3-test" ;# scenario file
set val(stop)          400.0         ;# simulation time
=================================================================
# Main Program
=================================================================
# Initialize Global Variables
# create simulator instance
set ns_ [new Simulator]
# setup topography object
set topo    [new Topography]
# create trace object for ns and nam
set tracefd [open wireless1-out.tr w] # trace file
set namtrace   [open wireless1-out.nam w] # nam file

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# define topology
$topo load_flatgrid $val(x) $val(y)

# Create God
set god_ [create-god $val(nn)]

# define how node should be created

#global node setting
```

```
$ns_ node-config -adhocRouting $val(adhocRouting) \
              -llType $val(ll) \
              -macType $val(mac) \
              -ifqType $val(ifq) \
              -ifqLen $val(ifqlen) \
              -antType $val(ant) \
              -propType $val(prop) \
              -phyType $val(netif) \
              -channelType $val(chan) \
        -topoInstance $topo \
        -agentTrace ON \
      -routerTrace OFF \
      -macTrace OFF
#  Create the specified number of nodes [$val(nn)] and "attach" them
#  to the channel.
# Part 2
for {set i 0} {$i < $val(nn) } {incr i} {
   set node_($i) [$ns_ node]
   $node_($i) random-motion 0     ;# disable random motion
}
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} {incr i} {
# 20 defines the node size in nam, must adjust it according to your scenario
# The function must be called after mobility model is defined
   $ns_ initial_node_pos $node_($i) 20
}
# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at $val(stop).0 "$node_($i) reset";
}
# at $val(stop).0002 time ns will stop
$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
# Part 3
# add informative header for CUM trace file
puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"

puts "Starting Simulation..."
$ns_ run
# before run ns, Make sure the connection-pattern and node-movement files exist under
the directories as declared above.
```

List1. TCl file sample

Firstly, we need to setup the parameters and option from the mobile nodes like part 1 which is shown in TCL file sample. So that we can create network topology, nodes object, trace file and namtrace file. Secondly traffic source and movement are created in part 2. In this TCL file sample they are the random setting by “cbrgen.tcl” and “setdest” utilities for connection and movement. There is optional setting for the part. For connection pattern, an agent protocol entity is attached to a mobile node; traffic source packet will be set. For node movement, mobile nodes are initialized the position; and then specified with the node moving destination and speed. Finally, it is processing procedures in part 3, in order to close the output files.

4. IMPLEMENTATION

In this chapter I will show how the program gets the TCL script. After getting the tcl script I need to check it in the Cgywin environment. During this part, I found the variables mistake in the pervious code. Because the utility “cbrgen” and the utility “setdest” are used in the wireless simulation. These two utilities generate the node variable is “node(i)” [i means which number of node is used] and “$ns_”. “n(i)” and “$ns” are used in the pervious code.

4.1 Coding

On the basis of Huang Cheng’s project, I improve some functions for this program. From the beginning, for the design of my parts I need to study the NS2 arithmetic. Here is output of its main script for simulation a network by TCL in the following.

```
public String getFinishProc(){
     String finishProc="";
 if(mc.getSimParameter().TFselected==true&&mc.getSimParameter().TNselected==tru
e) //select function 1
          finishProc = "proc finish {} {\n" +"    "+" global ns_ tf nf\n"+
                    "    $ns_ flush-trace\n"+"    close $tf\n"+
                    "    close $nf\n"+"    exec nam out.nam &\n"+
                    "    exit 0\n"+"}\n";
     else
 if(mc.getSimParameter().TFselected==true&&mc.getSimParameter().TNselected==fal
se) //select function 2
          finishProc = "proc finish {} {\n" +"    global ns_ tf\n"+
                        "    $ns_ flush-trace\n"+"    close $tf\n"+"}\n";
          else
 if(mc.getSimParameter().TFselected==false&&mc.getSimParameter().TNselected==tr
ue) //select function 3

     finishProc = "proc finish {} {\n" +"    global ns_ nf\n"+
                            "    $ns_ flush-trace\n"+"    close $nf\n"+
                  "    exec nam out.nam &\n"+"    exit 0\n"+ "}\n";
```

```
      return finishProc;
  }
```

Listing2. Get a “finish” procedure

Listing 2 has the “finish” procedure dialog for TCL script. When the different parameters will be set, there are different dialogs. And it depends on trace file or trace nam. The procedure will be selected and trace file or trace nam is given here as parameters.

```
if(mc.getModel().getLinkList().size()==0){
scriptContent=scriptContent+"\n#Create nodes\n"+
              "for {set i 0} {$i <= $val(nn) } {incr i} {\n"+
              "     set node_($i) [$ns_ node]\n"+
              "     $node_($i) random-motion 0
     ;#disable random motion"+"\n"+
 "}\n";                                                          }
```

Listing3. Create wireless nodes

Listing 3 is the part scripts for creating the node in the action button function. If link number is “0”, it will print out this script. Otherwise the script will return the wired nodes which showed on Listing 3.

```
public String getWiredNode(){
     String createNodes="";
     createNodes = "for {set i 0} {$i < $opt(nn)} {incr i} {\n"+
                   "    set node_($i) [$ns node]\n"+
                   "}\n";
     return createNodes;                                                 }
```

Listing4. Create wired nodes

Listing 4 is the creating wired nodes source.

```
public String getWiredLink(){
       String createLinks="";
       Vector links=mc.getModel().getLinkList();
       Link linkTemp;
```

```
        for(int i=0;i<links.size();i++){
            linkTemp=(Link)links.get(i);
            createLinks = createLinks   +"\n$ns_ "+linkTemp.type+
          " $node_("+linkTemp.sNode+") $node_("+linkTemp.dNode+
          ") "+linkTemp.bandwidth+" "+linkTemp.delay+
          " "+linkTemp.queueType+"\n";

            if(!linkTemp.orientation.equals(""))
                createLinks=createLinks +"$ns_ "+linkTemp.type+
              "-op $node_("+linkTemp.sNode+") $node_("+linkTemp.dNode+
              ") orient "+linkTemp.orientation+"\n";

          if(!linkTemp.linkMonitorPos.equals(""))
                createLinks=createLinks   +"$ns_ "+linkTemp.type+
              "-op $node_("+linkTemp.sNode+") $node_("+linkTemp.dNode+
              ") queuePos "+linkTemp.linkMonitorPos+"\n";

          if(!linkTemp.queueLimit.equals(""))
                createLinks=createLinks +"$ns_ queue-limit $node_("
              +linkTemp.sNode+") $node_("+linkTemp.dNode+") "
                +linkTemp.queueLimit+"\n";
        }
        return createLinks;                                         }
```

Listing5. Create links between the nodes

Listing 5 is loop for creating links setup and links types. This is available when the wired simulation is used. Link amount, orientation, monitor position and queue limit are given as parameters. If each parameter is true, its command will be print out.

```
public String getString(){
        String s ="\nset "+aName+" [new Agent/"+aType.replace('_', '/')
               +"]\n"+"$ns_ attach-agent $node_("+anodeID+") $"+aName+"\n";

        if(aType.equals("TCP")){
            if(!aWindow.equals("20"))
                s=s+"$"+aName+" set window_ "+aWindow+"\n";
            if(!aPacketSize.equals("1000"))
                s=s+"$"+aName+" set packetSize_ "+aPacketSize+"\n";
        }else
```

```
    if(aType.equals("TCPSink")){
        if(!aWindow.equals(""))
            s=s+"$"+aName+" set window_ "+aWindow+"\n";
        if(!aPacketSize.equals("40"))
            s=s+"$"+aName+" set packetSize_ "+aPacketSize+"\n";
    }else
        if(aType.equals("RTP")){
            if(!aWindow.equals(""))
                s=s+"$"+aName+" set window_ "+aWindow+"\n";
            if(!aPacketSize.equals("210"))
            s=s+"$"+aName+" set packetSize_"+aPacketSize+"\n";
        }else
            if(aType.equals("Message")){
                if(!aWindow.equals(""))
                    s=s+"$"+aName+" set window_ "+aWindow+"\n";
                if(!aPacketSize.equals("180"))
            s=s+"$"+aName+" set packetSize_ "+aPacketSize+"\n";
            }else
                if(aType.equals("Ping")){
                    if(!aWindow.equals(""))
                    s=s+"$"+aName+" set window_ "+aWindow+"\n";
                    if(!aPacketSize.equals("64"))
            s=s+"$"+aName+" set packetSize_ "+aPacketSize+"\n";
                }else
                    if(aType.equals("UDP")){
                        if(!aWindow.equals(""))
                s=s+"$"+aName+" set window_ "+aWindow+"\n";
                        if(!aPacketSize.equals("1000"))
            s=s+"$"+aName+" set packetSize_ "+aPacketSize+"\n";
                    }else{
                        if(!aWindow.equals(""))
                s=s+"$"+aName+" set window_ "+aWindow+"\n";
                        if(!aPacketSize.equals(""))
            s=s+"$"+aName+" set packetSize_ "+aPacketSize+"\n";
                    }
if(!afID.equals("0"))
    s=s+"$"+aName+" set fid_ "+afID+"\n";
if(!aPrio.equals("0"))
    s=s+"$"+aName+" set prio_ "+aPrio+"\n";
if(!aflag.equals("0"))
    s=s+"$"+aName+" set flags_ "+aflag+"\n";
if(!aTTL.equals("32"))
    s=s+"$"+aName+" set ttl_ "+aTTL+"\n";
```

```
       if(!aClass.equals("0"))
            s=s+"$"+aName+" set class_ "+aClass+"\n";
       if(!aAddress.equals("-1"))
            s=s+"$"+aName+" set agent_addr_ "+aAddress+"\n";
       if(!aPort.equals("-1"))
            s=s+"$"+aName+" set agent_port_ "+aPort+"\n";
       if(!aDstAddress.equals("-1"))
            s=s+"$"+aName+" set dst_addr_ "+aDstAddress+"\n";
       if(!aDstPort.equals("-1"))
            s=s+"$"+aName+" set dst_port_ "+aDstPort+"\n";
       return s;
   }
```

Listing6. Creating agent

Listing6 is the creating agent function.

```
public String getString(){

       String s="\nset "+Name+" [new Application/" + trafficType +"]\n"+
               "$"+Name+" attach-agent $" + sendAgent +"\n"+
               "$ns_ connect $" + sendAgent + " $"+receiveAgent+"\n";
       if(trafficType.equals("Traffic/CBR")){
            if(!tPacketSize.equals("210"))
                 s=s+"$"+Name+" set packetSize_ "+tPacketSize+"\n";
            if(!tRandom.equals("0"))
                 s=s+"$"+Name+" set random_  "+tRandom+"\n";
            if(!tInterval.equals("1.0"))
                 s=s+"$"+Name+" set interval_ "+tInterval+"\n";
            if(!tRate.equals("448kb"))
                 s=s+"$"+Name+" set rate_    "+tRate+"\n";
            if(!tMaxPacketNumber.equals("268435456"))
                 s=s+"$"+Name+" set maxpkts_  "+tMaxPacketNumber+"\n";
       }else
            if(trafficType.equals("Traffic/Exponential")){
                 if(!tPacketSize.equals("210"))
                      s=s+"$"+Name+" set packetSize_ "+tPacketSize+"\n";
                 if(!tBurstTime.equals("500ms"))
                      s=s+"$"+Name+" set burst_time_ "+tBurstTime+"\n";
                 if(!tIdleTime.equals("500ms"))
                      s=s+"$"+Name+" set idle_time_  "+tIdleTime+"\n";
                 if(!tRate.equals("64kb"))
```

```
                    s=s+"$"+Name+" set rate_      "+tRate+"\n";
            }else
                if(trafficType.equals("Traffic/Pareto")){
                    if(!tPacketSize.equals("210"))
                    s=s+"$"+Name+" set packetSize_ "+tPacketSize+"\n";
                    if(!tBurstTime.equals("500ms"))
                        s=s+"$"+Name+" set burst_time_ "+tBurstTime+"\n";
                    if(!tIdleTime.equals("500ms"))
                        s=s+"$"+Name+" set idle_time_  "+tIdleTime+"\n";
                    if(!tRate.equals("64kb"))
                        s=s+"$"+Name+" set rate_      "+tRate+"\n";
                    if(!tShape.equals("1.5"))
                        s=s+"$"+Name+" set shape_     "+tShape+"\n";
                }else{
                }
        return s;
    }
```

Listing7. Create the traffic source

Listing 7 is the traffic source.

```
if(source==cbrRun){

cbrTrafficTemp="ns cbrgen.tcl -type "+jcbTrafficType.getSelectedItem()
              +" -nn " + Integer.parseInt(mc.getSimParameter().getNoNodes())
              +" -seed " + jtfSeed.getText()+" -mc " + jtfMc.getText()
              +" -rate " + jtfRate.getText();
StringBuffer sb = new StringBuffer();
try {
    Process p = Runtime.getRuntime().exec(cbrTrafficTemp);   //process the command
    InputStreamReader ir = new InputStreamReader(p.getInputStream());
    LineNumberReader input = new LineNumberReader(ir);
    String getOutputdata;
        while((getOutputdata = input.readLine()) != null){
               sb.append(getOutputdata+"\n"); //get output data
              }
    }catch (java.io.IOException e1) {
              System.err.println("IOException "+e1.getMessage());
        } try{
            FileWriter fstream = new FileWriter(jtfName.getText());
                //save the file name
            BufferedWriter out = new BufferedWriter(fstream);
                    out.write(sb.toString()); //write down all the output
                    out.close();
            }
            catch (Exception ae){
                System.err.println("Error: " + ae.getMessage());
                }
     }
```

Listing8. cbrgen command output

Listing 8 is an independent component in this project. This is “cbrgen” utility format outputting. If the RUN button pressed, this command will be executed.

```
if(source==setdestRun){
    if(jcbVersionType.getSelectedItem().toString()==("2")){
       SetdestTemp="./setdest "+" -v "+jcbVersionType.getSelectedItem().toString()
                 +" -n "+jtfNoNode.getText()+" -P " + jtfPType.getText() + " -p "
                 + jtfPTime.getText() + " -s " + jtfSType.getText()+ " -t "
                 + jtfSTime.getText()+ " -m " + jtfMinS.getText() + " -M "
                 + jtfMaxS.getText()+ " -x " + jtfWoSpace.getText() + " -y "
                 + jtfHoSpace.getText();
    }
    if(jcbVersionType.getSelectedItem().toString()==("1")){
        SetdestTemp="./setdest "+" -v"+jcbVersionType.getSelectedItem().toString()
                  +" -n " +jtfNoNode.getText()+ " -p " + jtfPTime.getText()
                  +" -t " +jtfSTime.getText()+ " -M "+ jtfMaxS.getText()+" -x "
                  +jtfWoSpace.getText() + " -y " + jtfHoSpace.getText();
    }
    StringBuffer sb = new StringBuffer();
    try {
        Process p = Runtime.getRuntime().exec(SetdestTemp);
        InputStreamReader ir =new InputStreamReader(p.getInputStream());
        LineNumberReader input = new LineNumberReader(ir);
        String getOutputdata;
        while((getOutputdata = input.readLine()) != null){
              sb.append(getOutputdata+"\n");
        }
        }catch (java.io.IOException e1) {
            System.err.println("IOException "+e1.getMessage());
        }
    try{
        FileWriter fstream =new FileWriter(jtfOutputName.getText());
            BufferedWriter out = new BufferedWriter(fstream);
            out.write(sb.toString()); //write down all the output
            out.close();
        }
        catch (Exception ae){
            System.err.println("Error: " + ae.getMessage());
        }
    }
```

Listing9. setdest command output

Because Listing 9 is an alone component in this program as well as. When button RUN pressed, the command will be printed out and executed. Version type is given as a parameter. There are two versions command of "setdest" utility.

4.2 Debugging

Debugging is an important part of implementation of any software. With debugging, all possible errors in logic and function can be found. Debugging of this program was done with two different ways. One is debugging the program window, are there any errors in parameter fields? Another is debugging the generation script in Cygwin environment, any there any errors in tcl script? In this case, I tested many examples of tcl file.

4.3 Results

In the results of this project, I show you how the program working and looking like, and I also give some personal opinions about it. In this chapter, I try to explain how each part of this program works.

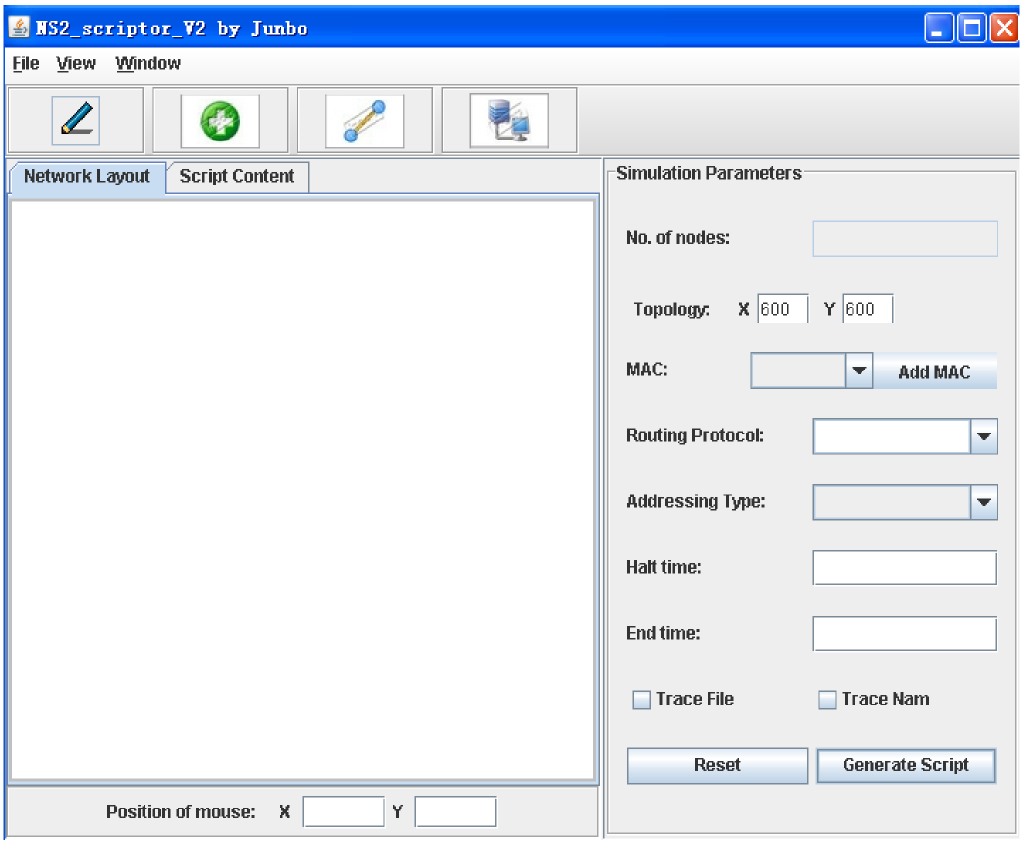


Figure10. Main Window

Figure10 shows the main window. In the title, it has the name of the program and version. Below there is menu, which has following items. File (Open, Save), View (Background color) and Window (New Window, Exit Window and ReadMe). Below the menu there are six component bars that are **Select**, **Node**, **Link** as well as **Agent**. Below these component bars it is divided from middle in two parts. Network Layout and Script content tabs are on the left side and Simulation Parameters is on the right side. Network diagram is displayed on the Network Layout, and the generated script is displayed on Script content. Final step is getting the script by Generate Script button.

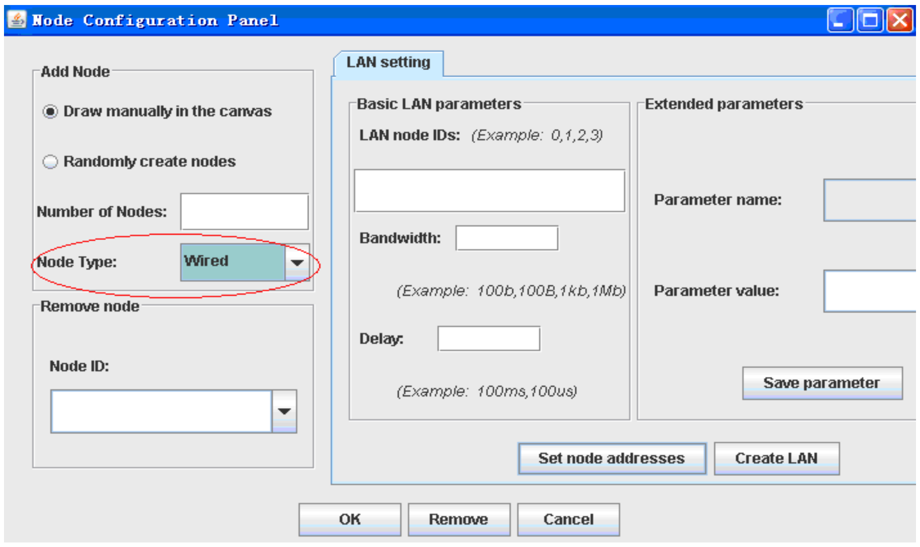


Figure11. Node (wired) Configuration Window

This project is started by setting node. When Node component bar selected, it show like Figure11. In this case, we need to choose which kind of network will be simulated. So node type (wired or wireless) is selected. And then we are going to set the parameters which they require. Finally when user presses OK button, the nodes will be displayed on Network Layout. If node type "wireless" is selected. It will be showed in Figure 12.

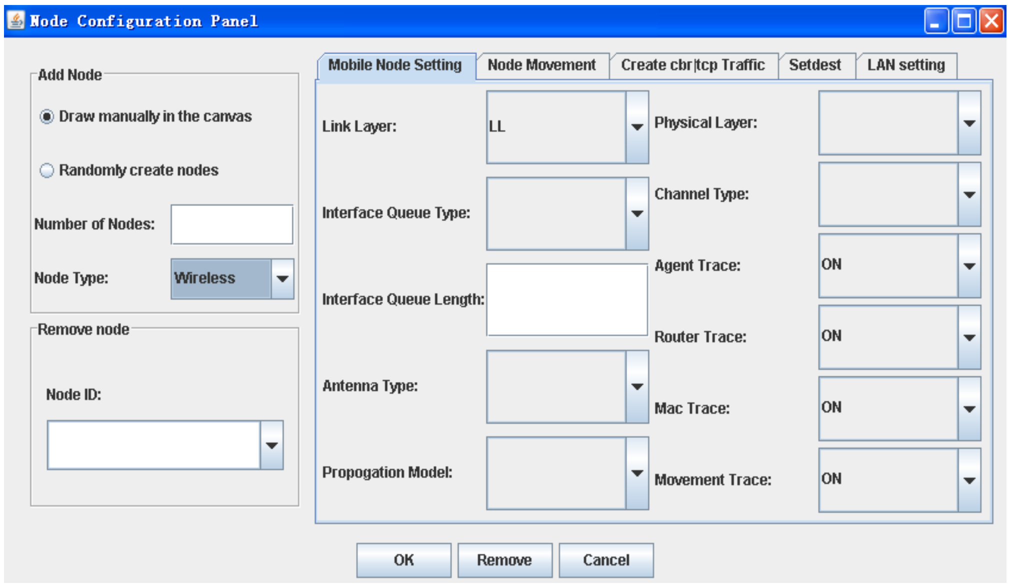
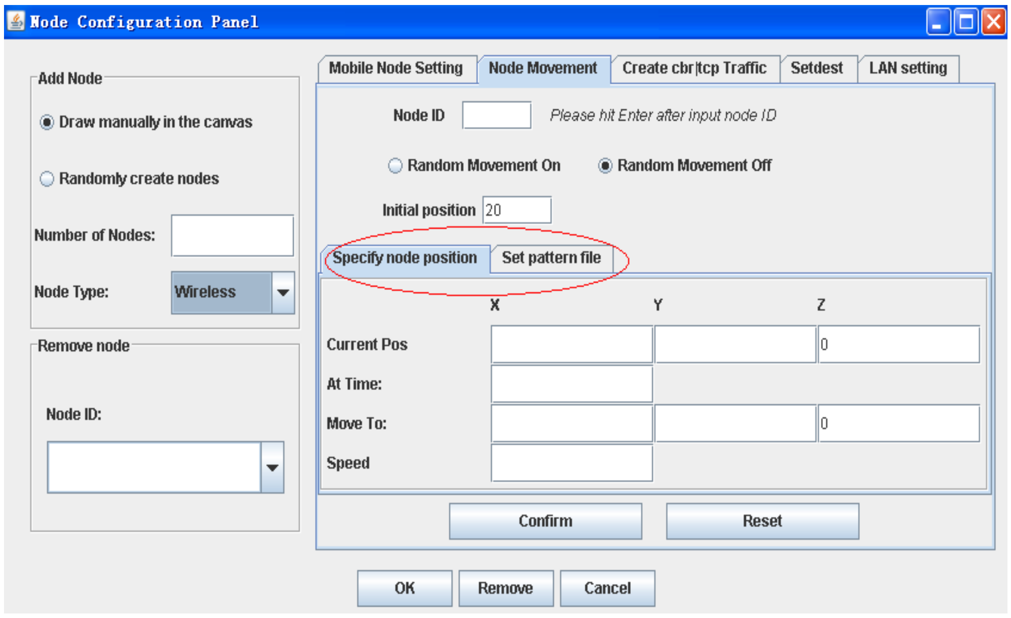


Figure12. Node (wireless) Configuration Window

Here shows all the parameters of wireless node configuration. There are mobile nodes setting, node movement, cbrgen command, setdest command as well as LAN setting. For the wireless nodes, we need to set the node movement (specify nodes position and set pattern file), which is showed in Figure13. In the nodes position configuration, there are random movements ON and OFF. When OFF is selected, we need to fill up all the parameters in specify nodes position field for each node. For the set pattern file tab, "create cbr|tcp traffic" tab (Figure14) and "setdest" tab (Figure15) are used firstly to generate a connection pattern and a scenario file. And user browses the connection file named as **Traffic Name** in "create cbr|tcp traffic" tab and scenario file named as **Output file name** in "setdest" tab. When all setting is done, remember to press the confirm button.


Node Configuration Panel

Add Node

[✓ Draw manually in the canvas] Randomly create nodes

Number of Nodes:

Node Type: Wireless

Remove node

Node ID:

Node ID — Please hit Enter after input node ID

Random Movement On [✓ Random Movement Off]

Initial position 20

	X	Y	Z
Current Pos			0
At Time:			
Move To:			0
Speed			

Figure13. Node movement of wireless nodes

Here is node movement configuration panel. The current position of node (X, Y) moves to another position (X, Y) at X time by X speed.

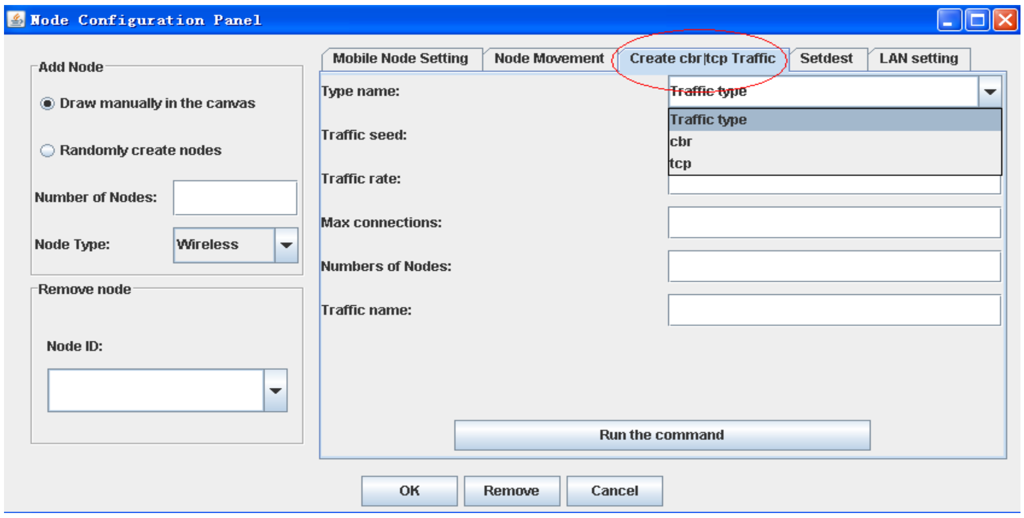
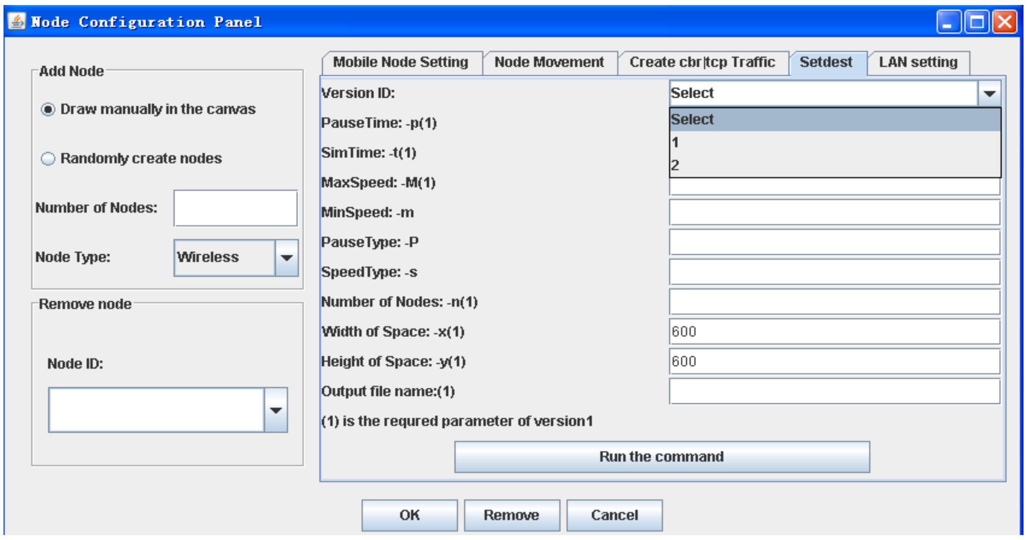


Figure14. Cbrgen command layout

When user selects cbrtraffic from the "create cbr|tcp traffic" tabs, random traffic setting window is shown. This window is used alone to run "cbrgen" command for generating the connection file available in wireless simulation. This command is for creating a traffic-connection file named as Traffic name. An automatic traffic generation utility "cbrgen" should be launch "`ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-rate rate] [-seed seed] [-mc connections] >file_name`". In this part, I can not execute the utility "cbrgen" command directly. Because Cgywin platform is the Linux environment for my computer. I need to input the command in Cgywin and generate the connection file (named as *file_name*).


Node Configuration Panel

Add Node

(● Draw manually in the canvas) ○ Randomly create nodes

Number of Nodes:

Node Type: Wireless

Remove node

Node ID:

Mobile Node Setting | Node Movement | Create cbr|tcp Traffic | Setdest | LAN setting

Version ID: Select (Select / 1 / 2)

PauseTime: -p(1)

SimTime: -t(1)

MaxSpeed: -M(1)

MinSpeed: -m

PauseType: -P

SpeedType: -s

Number of Nodes: -n(1)

Width of Space: -x(1) 600

Height of Space: -y(1) 600

Output file name:(1)

(1) is the requred parameter of version1

Figure15. Setdest command layout

When the user needs to create node-movements for wireless scenarios, “setdest” component tab is used. In NS2 there are two version “setdest” commands, he or she can select one of them. There are difference required parameters between version1 and version2. A random movement path utility “setdest” should be launch.

```
“./setdest [-v Version 1] [-p pause time] [-t sim time] [-M max speed] [-n nodes] [-x width space] [-y height space] > File Name” or
```

```
“./setdest [-v Version 2] [-p  pause time] [-t sim time] [-M max speed] [-m min speed] [-P pause type] [-s speed type] [-n nodes] [-x width space] [-y height space] > File Name ”
```

The same as utility “cbrgen”, I need to get the output of the command, and execute the command in Cgywin for generating the scenarios file (named as *File Name*).

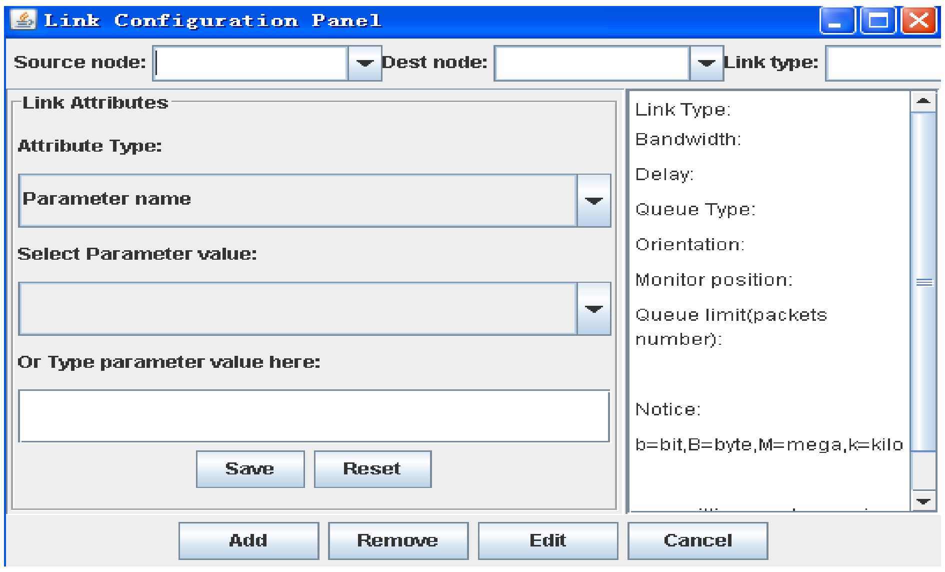


Figure16. Link Configuration Window

When node type is wired, we need to configure the Link parameters. In this component, it defines what kind of link we selected. When user is satisfied for each attribute, he or she can press SAVE button to save the data on right side. If he or she makes any mistake, it can be done easily with RESET button, and does it again. User can click ADD button to connect two nodes displaying on Network Layout. Removing the link can again be easily done with ROMOVE button. EDIT button can be used, when user wants to edit the data for the link.

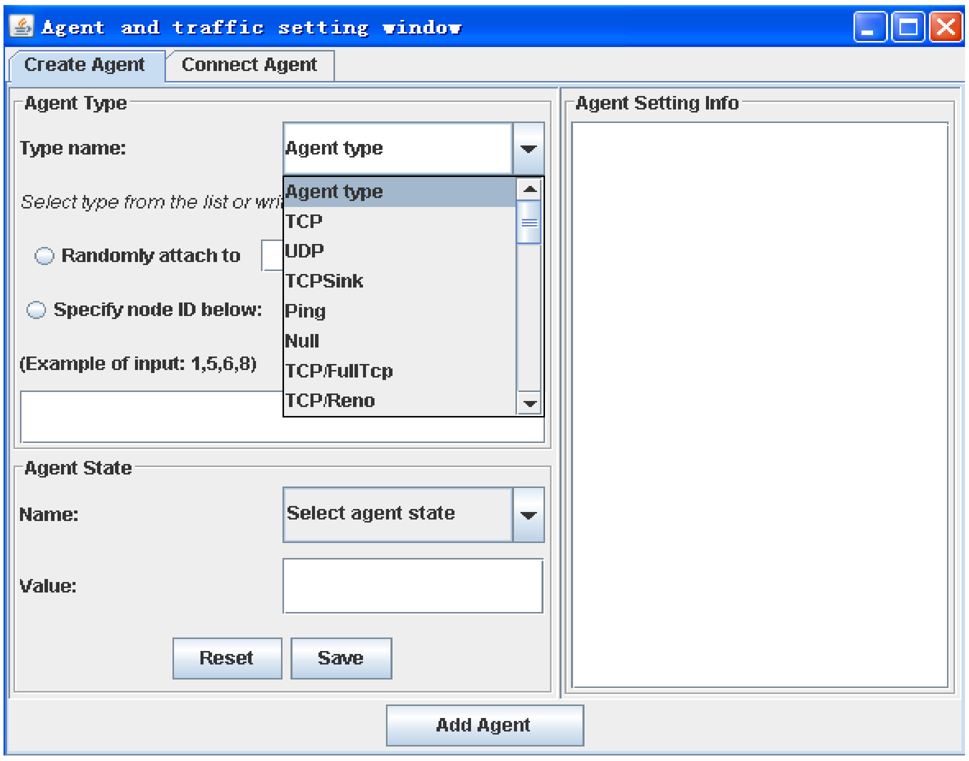


Figure17. Agent and Traffic setting Window

Creating agent, which can be done through Agent component tab, attaches to the nodes. When the Add Agent button is pressed, the agent will be attached to the nodes. For example, agent TCP and node ID 0 are set, when Add Agent button pressed, the message “TCP attach node 0” will show. In addition, we need to connect the agents and create the traffic by clicking Connect Agent tab, which shows in Figure18.

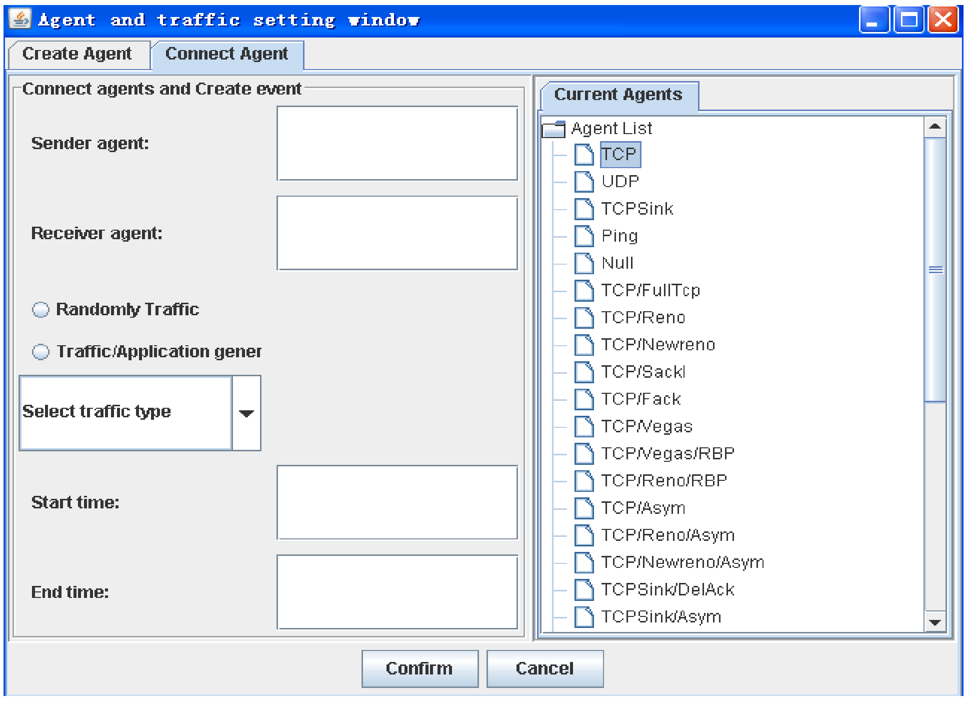


Figure18. Connect Agent Tab in Agent Window

.This window shows the connect agent message and traffic generation configuration. Here has the randomly traffic which is designed by java program.

The user can set the traffic parameters self by selecting traffic type as well as. And then he or she set the start time and end time. It means at S_time the traffic starts and at E_time the traffic will be end. When all configurations are done, remember to confirm.

When the nodes, link and agent frame configurations are done, go to the Main Window for generating the script, and save the content as a tcl file. Until here a tcl script is generated. Then we need to execute the tcl file under the ns environment. A tcl file should be launch. *"ns file_name.tcl"*

5. CONCLUSION

This project has carried out the purpose to generate NS scenario script for most networking researchers and engineers. The project work has almost fulfilled all of the requirements of wired and wireless network simulation. It works well so that all the functions are implemented and it offers user-friendliness which makes ns commands easier for user to use Network Simulation. But there are some attentions for wireless setup.

It will definitely help ns users to generate the scripts easily. For most of the ns command, users can get the scripts by giving the parameters value. Take creating nodes and links as example; in ns the command is like this below.

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

User can access the configuration from this project. Hence, users can get the commands directly without thinking about the key words in `ns`.

From my point of view, this project has been very beneficial for me. The project has been very challenging, but it has also helped me to learn a lot more about Java Swing and TCL language. One of the biggest challenges has been the fact that I did not know TCL language before. When the project generates the script and execute in ns, if there is any errors displayed in ns. I need to check it out what problem they have. This forced me to look and practice more deeply inside the source code of TCL, which gave me deeper knowledge of how tcl works.

Besides, I have also seen how powerful the internet search engines are. Whenever I meet problems or challenges, I search the relevant topics from the search engine and it always finds me pages from where I can find the way towards the solution. In fact, sometimes I can see others having similar questions about ns and good answers for them by experienced ns users, which is important and beneficial for me to find out the solution of the problems or challenges. Thus I can not only retrieve more information of NS-2, but also understand the contents of NS-2.

In my opinion, for the future there is a field that can be improved in this program. That is implementation directly, which user could be able to get the network simulation results with this project. Users do not need to go to NS environment for executing the tcl file. If we want to retrieve data from the trace file, we have to know how the trace files are constructed and what information we need. So we need to understand the trace format clearly.

In conclusion, the project has benefited me since it has developed my programming skill and understood what parameters are associated with a wireless network and how the communication in wireless networks differs from that of the wired ones.

6. SUMMARY

The aim of my final thesis project work was to develop a program with which ns user is able to generate the ns scenario script. Moreover, it offers simple and user friendly interface so that the user can use this program easily. This project was done in these phases: information gathering, analysis, design, implementation and debugging.

I have used Java Swing technology to code my project. Besides, an open source of ns tutorial was used for the reference of ns. I have chosen MyEclipse as the IDE (Integrated Development Environment) to create the application. I know the MyEclipse is a powerful tool to develop Java application.

To improve this project I need to study more about NS so that it can generate most of ns scripts.

LIST OF REFERENCES

[1]. Final Thesis: Visualizing NS-2 configuration with Java 2, Huang Cheng, VAMK, 2007

[2]. NS official manual: Kevin Fall & Kannan Varadhan, 2008

<URL: http://www.isi.edu/nsnam/ns/doc/>

[3]. NS tutorial: Marc Greis, 2008

<URL: http://www.isi.edu/nsnam/ns/tutorial/index.html>

[4]. NS by Example: Jae Chung & Mark Claypool, 2008

<URL: http://nile.wpi.edu/NS/>

[5]. NS-2 for wireless: Wu Zhibin, 2007

<URL: http://www.winlab.rutgers.edu/~zhibinwu/html/network_simulator_2.html>