ns-2 for the impatient

Imad Aad, Mohammad Hossein Manshaei, and Jean-Pierre Hubaux

EPFL
Lausanne, Switzerland
[hossein.manshaei, jean-pierre.hubaux]@epfl.ch
March, 2009

1. INTRODUCTION

This document is a quick tutorial on ns-2 [4]. It introduces the basic simulation aspects of wireless networking. Other detailed tutorials and documentation can be found in [4, 2, 3]. We assume the reader is using a Linux machine (on which *ns* is already installed) and is familiar with common Linux basics.

Here are some conventions used throughout this document:

- `~ns/` is the *ns* directory (`.../ns-allinone-2.33/`)
- Templates for the examples in this tutorial, can be found in `~ns/quick-tutorial/`.
- This document can be found in `~ns/quick-tutorial/quick-tutorial.pdf`
- If you need to go into more details, you can find the complete *ns Manual* in `~ns/quick-tutorial/ns-manual.pdf`

ns is written in *tcl* and in *C++*. We use *tcl* to configure the topology, the nodes, the channel, to schedule the events, etc. *C++* is used to implement the protocols (have a look at the 802.11 implementation in `~ns/ns-2.33/mac/mac-802_11.[cc,h]`). We will be using *tcl* scripts in this tutorial, without having to use the *C++* implementations (to modify the protocols for example). Other independent tools will be needed to filter, compute and display the results.

Before starting, you should reinstall the tutorial scripts: go to `~ns/` and run: `./tutorial-install.sh`.

2. A VERY SIMPLE SCENARIO: A "PURE" AD HOC NETWORK OF 6 NODES

Let us first start by showing the result of this example scenario, then we will explain the way to go there. Go to `~ns/quick-tutorial/` and run: `ns ex6sta.tcl`.

You will see a graph popping up, with 3 curves on it (Figure 1): the throughputs of 3 nodes when they contend to access the channel to send UDP packets to 3 others. They start at $t_1 = 20s$, $t_2 = 60s$ and $t_3 = 100s$ respectively. Simulation stops at $t_4 = 150$.

Let us have a close look at `ex6sta.tcl`. The nodes (`WT(1) ... WT(6)`) are created then positioned as follows (Figure 2):

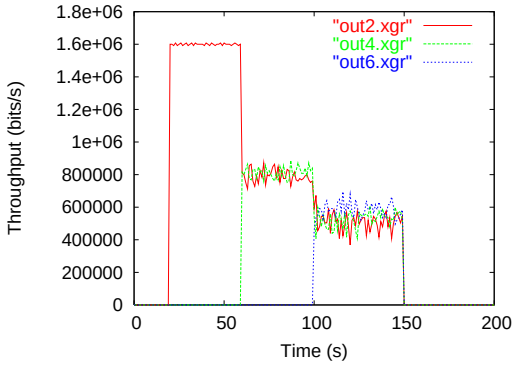

Figure 1: Throughputs of nodes WT(1), WT(3) and WT(5)

```
for {set i 1} {$i <= $opt(nn)} {incr i} {
    set WT($i) [ $ns_ node $i ]
}

proc coord_proc {a} {
    return [expr 10 * $a ]
}

for {set i 1} {$i <= $opt(nn)} {incr i} {
   $WT($i) set X_ [coord_proc $i]
   $WT($i) set Y_ [coord_proc $i]
   $WT($i) set Z_ 0.0
}
```

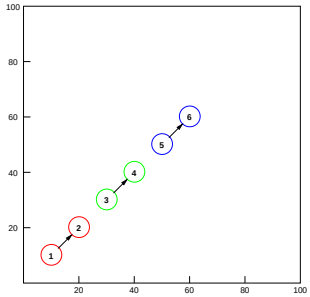

Figure 2: The ad hoc network topology

Like all the array variables `opt()` used throughout the script, `opt(nn)`

is defined (=6) at the beginning of `ex6sta.tcl`. `coord_proc` is a procedure we define to compute the values to be assigned to X_, Y_ (coordinates) of each node `WT(i)`. *ns-2* does not take the Z_ elevation into consideration (flat 2-D topologies only).

We use CBR (constant bit rate) sources, over UDP layers, that we attach to nodes WT(1),WT(3) and WT(5):

```
for {set i 1} {$i < $opt(nn)} {incr i 2} {
   set udp($i) [new Agent/UDP]
   $ns_ attach-agent $WT($i) $udp($i)

   set sink($i) [new Agent/Null]
   $ns_ attach-agent $WT([expr $i +1]) $sink($i)
   $ns_ connect $udp($i) $sink($i)

   set cbr($i) [new Application/Traffic/CBR]
   $cbr($i) set packetSize_ 1000
   $cbr($i) set interval_ 0.005
   $cbr($i) attach-agent $udp($i)

   $ns_ at [expr 20.0 * $i]  "$cbr($i) start"
   $ns_ at $opt(stop) "$cbr($i) stop"
}
```

For a UDP source in WT(i) we place a "sink" at WT(i+1) to receive the packets. Each CBR source sends 1000-Bytes packets each 5ms, therefore it can occupy alone the total the channel capacity (2 Mbits/s in this example). We schedule the sources to start at $t_1 = 20s$, $t_2 = 60s$ and $t_3 = 100s$ respectively, and to stop at $t_4 = 150s$.

Lower layers of Mobile nodes are configured as follows:

```
$ns_ node-config -adhocRouting  DumbAgent \
   -llType        LL \
   -macType       Mac/802_11 \
   -ifqType       Queue/DropTail/PriQueue \
   -ifqLen        50 \
   -antType       Antenna/OmniAntenna \
   -propType      Propagation/FreeSpace \
   -phyType       Phy/WirelessPhy \
   -channelType   Channel/WirelessChannel \
   -topoInstance  $topo \
```

`adhocRouting` specifies which ad hoc routing protocol to use (DSR, DSDV, AODV). Since all nodes are in the same collision domain, no ad hoc routing protocol is needed. `ifqLen` is the interface queue size. It has large influence on packet delays and drop rate. We use `FreeSpace` as a propagation model, which assumes there is a a single line of sight between senders and receivers (no radio wave reflections). More complex propagation models include `TwoRayGround` and `Shadowing`.

We consider a $100m \times 100m$ topology
(`$topo load_flatgrid $opt(x) $opt(y)`). The default communication range of 802.11 nodes in ns-2 is 240m. Therefore all 6 (=`opt(nn)`) nodes considered here are within receive range of each other.

We define what traces should be put in the trace file (`out.tr`):

```
   -agentTrace ON \
   -routerTrace OFF \
   -macTrace OFF \
   -movementTrace OFF
```

`agent` traces are the ones written by traffic sources/destinations. Router traces are written by packet forwarders. MAC traces are written by each node's MAC. Nodes' movements can be traced by switching `movementTrace ON` (we don't need it for the moment).

We use the basic mode (no RTS/CTS,
i.e. `Mac/802_11 set RTSThreshold_ 3000`), and the channel capacity is set to 2Mbps
(`Mac/802_11 set dataRate_ 2.0e6`)

A god (General Operations Director) needs to be created:

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

`god` is an object that is used to store global information about the state of the environment, network and nodes that an omniscient observer would have (without being available to any participant in the simulation). It accelerates the computation of the network routing/connectivity prior to the simulation run. We still need to create it even though we use no routing in our simulations.

Finally, at the end of the simulation (at t=`opt(stop)`), we reset all the wireless nodes and we call a procedure `finish`. In this procedure we call some external tools to filter, compute and display the throughput graphs. Everything is based on the trace file `out.tr`. Have it open and displayed while you follow the explanation. `out.tr` looks like this:

```
s 20.050000000 _0_ AGT  --- 10 cbr 1000 ...
r 20.051231990 _1_ AGT  --- 9 cbr 1000 ...
```

That is, at second `20.05` the agent in WT(1) (`_0_`)[1] sent (`s`) `cbr` packet number `10` of size `1000` (bytes). And at second `20.05123199`, agent in WT(2) received `cbr` packet number `9`, and so on.

Our goal is to compute the throughput of a given node `_i_` along the simulation time. That is, we need to sum up all packets sizes (column 8) of the rows starting with `r`, between second t and $t+1$. The "array of seconds" will contain the throughput curve of node `_i_`.

To do this computation, the *tcl* script (`ex6sta.tcl`) calls an external command, `awk` [1].

```
exec awk -f fil$i.awk out.tr > out$i.xgr
```

`awk` filters the trace file `out.tr` using filter `fil2.awk` and directs the output to `out2.xgr` (to be displayed with `xgraph` later on). Same with `fil4.awk` and `fil6.awk`.

Have a look at `fil2.awk`. `BEGIN{}` initializes an array `sec[i]`, which will contain the throughput at second i. This is done in the

[1]Note that node numbering starts from 0 in the trace file

following lines (between {}) which are applied to each line of the trace file.

```
if ($1=="r" && $7=="cbr" && $3=="_1_") {
    sec[int($2)]+=$8;
};
```

It indicates that if the value in column 1 ($1) is `r` (packet received) and the value in column 7 ($7) is `cbr` (packet type) and the value in column 3 ($3) is _1_ (WT(2)), then add the packet size ($8) to the throughput in `sec[int($2)]`.

at the end of the trace file, `awk` will display (output directed to `out1.xgr`) the array values as:

```
X        Y
sec[i]   throughput[i]
```

in bits/s.

Last, the output files (`out2.xgr, out4.xgr` and `out6.xgr`) are displayed calling:

```
exec xgraph out2.xgr out4.xgr out6.xgr &
```

from the `ex6sta.tcl` script.

Exercises:

With this basic information, you are (able and) asked to do the following:

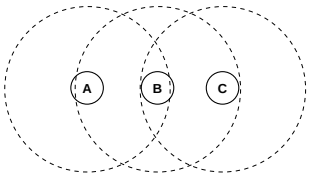

Figure 3: The hidden node scenario

1. Simulate a scenario with 3 nodes, A,B and C, where A is hidden to C[2], and vice versa, see Figure 3. A and C send full rate CBR/UDP packets to B (situated in their receive ranges). A and C use the basic mode (no RTS/CTS) and start transmissions at different times. Show the throughputs.
2. Same as 1, using RTS/CTS. Show both results on a single graph.
3. **Bianchi model verification:** In a topology with no hidden nodes, show the total system throughput when the number of flows increases from 1 to 50 (i.e. 100 nodes). Consider both basic and RTS/CTS modes and compare the results with the Bianchi model numerical simulations.
4. Simulate a multihop scenario (using AODV) and show the throughput vs. number of hops.

[2]You will have to reduce the carrier sense range (to 251m) using: `Phy/WirelessPhy set CSThresh_ 30.5e-10`

3. ADDING AN INFRASTRUCTURE

As in the previous section, we will first start by checking the result of the example. Go to `~ns/quick-tutorial/` and run `ns infra.tcl`. Simulations will start, then *xgraph* pops up showing a graph. *Nam* (Network AniMator) pops up two windows, one of which shows a network topology (you should zoom-in to see it properly). Figure 4 shows it more clearly.

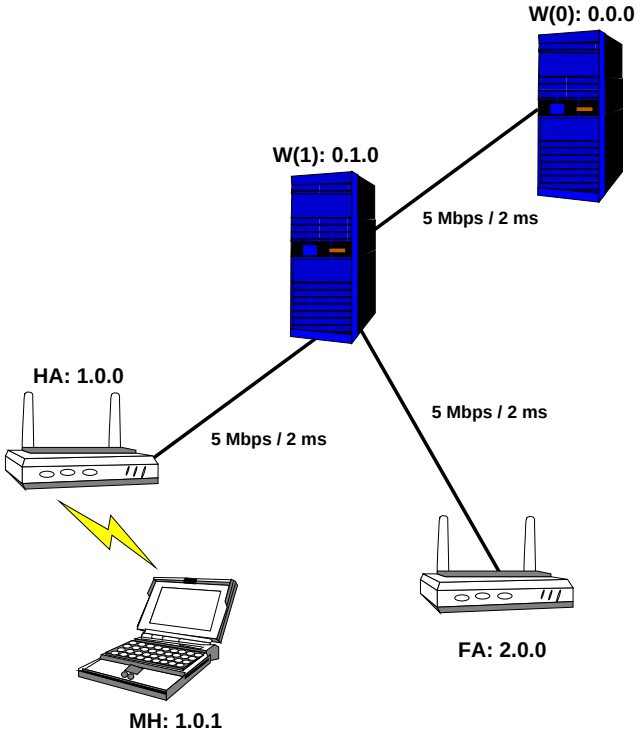

Figure 4: W(1) routes packets from W(0) to the Mobile Host (MH), associated with his Home Agent (HA) or with a Foreign Agent (FA)

A wired node W(0), with address `0.0.0` is sending TCP packets to a Mobile Host (MH: `1.0.0`) associated to its Home Agent (HA: `1.0.0`). MH starts moving towards the Foreign Agent (FA: `2.0.0`), gets disconnected from the HA (around $t = 113s$), then connects to the FA later on (at $t = 174s$). While moving back towards the HA, the MH gets disconnected again (at $t = 213s$) then reconnects to the HA's network (at $t = 239s$). This can be observed in the `xgraph` output curve.

Wired node W(1) (`0.1.0`) routes the packets from W(0) to the HA and FA. All wired links are 5Mbps and have 2ms delays (cf. `infra.tcl`):

```
$ns_ duplex-link $W(0) $W(1) 5Mb 2ms DropTail
$ns_ duplex-link $W(1) $HA 5Mb 2ms DropTail
$ns_ duplex-link $W(1) $FA 5Mb 2ms DropTail
```

We can control how these wired links look like in *NAM* by setting:

```
$ns_ duplex-link-op $W(0) $W(1) orient down
...
```

How do we tell the HA to forward the MH's packets to its new location (FA)? *MobileIP* takes care of this:

```
$ns_ node-config -mobileIP ON
```

And we have to register the MH at the corresponding Home Agent (HA):

```
set MH [$ns_ node 1.0.1]
set HAaddress [AddrParams addr2id [$HA node-addr]]
[$MH set regagent_] set home_agent_ $HAaddress
```

Therefore, when the MH moves towards the FA (starting at second 100)

```
$ns_ at 100.00 "$MH setdest 640.00 610.00 20.00"
```

at a speed of 20m/s (640 and 610 are the coordinates of the FA), the home agent (HA) will take care of tunneling MH's packets to the FA.

In this topology we are using hierarchical addresses:

```
$ns_ node-config -addressType hierarchical
```

composed of 3 domains (`0.x.x` to which the wired nodes belong, `1.x.x` where HA and MH belong, and `2.x.x` where the FA belongs), therefore:

```
AddrParams set domain_num_ 3
```

The first of the 3 domains has 2 clusters: `0.0.x` and `0.1.x`. The second domain (`1.x.x`) has only 1 cluster (i.e. `1.0.x`), and the last domain also has 1 cluster (i.e. `2.0.x`), therefore:

```
lappend cluster_num 2 1 1
```

Finally, cluster `0.0.x` has 1 node; cluster `0.1.x` has 1 node; cluster `1.0.x` has 2 nodes; and cluster `2.0.x` has 1 node. Therefore:

```
lappend eilastlevel 1 1 2 1
```

The rest of the *tcl* script is similar to the previous example (in Section 2).

Note that in this example we have 2 trace files:

```
set tracefd  [open $opt(tr-ns) w]
set namtrace [open $opt(tr-nam) w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace ...
```

One of these files is used by *NAM*. We will let you discover *NAM*'s GUI intuitively, and watch the packet runs on the wired links. Mobile nodes can be seen moving in *NAM*. However, packets are not shown on the wireless channel.

As for the *ns* trace file, the structure differs slightly from the one seen before since it combines wireless (similar to the previous one) and wired traces. Since we are only interested in the throughput of the MH (wireless), we only had to adapt the *awk* filter `fil-tcp.awk` to count tcp packets instead of cbr ones:

```
if ($1=="r" && $7=="tcp" && $3=="_4_") {...
```

Exercises:

Now that you are familiar with wireless and wired networking simulations, you are (able and) asked to simulate all scenarios presented in Figure 5 and Table 1.

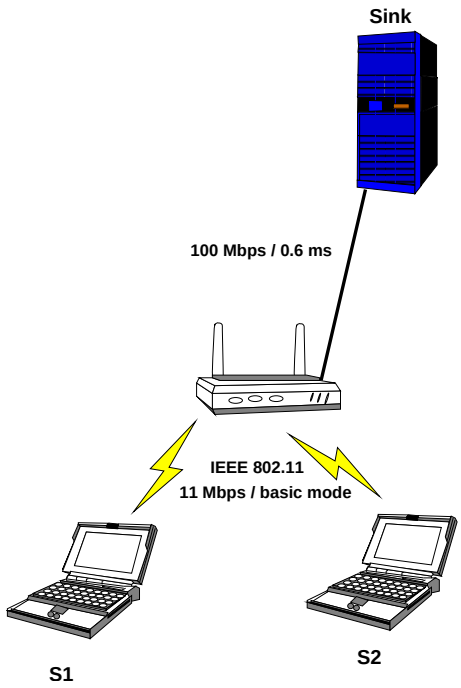

Figure 5: S1 and S2 contend to access the channel to send their traffic to the Sink

4. FAQ

Q: Compilation problem... what's wrong?
A: Make sure there are no spaces after
`$ns_ node-config -adhocRouting  DumbAgent \`
and all `\`s of the following lines.

Q: Even though the hidden nodes are far enough from each other, it seems they still can sense each other (no decrease in throughput). Why?
A: A common problem is to miss one of the "h"s in:
`Phy/WirelessPhy set CSThresh\_`

Scenario #	S1
1	7 Mbps UDP, pkt_size= 1448 B
2	7 Mbps UDP, pkt_size= 1448 B
3	7 Mbps UDP, pkt_size= 1448 B
4	7 Mbps UDP, pkt_size= 724 B
5	7 Mbps UDP, pkt_size= 2000 B
Scenario #	S2
1	7 Mbps UDP, pkt_size= 1448 B
2	7 Mbps UDP, pkt_size= 724 B
3	TCP
4	TCP
5	TCP

Table 1: Scenarios for topology in Figure 5

5. REFERENCES

[1] http://www.cs.hmc.edu/qref/awk.html.

[2] Kevin Fall and Kannan Varadhan. The *ns* manual. http://www.isi.edu/nsnam/ns/ns-documentation.html.

[3] M. Greis. Marc Greis' Tutorial for the UCB/LBNL/VINT Network Simulator "ns". http://www.isi.edu/nsnam/ns/tutorial/index.html.

[4] The VINT Project. Network Simulator. http://www.isi.edu/nsnam/ns/.