Appendix A

NETWORK SIMULATOR (NS)

A.1 INTRODUCTION

Network Simulator 2 (NS2) [121, 123] was developed at Information Science Institute at the University of Southern California. NS2 works on all Unix and Windows platforms and supports networking research in the areas of protocol design, protocol comparison and traffic studies etc. The NS project is now a part of the VINT project that develops tools for simulation results’ display, analysis and converters that convert network topologies generated by well-known generators to NS formats.

A.2 FEATURES OF NS2

NS is an object oriented discrete event simulator because it models the world as events and executes one event after another from the list. Each event happens in an instant of virtual (simulated) time, but takes an arbitrary amount of real time. It implements network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR and real audio, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. With multiple levels of detail in one simulator, it can simulate varieties of networks such as wired, wireless, mobile and satellite etc. It can do packet-level simulation for link layer and upper layers i.e. network, transport, session, presentation and application. It is focused on modeling network protocols and supports the following:

A.3 COMPONENTS OF NS2

NS2 has a variety of components as mentioned below:

- NS, the simulator itself
- NAM, the network animator
  - Visualize ns (or other) output
  - NAM editor: GUI interface to generate ns scripts
- Pre-processing:
- Traffic generators
- Handwritten TCL or Topology generator

- Post-analysis:
  - Simple trace analysis using Perl/TCL/AWK/MATLAB

**A.4 NETWORK ANIMATOR (NAM)**

NS2 has a visualization tool named Network Animator (see Fig. A.1) that is Tcl based animation tool. NAM can view network simulation traces (see Fig. A.6) and real world packets trace data. It reads large animation data sets produced by NS and present it in a visualized format.

![Network Animator](image)

Figure A.1: Network Animator
A.5 LANGUAGES USED

Simulator had two distinct requirements:
- Detailed simulation of Protocol (Run-time speed)
- Varying parameters or configuration (Change model & rerun)

Due to the above reasons, NS2 uses two languages – OTcl and C++. C++ is fast to run but slower to change. On the contrary Otcl runs much slower but can be changed quickly.

At the front-end, OTcl is used for controlling the simulation in terms of simulation setup, configuration of objects and schedule of events. Simulation of slightly varying parameters or configurations can be done easily by quickly exploring a number of scenarios.

Detailed protocol simulations require systems programming language for byte manipulation, packet processing and algorithm implementation. C++ is employed for creation of objects because of its efficiency. It provides detailed and complete control over packet-processing. It is preferred because of the speed as turn around time (run simulation, find bug, fix bug, recompile, re-run) is slower.

From the user’s perspective, NS2 is an OTcl interpreter that takes an OTcl script as configuration interface and using C++ objects/functions to simulate network components/protocols and produces a trace file as output as shown in Fig. A.2.

Figure A.2: Simplified user's view of NS
A.6 DIRECTORY STRUCTURE OF NS2

Figure A.3 shows the directory structure of NS2 and Fig. A.4 shows the addition of code generated for the new protocol.

A.7 ADDING NEW CODE TO THE DIRECTORY STRUCTURE
A.8 SCHEDULING PROCEDURE

The event at the head of the event queue is delivered to its handler of some network component (object). Then, this network object may call other network objects to further handle this event, and may generate new events which are inserted into the event queue as

![Image of Discrete Event Scheduler](image)

Figure A.5: Discrete Event Scheduler

A.9 STEPS TO SIMULATE

Steps which are followed to simulate on NS2 involves:

- Initialize the simulator
- Define files for output (tracing)
- Set up the topology
- Set up the agents
- Set up the traffic between the nodes
- Start the simulation
- Analyze the trace files to compute the parameters of interest
- Turn on nam tracing in your Tcl script
- Specify link orientation (or node position for wireless)
- Execute nam
Figure A.6: Visualization on NAM

Wireless Example for ad hoc routing with following scenario:

- mobile nodes
- moving within 670mX670m flat topology
- using DSR ad hoc routing protocol
- Random Waypoint mobility model
- TCP and CBR traffic

A mobile node consists of network components like Link Layer (LL), Interface Queue (IFQ), MAC layer and the wireless channel. At the beginning of a wireless simulation, type for each of these network component is defined. Additionally, we need to define other parameters like the type of antenna, the radio-propagation model, the type of ad-hoc routing protocol used by mobile nodes etc.

```plaintext
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) 670 ;# X dimension of the topography
set val(y) 670 ;# Y dimension of the topography
set val(ifqlen) 50 ;# max packet in ifq
set val(seed) 0.0
set val(rp) DSR ;# ad-hoc routing protocol
set val(nn) 3 ;# number of mobile nodes
set val(cp) "/root/Desktop/ns-allinone-2.33/ns-2.33/tcl/mobility/scene/cbr-3-test"
set val(sc) "/root/Desktop/ns-allinone-2.33/ns-2.33/tcl/mobility/scene/scen-3-test"
set val(stop) 400.0 ;# simulation time

# Initialize Global Variables
# Open namtrace files
set nf [open out.nam w]
$ns_namtrace-all-wireless $nf $val(x) $val(y)

# setup trace support by opening file out.tr and call the procedure trace-all
set tracefd [open out.tr w]
$ns_trace-all $tracefd

# Create a simulator object
set ns [new Simulator]

# create a topology object that keeps track of movements of mobile nodes
set topo [new Topography]

# provide the topography object with x and y co-ordinates of the boundary
$topo load_flatgrid $val(x) $val(y)

# create the object God
set god_ [create-god $val(nn)]
# Create instance like this

set chan_1_ [new $val(chan)]

# 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) \ 
-topolnstance $topo \ 
-channelType $val(chan) \ 
-agentTrace ON \ 
-routerTrace OFF \ 
-macTrace OFF \ 
-channel $chan_1_ \ 

# Create the specified number of nodes [$val(nn)] and "attach" them to the channel.

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

176
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";
}

# Call the stop procedure after 400 seconds of simulation time
$ns_at 400_0 "stop"
$ns_at $val(stop)_0002 "puts \"NS EXITING...\" ; $ns_halt"

# add informative headers for the CMUTrace file, just before the line "ns_run"
puts $tracefd "M 0_0 no $val(no) x $val(x) y $val(y) rp $val(rp)"
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)"

# define stop procedure
proc stop {} {
    global ns_tracefd nf
    $ns_flush-trace
    close $tracefd
    close $nf ;#insert this line
    exec nam out.nam & ;#execute nam file
    exit 0 ;#insert this line
}

# run the simulation
puts "Starting Simulation..."
$ns_run
The script can be executed typing "ns ns-simple.tcl" at your shell prompt.

A.10 OUTPUT

Following output are seen as the result of the above mentioned script.

- Three mobile nodes moving in NAM window can be seen.

- However note that only node movements can currently be seen in NAM. Traffic flow and thus visualization of data packet movements in NAM for wireless scenarios is still not supported.

- From the CMU Trace output we find nodes 0 and 2 are out of range and so cannot hear one another. Node 1 is in range with nodes 0 and 2 and can communicate with both of them. Thus all packets destined for nodes 0 and 2 are routed through node 1.

A.11 MOTIVATION FOR SIMULATIONS

The simulator provides a collaborative environment because it is freely distributed as an open source and allows the sharing of code, protocols, models, etc. Complex scenarios can be easily tested and results can be quickly obtained, basically more ideas can be tested in a smaller timeframe.

A.12 LIMITATIONS

Real systems are too complex to model in the form of events and two languages i.e. OTcl and C++ are required to learn and debug in.