
An Introduction to NS-2*

Kiran Kotla
Teaching Assistant
ECEN 602

**Adapted from a presentation by Gayatri, UCSB*

NS-2 Learning Resources

- Course webpage for ECEN 602:

http://www.ece.tamu.edu/~reddy/ee602_07.html

NS-2 learning resources are available on this page

Why Simulation?

It is difficult to deploy and test customized protocols in different machines across the world

To analyze how different protocols work in complex real world scenarios.

Why NS-2?

NS-2 stands for Network Simulator
version-2

The most widely used open source
network simulator available for free with
Source code

This is developed by ISI at USC

www.isi.edu

What is ns?

- Object-oriented, discrete event-driven network simulator
- Written in C++ and OTcl

Hello World – Interactive mode

```
bash-shell$ ns
% set ns [new Simulator]
% $ns at 1 "puts \"Hello World!\""
% $ns at 1.5 "exit"
% $ns run
Hello World!
bash-shell$
```

Basic Tcl: ex-tcl.tcl

```
# Writing a procedure called "test"  
proc test {} {  
    set a 43  
    set b 27  
    set c [expr $a + $b]  
    set d [expr [expr $a - $b] * $c]  
    for {set k 0} {$k < 10} {incr k} {  
        if {$k < 5} {  
            puts "k < 5, pow = [expr pow($d, $k)]"  
        } else {  
            puts "k >= 5, mod = [expr $d % $k]"  
        }  
    }  
}  
  
# Calling the "test" procedure created above  
test
```

NS-2 Generic Script Structure

1. Create Simulator object
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Step1: Create Simulator Object

- Create event scheduler
 - `set ns [new Simulator]`

Step2: Tracing

- Insert immediately after scheduler!

- Trace packets on all links

```
set nf [open out.nam w]
```

```
$ns trace-all $nf
```

```
$ns namtrace-all $nf
```

Step 3: Create network

- Two nodes, One link



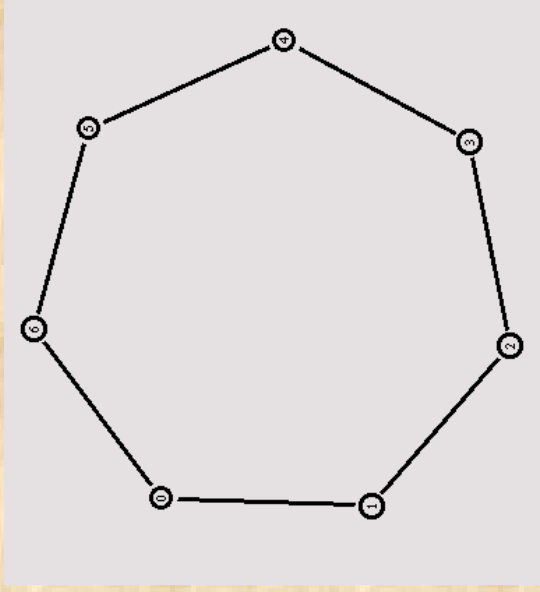
Step 3: Create Network

- Nodes
 - `set n0 [$ns node]`
 - `set n1 [$ns node]`
- Links and queuing
 - `$ns duplex-link $n0 $n1 1Mb 10ms RED`
 - `$ns duplex-link $n0 $n1 <bandwidth> <delay>`
 - <queue_type>
 - <queue_type>: DropTail, RED, etc.



Creating a larger topology

```
for {set i 0} {$i < 7} {incr i} {  
  set n($i) [$ns node]  
}  
for {set i 0} {$i < 7} {incr i} {  
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms RED  
}
```



NS-2 Generic Script Structure

1. Create Simulator object
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Step 4: Network Dynamics

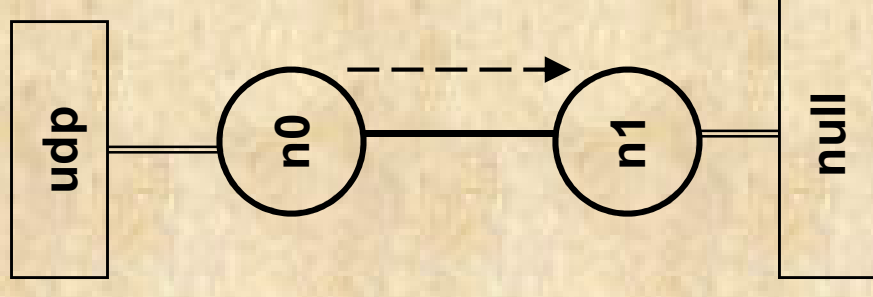
- Link failures
 - Hooks in routing module to reflect routing changes
- `$ns rtmodel-at <time> up|down $n0 $n1`
- For example:

```
$ns rtmodel-at 1.0 down $n0 $n1
```

```
$ns rtmodel-at 2.0 up $n0 $n1
```

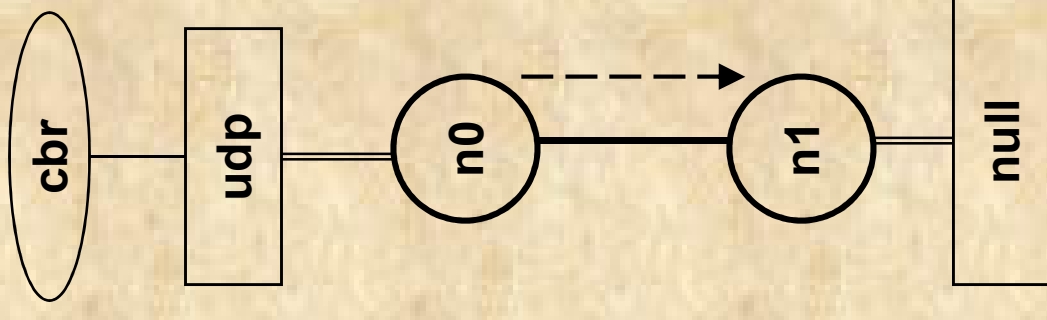
Step 5: Creating UDP connection

```
set udp [new Agent/UDP]  
set null [new Agent/Null]  
  
$ns attach-agent $n0 $udp  
$ns attach-agent $n1 $null  
  
$ns connect $udp $null
```



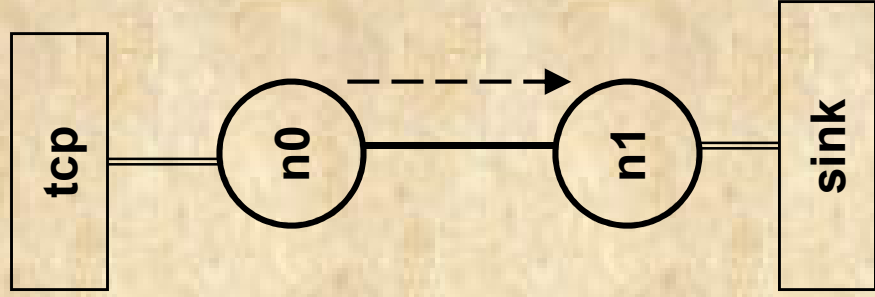
Step 6: Creating Traffic (On Top of UDP)

- CBR
 - `set cbr [new Application/Traffic/CBR]`
 - `$cbr set packetSize_ 500`
 - `$cbr set rate_ 2Mb`
 - `$cbr attach-agent $udp`



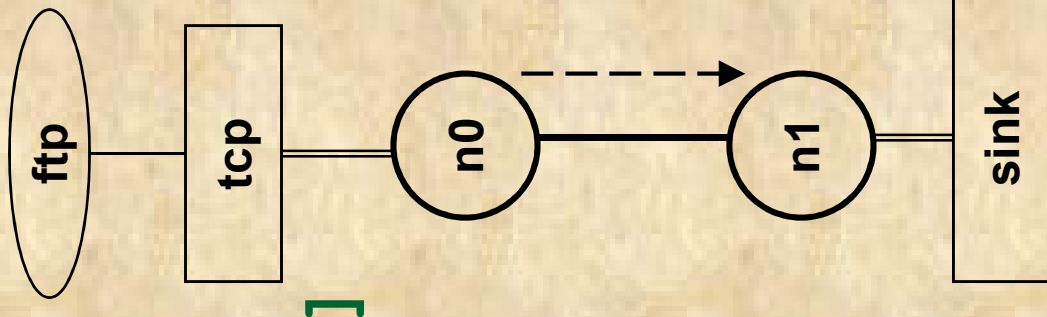
Creating TCP connection

```
set tcp [new Agent/TCP]
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $tcpsink
$ns connect $tcp $tcpsink
```



Step 6: Creating Traffic (On Top of TCP)

- FTP
 - `set ftp [new Application/FTP]`
 - `$ftp attach-agent $tcp`
- Telnet
 - `set telnet [new Application/Telnet]`
 - `$telnet attach-agent $tcp`



Recall: Generic Script Structure

1. set ns [new Simulator]
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation



Examples

Post-Processing Procedures

- Add a 'finish' procedure that closes the trace file and starts nam.

```
proc finish {} {  
    global ns nf  
    $ns flush-trace  
    close $nf  
    exec nam out.nam &  
    exit 0  
}
```

Run Simulation

- Schedule Events
 - `$ns at <time> <event>`
 - `<event>`: any legitimate ns/tcl commands
 - `$ns at 0.5 "$cbr start"`
 - `$ns at 4.5 "$cbr stop"`
- Call 'finish'
 - `$ns at 5.0 "finish"`
- Run the simulation
 - `$ns run`

Recall: Generic Script Structure

1. set ns [new Simulator]
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation



Examples

Step2: Tracing

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id
-------	------	-----------	---------	----------	----------	-------	-----	----------	----------	---------	--------

```
r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop (at queue)
```

```
src_addr : node.port (3.0)
dst_addr : node.port (0.0)
```

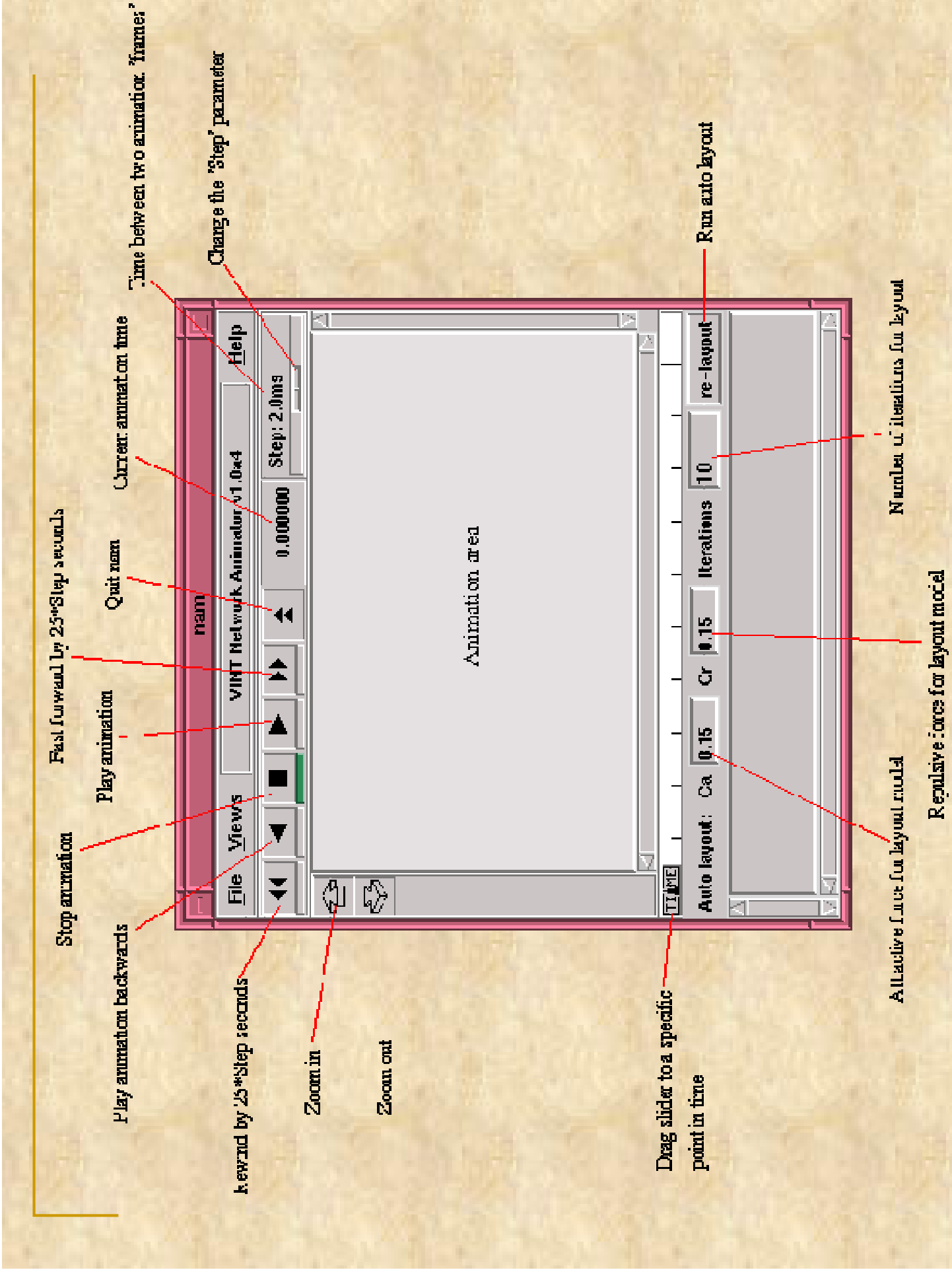
```
r 1.3556 3 2 ack 40 ----- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ----- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ----- 2 1.0 3.1 157 207
```

NS-2 Generic Script Structure

1. Create Simulator object
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Visualization Tools

- nam-1 (Network Animator Version 1)
 - Packet-level animation
 - Well supported by ns
- xgraph
 - Simulation results



nam Interface: Nodes

- Color
`$node color red`
- Shape (can't be changed after sim starts)
`$node shape box (circle, box, hexagon)`
- Label (single string)
`$ns at 1.1 "$n0 label \"web cache 0\""`

nam Interfaces: Links

- Color
 - `$ns duplex-link-op $n0 $n1 color`
 - `"green"`
- Label
 - `$ns duplex-link-op $n0 $n1 label`
 - `"backbone"`

nam Interface: Topology Layout

- “Manual” layout: specify everything

```
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(1) $n(2) orient right
$ns duplex-link-op $n(2) $n(3) orient right
$ns duplex-link-op $n(3) $n(4) orient 60deg
```

- If anything missing → automatic layout

Simulation Example

