Network Simulator

Network Simulator (ns-2)
6.829 tutorial

Why Network simulation

- protocol validation
- controlled experimental conditions
- low cost in $, time, collaboration, complexity

Why NS?

Provides:

- protocols: TCP, UDP, HTTP, etc.
- Traffic Models: Web Traffic, CBR,
- Topology Generation tools
- Visualization tools
- large validation package (people believe it works)

NS Structure

- C++ event scheduler
  protocols (most)
- TCL scripts
  protocols (mostly extensions to C++ core)
- TCL objects expose an interface to C++ objects (shadow objects)
  system configuration (defaults, etc.)

1 minute TCL tutorial

```
set a 1
set b [expr $a + 2]

proc list_total {list} {
    set tmp 0
    foreach l $list {
        incr tmp $l
    }
    return $tmp
}

set c [list_total {1 2 3 4 5}]
puts "a: $a b: $b c: $c"
```

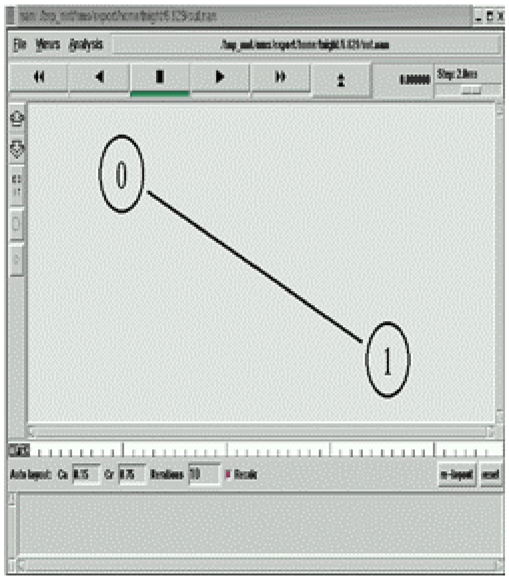
1 minute oTCL tutorial

```
Class car
car instproc init {args} {
    $self instvar wheels_
    set wheels_ 4
}

Class vw -superclass car
vw instproc init {args} {
    $self instvar wheels_ vendor_
    set vendor_ "vw"
    eval $self next $args
}

set a [new car]
puts "A: a car with [$a set wheels_] wheels"
set b [new vw]
puts "B: a [$b set vendor_] with [$b set wheels_] wheels"
```

ns 1: Creating Topology

We first create a Simulator object which we use to generate nodes and links. This code will generate a topology (2 nodes, 1 link), but won't generate any traffic. Note: we can schedule any arbitrary tcl code to be run with a `$ns at ...` command. Here we schedule the end of the simulation at 5 seconds.

```
# Create simulation object
set ns [new Simulator]

# Ask ns for nodes
set n0 [$ns node]
set n1 [$ns node]

# Create a duplex link b/w n0 &
n1
$ns duplex-link $n0 $n1 1Mb 10ms
DropTail

# Schedule End
$ns at 5.0 "exit 0"
# Run Simulation
$ns run
```



ns 2: Attaching Agents

Agents implement differenct connections between nodes. These connections are of some protocol type: TCP, UDP, etc. Here we show a simple UDP example, since a UDP sink would do nothing we use the null sink.

```
# Create a UDP agent
set udp1 [new Agent/UDP]
# Create a Null agent
set sink1 [new Agent/Null]
# Attach agent udp1 to node n0
$ns attach-agent $n0 $udp1
# Attach agent sink1 to node n1
$ns attach-agent $n1 $sink1
# Connect the agents
$ns connect $udp1 $sink1
```

ns 2: TCP Agents

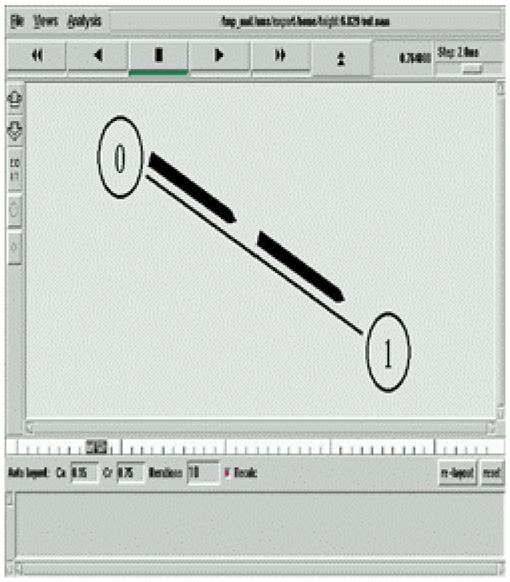
Of course instead of using UDP we could use TCP, but then we'll have to use a specialized sink to generate acknowledgements.

```
# Create a TCP agent
set tcp1 [new Agent/TCP]
# Create a Null agent
set sink1 [new Agent/TCPSink]
# Attach agent tcp1 to node n0
$ns attach-agent $n0 $tcp1
# Attach agent sink1 to node n1
$ns attach-agent $n1 $sink1
# Connect the agents
$ns connect $tcp1 $sink1
```

ns 3: Attaching Sources

Still, our network will not carry any data. We need a data source which will feed bytes to our agent. Here we generate a constant bit rate source (CBR).

```
# Create Source
set cbr1 [new
Application/Traffic/CBR]

# Configure Source
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005

# Attach source to agent
$cbr1 attach-agent $udp1

# Schedule cbr on
$ns at 0.5 "$cbr1 start"
# Schedule cbr off
$ns at 4.5 "$cbr1 stop"
```



ns 4: Tracing

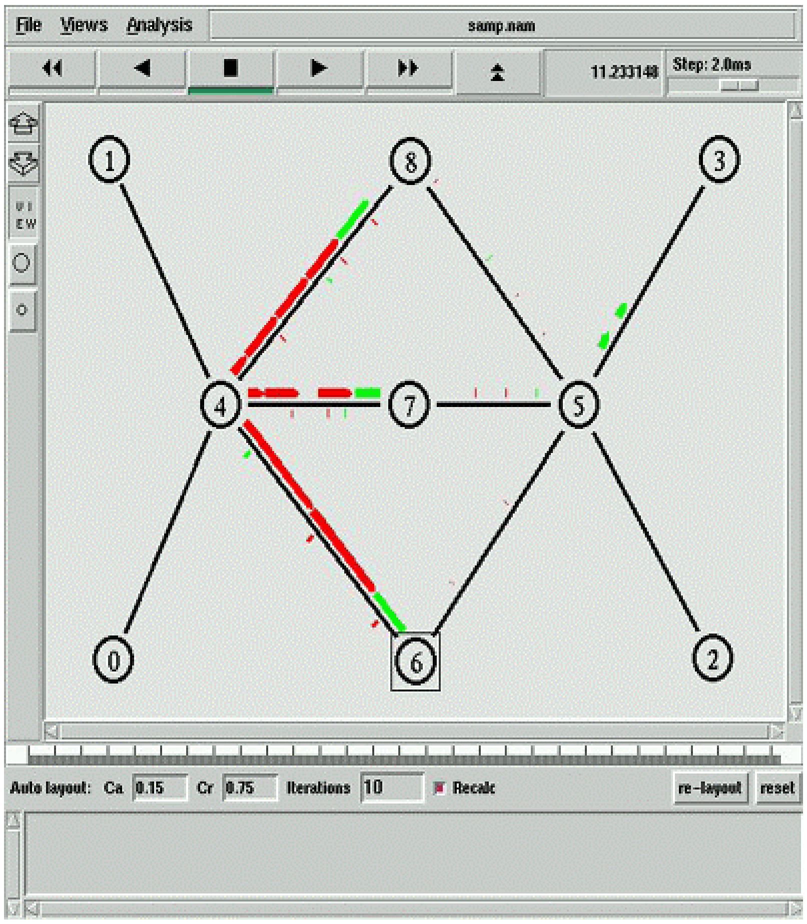
While our current simulation will create sources and generate traffic which will travel over our link, it will not create any useful output for us. Here we tell our simulation to trace events for us. Note in the full code file that the trace-all and nam-trace-all commands must be run before nodes and agents are created, but the variable trace commands must be run afterwards:

```
# all packet trace:
$ns trace-all [open out.tr w]

# animator trace:
$ns namtrace-all [open out.nam w]

# variable trace:
set tf [open "cwnd.tr" w]
set tracer [new Trace/Var]
$tracer attach $tf
$tcp trace cwnd_ $tracer
```

nam Network Animator

You can run the network animator using the nam command.

```
%nam out.nam
```



File
Views
Analysis
samp.nam
11.233148
Step: 2.0ms
1
8
3
4
7
5
0
6
2
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset