ns Tutorial

Table of Contents

Wireless Simulations

In this section, you are going to learn to use the mobile wireless simulation model available in ns. The section consists of two parts. In the first subsection, we discuss how to create and run a simple 2-node wireless network simulation. In second subsection, we will extend our example (in subsection 1) to create a relatively more complex wireless scenario.

IMPORTANT: This tutorial chapter uses new node APIs which are available as of ns-2.1b6, released January 18, 2000. If you have an earlier version of ns you must upgrade to use these features.

IX.1. Creating a simple wireless scenario

We are going to simulate a very simple 2-node wireless scenario. The topology consists of two mobilenodes, node_(0) and node_(1). The mobilenodes move about within an area whose boundary is defined in this example as 500mX500m. The nodes start out initially at two opposite ends of the boundary. Then they move towards each other in the first half of the simulation and again move away for the second half. A TCP connection is setup between the two mobilenodes. Packets are exchanged between the nodes as they come within hearing range of one another. As they move away, packets start getting dropped.

Just as with any other ns simulation, we begin by creating a tcl script for the wireless simulation. We will call this file simple-wireless.tcl. If you want to download a copy of simple-wireless.tcl click here.

A mobilenode consists of network components like Link Layer (LL), Interface Queue (IfQ), MAC layer, the wireless channel nodes transmit and receive signals from etc. For details about these network components see section 1 of chapter 15 of ns Notes & Documentation (now renamed as ns Manual). At the beginning of a wireless simulation, we need to define the type for each of these network components. 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 mobilenodes etc. See comments in the code below for a brief description of each variable defined. The array used to define these variables, val() is not global as it used to be in the earlier wireless scripts. For details and available optional values of these variables, see chapter 15 (mobile networking in ns) of ns documentation. We begin our script simple-wireless.tcl with a list of these different parameters described above, as follows:

```
# ======================================================================
# Define options
# ======================================================================
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-propagation model
set val(ant)          Antenna/OmniAntenna      ;# Antenna type
set val(ll)           LL                       ;# Link layer type
set val(ifq)          Queue/DropTail/PriQueue  ;# Interface queue type
set val(ifqlen)       50                       ;# max packet in ifq
set val(netif)        Phy/WirelessPhy          ;# network interface type
set val(mac)          Mac/802_11               ;# MAC type
set val(rp)           DSDV                     ;# ad-hoc routing protocol
set val(nn)           2                        ;# number of mobilenodes
```

Next we go to the main part of the program and start by creating an instance of the simulator,

```
set ns_    [new Simulator]
```

Then setup trace support by opening file simple.tr and call the procedure trace-all {} as follows:

```
set tracefd     [open simple.tr w]
$ns_ trace-all $tracefd
```

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

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

We had earlier mentioned that mobilenodes move within a topology of 500mX500m. We provide the topography object with x and y co-ordinates of the boundary, (x=500, y=500) :

```
$topo load_flatgrid 500 500
```

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

Next we create the object God, as follows:

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

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

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

Next, we create mobilenodes. The node creation APIs have been revised and here we shall be using the new APIs to create mobilenodes.
IMPORTANT NOTE: The new APIs are not available with ns2.1b5 release. Download the daily snapshot version if the next release (2.1b6 upwards) is not as yet available.

First, we need to configure nodes before we can create them. Node configuration API may consist of defining the type of addressing (flat/hierarchical etc), the type of adhoc routing protocol, Link Layer, MAC layer, IfQ etc. The configuration API can be defined as follows:

```
                                  (parameter examples)
# $ns_ node-config -addressingType flat or hierarchical or expanded
#                  -adhocRouting   DSDV or DSR or TORA
#                  -llType         LL
#                  -macType        Mac/802_11
#                  -propType       "Propagation/TwoRayGround"
#                  -ifqType        "Queue/DropTail/PriQueue"
#                  -ifqLen         50
#                  -phyType        "Phy/WirelessPhy"
#                  -antType        "Antenna/OmniAntenna"
#                  -channelType    "Channel/WirelessChannel"
#                  -topoInstance   $topo
#                  -energyModel    "EnergyModel"
#                  -initialEnergy  (in Joules)
#                  -rxPower        (in W)
#                  -txPower        (in W)
```

```
#                   -agentTrace     ON or OFF
#                   -routerTrace    ON or OFF
#                   -macTrace       ON or OFF
#                   -movementTrace  ON or OFF
```

All default values for these options are NULL except:
addressingType: flat

We are going to use the default value of flat addressing; Also lets turn on only AgentTrace and RouterTrace; You can experiment with the traces by turning all of them on. AgentTraces are marked with AGT, RouterTrace with RTR and MacTrace with MAC in their 5th fields. MovementTrace, when turned on, shows the movement of the mobilenodes and the trace is marked with M in their 2nd field.

The configuration API for creating mobilenodes looks as follows:

```
# 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
```

Next we create the 2 mobilenodes as follows:

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

The random-motion for nodes is disabled here, as we are going to provide node position and movement(speed & direction) directives next.

Now that we have created mobilenodes, we need to give them a position to start with,

```
#
# Provide initial (X,Y, for now Z=0) co-ordinates for node_(0) and node_(1)
#
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
$node_(1) set Z_ 0.0
```

Node0 has a starting position of (5,2) while Node1 starts off at location (390,385).

Next produce some node movements,

```
#
# 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 490.0 480.0 15.0"
```

$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0" means at time 50.0s, node1 starts to move towards the destination (x=25,y=20) at a speed of 15m/s. This API is used to change direction and speed of movement of the mobilenodes.

Next setup traffic flow between the two nodes as follows:

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

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"
```

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

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

```
#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 150.0 "$node_($i) reset";
}
$ns_ at 150.0001 "stop"
$ns_ at 150.0002 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
    global ns_ tracefd
    close $tracefd
}
```

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

And finally the command to start the simulation,

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

Save the file simple-wireless.tcl. In order to download a copy of the file click here. Next run the simulation in the usual way (type at prompt: "ns simple-wireless.tcl")

At the end of the simulation run, trace-output file simple.tr is created. As we have turned on the AgentTrace and RouterTrace we see DSDV routing messages and TCP pkts being received and sent by Router and Agent objects in node _0_ and _1_. Note that all wireless traces starts with WL in their first field. See Chapter 15 of ns documentation for details on wireless trace. We see TCP flow starting at 10.0s from node0. Initially both the nodes are far apart and thus TCP pkts are dropped by node0 as it cannot hear from node1. Around 81.0s the routing info begins to be exchanged between both the nodes and around 100.0s we see the first TCP pkt being received by the Agent at node1 which then sends an ACK back to node0 and the TCP connection is setup. However as node1 starts to move away from node0, the connection breaks down again around time 116.0s. Pkts start getting dropped as the nodes move away from one another.

IX.2. Using node-movement/traffic-pattern files and other features in wireless simulations

As an extension to the previous sub-section, we are going to simulate a simple multihop wireless scenario consisting of 3 mobilenodes here. As before, the mobilenodes move within the boundaries of a defined topology. However the node movements for this example shall be read from a node-movement file called scen-3-test. scen-3-test defines random node movements for the 3 mobilenodes within a topology of 670mX670m. This file is available as a part of the ns distribution and can be found, along with other node-movement files, under directory ~ns/tcl/mobility/scene. Random node movement files like scen-3-test can be generated using CMU's node-movement generator "setdest". Details on generation of node movement files are covered in section XI.2 of this tutorial.

In addition to node-movements, traffic flows that are setup between the mobilenodes, are also read from a traffic-pattern file called cbr-3-test. cbr-3-test is also available under ~ns/tcl/mobility/scene. Random CBR and TCP flows are setup between the 3 mobilenodes and data packets are sent, forwarded or received by nodes within hearing range of one another. See cbr-3-test to find out more about the traffic flows that are setup. These traffic-pattern files can also be generated using CMU's TCP/CBR traffic generator script. More about this is discussed in section XI.1 of this tutorial.

We shall make changes to the script, simple-wireless.tcl, we had created in section IX.1. and shall call the resulting file wireless1.tcl. For a copy of wireless1.tcl download from here. In addition to the variables (LL, MAC, antenna etc) that were declared at the beginning of the script, we now define some more parameters like the connection-pattern and node-movement file, x and y values for the topology boundary, a seed value for the random-number generator, time for the simulation to stop, for convinience. They are listed as follows:

```
set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/802_11
set val(ifq)        Queue/DropTail/PriQueue
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
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(adhocRouting)   DSR
set val(nn)             3             ;# how many nodes are simulated
set val(cp)             "../mobility/scene/cbr-3-test"
set val(sc)             "../mobility/scene/scen-3-test"
set val(stop)           2000.0          ;# simulation time
```

Number of mobilenodes is changed to 3; Also we use DSR (dynamic source routing) as the adhoc routing protocol inplace of DSDV (Destination sequence distance vector);

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

```
set tracefd  [open wireless1-out.tr w]      ;# for wireless traces
$ns_ trace-all $tracefd

set namtrace [open wireless1-out.nam w]           ;# for nam tracing
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
```

Next (after creation of mobilenodes) source node-movement and connection pattern files that were defined earlier as val(sc) and val(cp) respectively.

```
#
# Define node movement model
#
```

```
puts "Loading connection pattern..."
source $val(cp)

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

In node-movement file scen-3-test, we see node-movement commands like

```
$ns_ at 50.000000000000 "$node_(2) setdest 369.463244915743 \
170.519203111152 3.371785899154"
```

This, as described in earlier sub-section, means at time 50s, node2 starts to move towards destination (368.4,170.5) at a speed of 3.37m/s. We also see other lines like

```
$god_ set-dist 1 2 2
```

These are command lines used to load the god object with the shortest hop information. It means the shortest path between node 1 and 2 is 2 hops. By providing this information, the calculation of shortest distance between nodes by the god object during simulation runs, which can be quite time-consuming, is prevented.

The setdest program (see section XI.2) generates movement pattern files using the random waypoint algorithm. The node-movement files generated using setdest (like scen-3-test) already include lines like above to load the god object with the appropriate information at the appropriate time.

A program called calcdest (~ns/indep-utilities/cmu-scen-gen/setdest/calcdest) can be used to annotate movement pattern files generated by other means with the lines of god information. calcdest makes several assumptions about the format of the lines in the input movement pattern file which will cause it to fail if the file is not formatted properly. If calcdest rejects a movement pattern file you have created, the easiest way to format it properly is often to load it into ad-hockey and then save it out again. If ad-hockey can read your input correctly, its output will be properly formatted for calcdest.

Both setdest and calcdest calculate the shortest number of hops between nodes based on the nominal radio range, ignoring any effects that might be introduced by the propagation model in an actual simulation. The nominal range is either provided as an argument to the programs, or extracted from the header in node-movement pattern files.

The path length information provided to god was used by CMU's Monarch Project to analyze the path length optimality of ad hoc network routing protocols, and so was printed out as part of the CMUTrace output for each packet.

Other uses that CMU has found for the information are:

- Characterizing the rate of topology change in a movement pattern.
- Identifying the frequency and size of partitions.
- Experimenting with the behavior of the routing protocols if the god information is used to provide them with ``perfect'' neighbor information at zero cost.

Next add the following lines for providing initial position of nodes in nam. However note that only node movements can currently be seen in nam . Dumping of traffic data and thus visualization of data pkt movements in nam for wireless scenarios is still not supported (future work).

```
# Define node initial position in nam
for {set i 0} {$i < $val(nn)} {incr i} {

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

```
        # The function must be called after mobility model is defined
        $ns_ initial_node_pos $node_($i) 20
}
```

Next add informative headers for the CMUTrace file, just before the line "ns_ run" :

```
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)"
```

The rest of the script remains unchanged.

Save the file wireless1.tcl. Make sure the connection-pattern and node-movement files exist under the directories as declared above.
Run the script by typing at the prompt:

```
ns  wireless1.tcl
```

On completion of the run, CMUTrace output file "wireless1-out.tr" and nam output file "wireless1-out.nam" are created. Running wireless1-out.nam we see the three mobilenodes moving in nam window. However as mentioned earlier no traffic flow can be seen (not supported as yet). For a variety of coarse and fine grained trace outputs turn on/off AgentTrace, RouteTrace, MacTrace and movementTrace as shown earlier in the script. From the CMUTrace output we find nodes 0 and 2 are out of range and so cannot hear one another. Node1 is in range with nodes 0 and 2 and can communicate with both of them. Thus all pkts destined for nodes 0 and 2 are routed through node 1. For details on CMUTraces see chapter 15 of ns documentation.

[Previous section] [Next section] [Back to the index]

VINT

ns-users@isi.edu