Wireless LANs
June – September 2009

Network Simulator (NS2)

รศ. ดร. อนันต์ ผลเพิ่ม

Assoc. Prof. Anan Phonphoem, Ph.D.

anan.p@ku.ac.th

http://www.cpe.ku.ac.th/~anan

Computer Engineering Department

Kasetsart University, Bangkok, Thailand

Slide materials

Modified from many good sources:

- Padmaparna Haldar and Xuan Chen, ISI, 2002

  http://www.isi.edu/nsnam/ns/ns-tutorial/tutorial-02/slides/NS_Fundamentals_1.ppt

- Choe, Hyun Jung (Stella), U of Texas, Arlington, 2008

  http://crystal.uta.edu/~zaruba/CSE5346/ns2Tutorial-2008spring_v1.1.ppt

Outline

- Overview
- Running NS

NS Overview

- VINT Project (1995)
  - UC Berkeley, LBL, USC/ISI, and Xerox PARC
- Goals
  - Support networking research and education
  - Freely distributed (open source)
  - Protocol / model verification and comparison

NS Overview

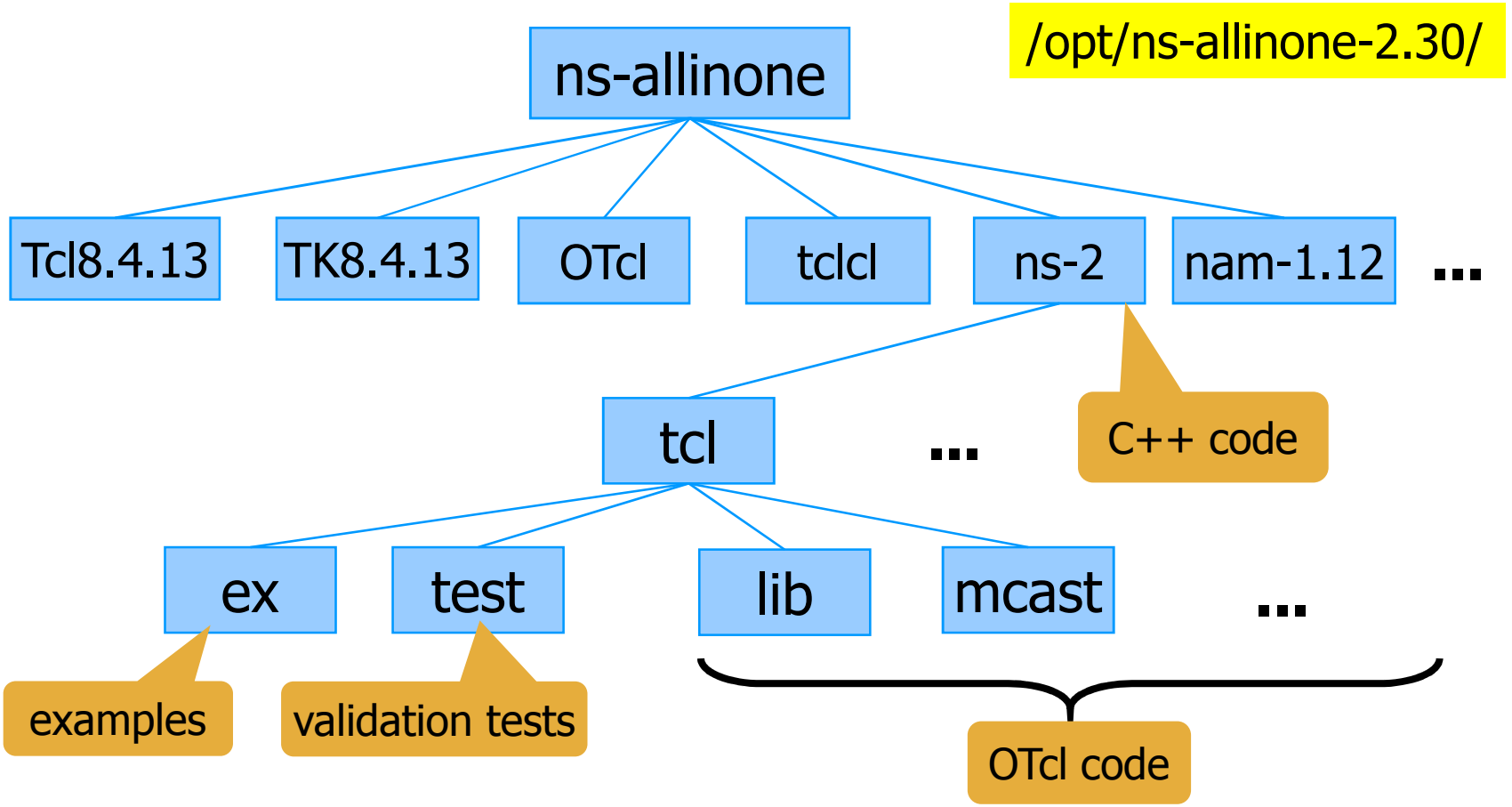
- Discrete Event Simulator
- Link layer and up
- Support wired and wireless

Current Status (Sep 2009)

Periodical release	Current release	Description
NS2	ns-2.34, June 14, 2009 http://www.isi.edu/nsnam/ns/ns-build.html	• **C++** and **OTcl** • test suites and examples • ns manual
NS3	ns-3.5, July 4, 2009 http://www.nsnam.org/	• **C++** and **Python** • Next release: ns-3.6, October 1, 2009 (multi-freq in WiFi, IPv6)

Platform support	FreeBSD, Linux, Solaris Windows (using cygwin) Mac

NS-2 Directory Structure



Functionalities and Models

- Wired (point-to-point, LAN)
  - **Routing:** DV, LS, PIM-SM etc.
  - **Transportation:** TCP (Reno, Vegas, etc.) and UDP
  - **Traffic sources:** web, ftp, telnet, cbr, stochastic etc.
  - **Queuing disciplines:** drop-tail, RED, FQ, SFQ, DRR etc.
  - **QoS:** IntServ and Diffserv …
  - Emulation
- Wireless
  - Ad hoc routing, mobile IP and Satellite
  - Directed diffusion, sensor-MAC

Discrete Event Simulation

- Model world as **events**
  - Simulator has list of events
  - Process: take next one, run it, until done
  - Each event happens in virtual (simulated) time
    ➔ but takes an arbitrary real time
- Ns uses simple model
  - single thread of control
    ➔ no locking or race conditions (very easy)

Discrete Event Examples

Consider two nodes on an Ethernet:

simple queuing model:

t=1, A enqueues pkt on LAN
t=1.01, LAN dequeues pkt and triggers B

detailed **CSMA/CD** model:

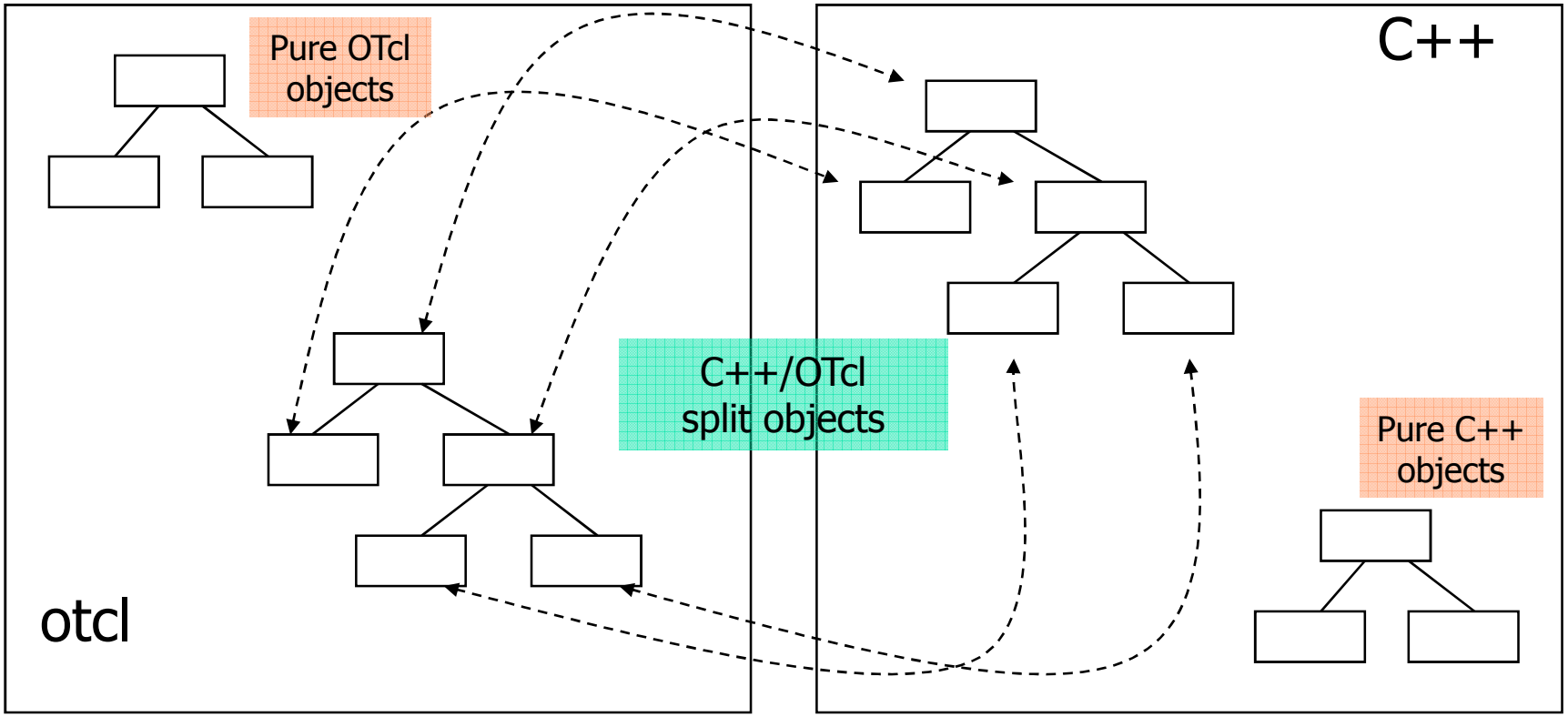
t=1.0: A sends pkt to NIC
A's NIC starts carrier sense
t=1.005: A's NIC concludes cs, starts tx
t=1.006: B's NIC begins reciving pkt
t=1.01: B's NIC concludes pkt
B's NIC passes pkt to app

Architecture

- Object-oriented (C++, Otcl)
- data / control separation
  - **C++** for **data**
    - per packet processing, core of *ns*
    - fast to run, detailed, complete control
  - **OTcl** for **control**
    - Simulation scenario configurations
    - Periodic or triggered action
    - Manipulating existing C++ objects
    - fast to write and change

\+ **running vs. writing speed**
– **Learning and debugging (two languages)**

OTcl and C++: The Duality



Basic Tcl

variables:

```
set x 10
puts “x is $x”
```

functions and expressions:

```
set y [pow x 2]
set y [expr x*x]
```

control flow:

```
if {$x > 0} { return $x }
   else {
   return [expr -$x] }
while { $x > 0 } {
   puts $x
   incr x –1
```

procedures:

```
proc pow {x n} {
   if {$n == 1} { return $x }
   set part [pow x [expr $n-1]]
   return [expr $x*$part]
}
```

Also lists, associative arrays, etc.

=> can use a real programming language to build network topologies, traffic models, etc.

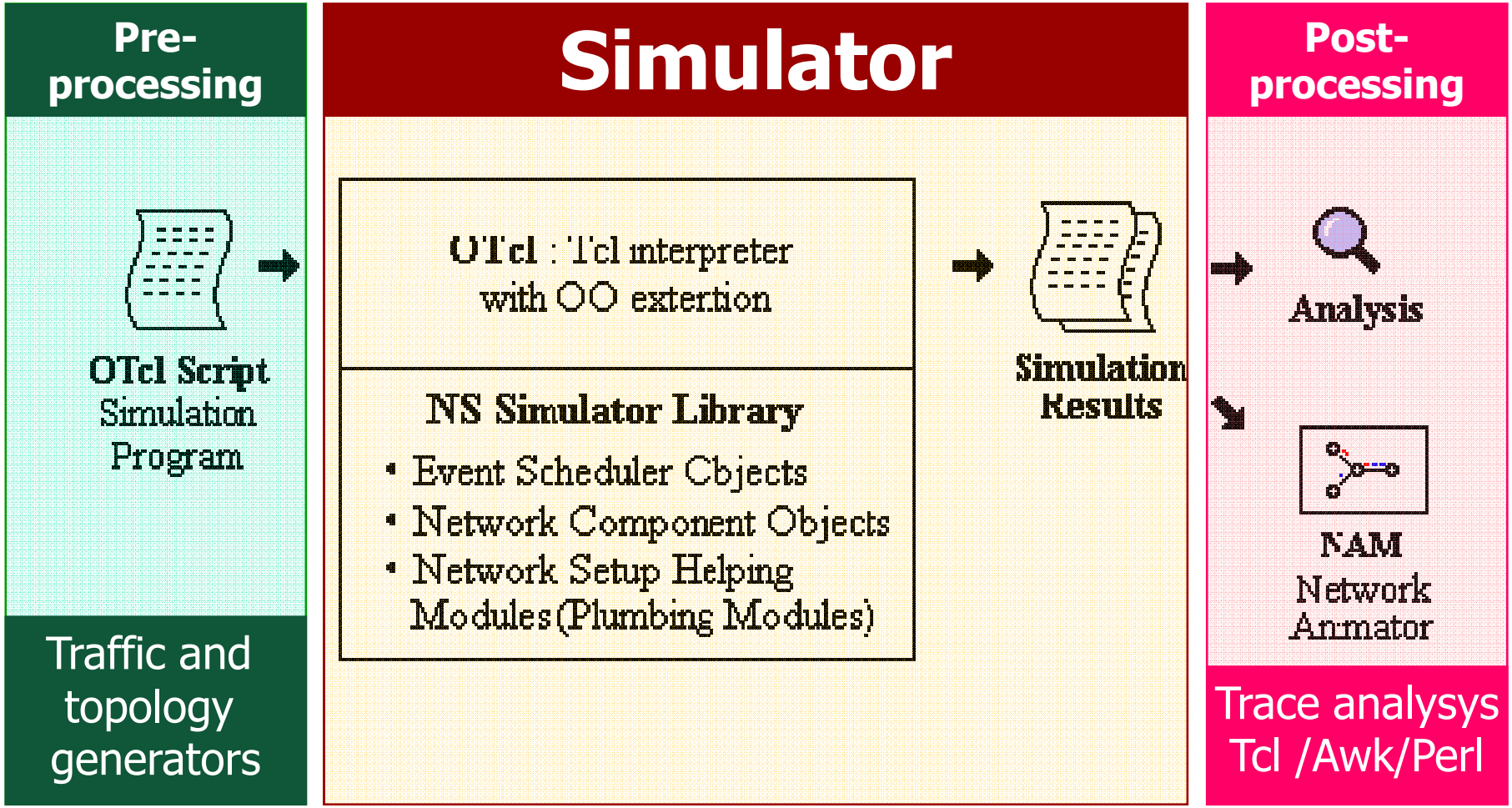
Basic OTcl

```
Class Person

# constructor:
Person instproc init {age} {
        $self instvar age_
        set age_ $age
}

# method:
Person instproc greet {} {
        $self instvar age_
        puts “$age_ years old:
How are you doing?”
}
```

```
# subclass:
Class Kid -superclass Person
Kid instproc greet {} {
   $self instvar age_
   puts “$age_ years old kid:
   What’s up, dude?”
}
```

```
set a [new Person 45]
set b [new Kid 15]
$a greet
$b greet
```

NS Components



Outline

- Overview
- Running NS

Running NS

- Create the event scheduler
- Turn on tracing
- Create network
- Setup routing
- Insert errors
- Create transport connection
- Create traffic
- Transmit application-level data

Creating Event Scheduler

- Create event scheduler

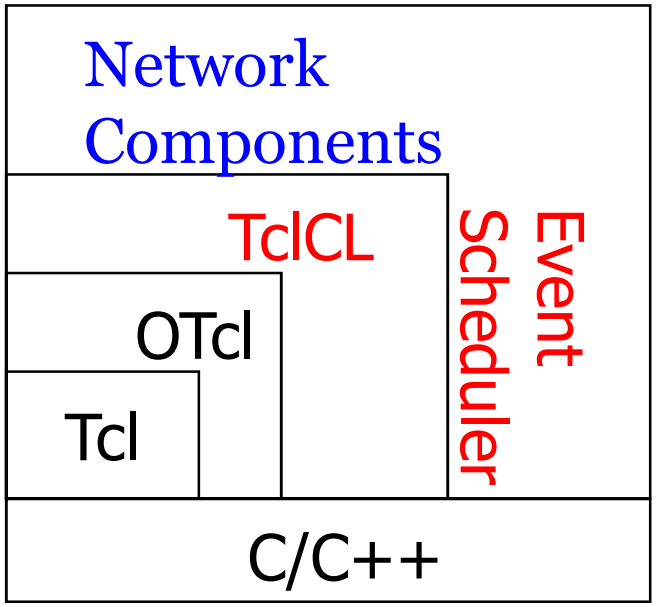
```
set ns [new Simulator]
```

- Schedule events

```
$ns at <time> <event>
$ns at 5.0 “finish”
```

- Start scheduler

```
$ns run
```



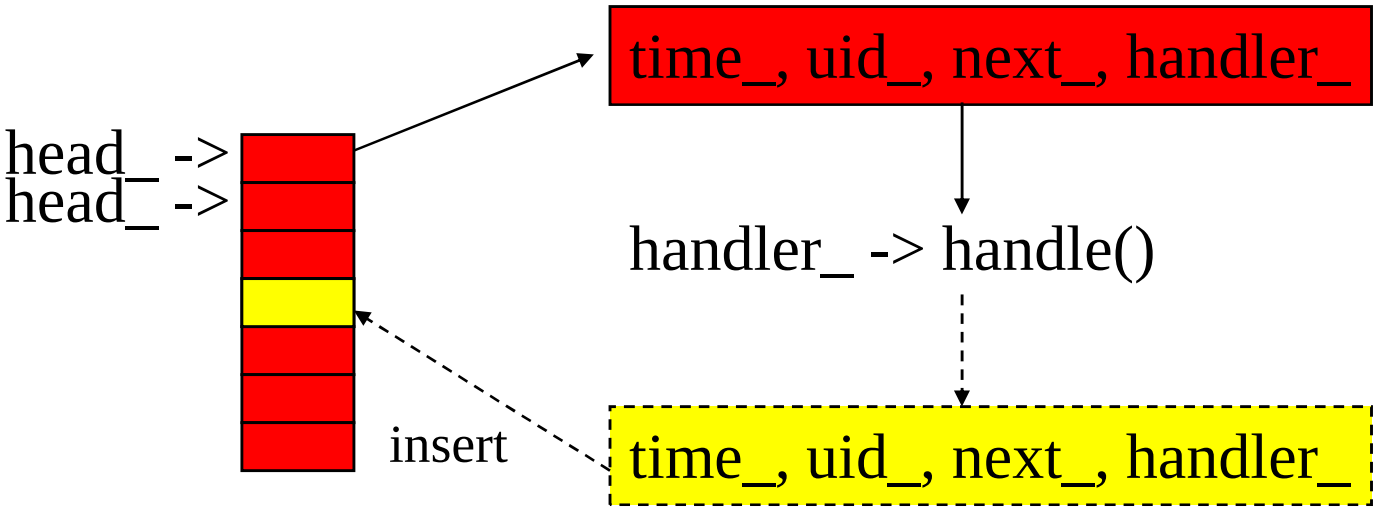
NS-2

Event Scheduler

- Event: at-event and packet
- List scheduler: default
  - Heap and calendar queue scheduler
- Real-time scheduler
  - Synchronize with real-time
  - Network emulation

```
set ns_ [new Simulator]
$ns_ use-scheduler Heap
$ns_ at 300.5 “$self halt”
```

Discrete Event Scheduler



Hello World - Interactive Mode

Interactive mode:

```
swallow 71% ns
% set ns [new
   Simulator]
_o3
% $ns at 1 “puts
   \“Hello World!\””
1
% $ns at 1.5 “exit”
2
% $ns run
Hello World!
swallow 72%
```

Batch mode:

simple.tcl

```
   set ns [new Simulator]
   $ns at 1 “puts \“Hello
     World!\””
   $ns at 1.5 “exit”
   $ns run
```

```
swallow 74% ns simple.tcl
Hello World!
swallow 75%
```

Creating Network

- Nodes

```
set n0 [$ns node]
set n1 [$ns node]
```

- Links and queuing

```
$ns duplex-link $n0 $n1 <bandwidth>
  <delay> <queue_type>
```

<queue_type>: DropTail, RED, CBQ, FQ, SFQ, DRR

```
$ns duplex-link-op $no $n1 OPTIONS
```

Setup Routing

- Unicast

  `$ns rtproto <type>`

  <type>: Static, Session, DV, cost, multi-path

- Multicast

  `$ns multicast` (right after [new Simulator])

  `$ns mrtproto <type>`

  <type>: CtrMcast, DM, ST, BST

Creating Connection: UDP

- UDP

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

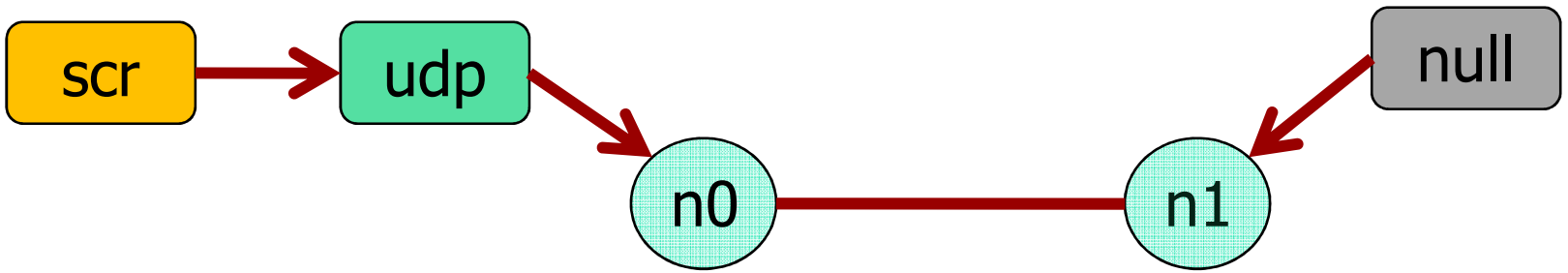
Creating Traffic: On Top of UDP

- CBR

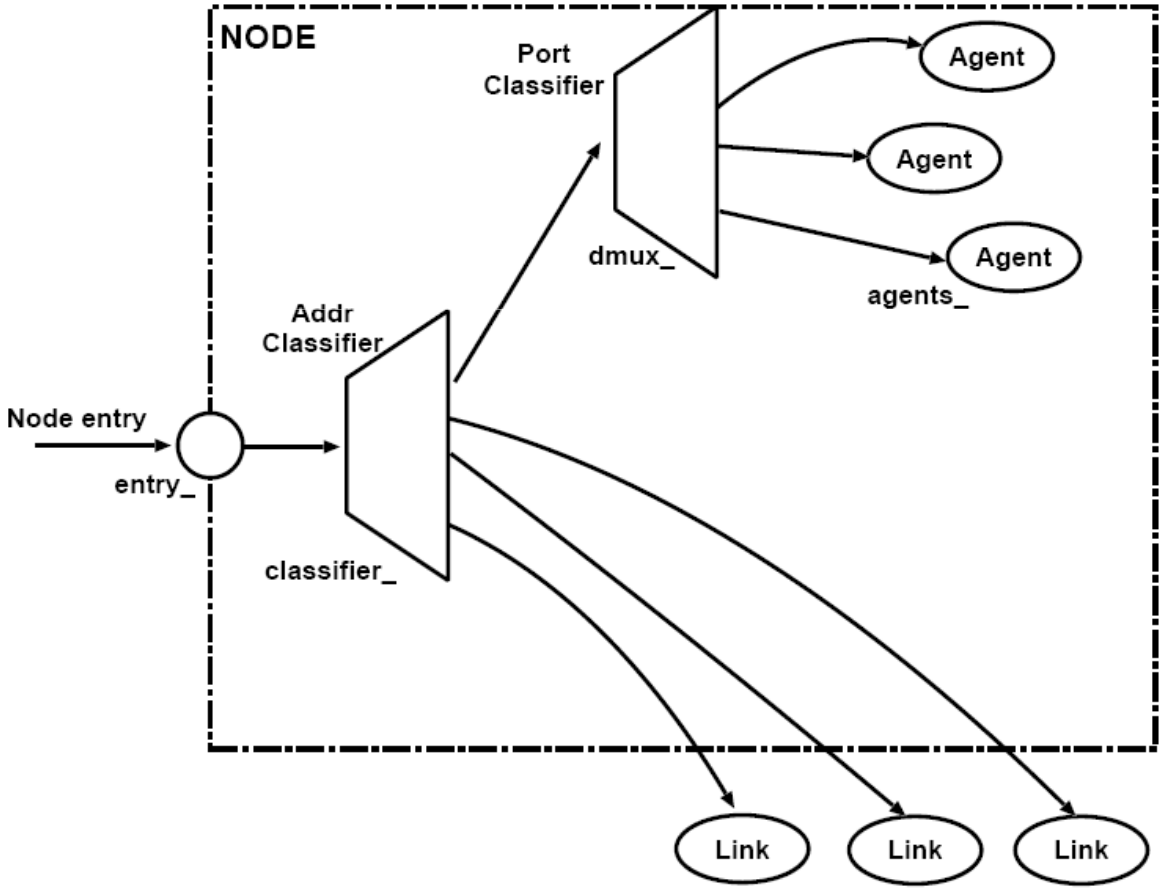
```
set src [new Application/Traffic/CBR]
$src attach-agent $udp
```

- Exponential or Pareto on-off

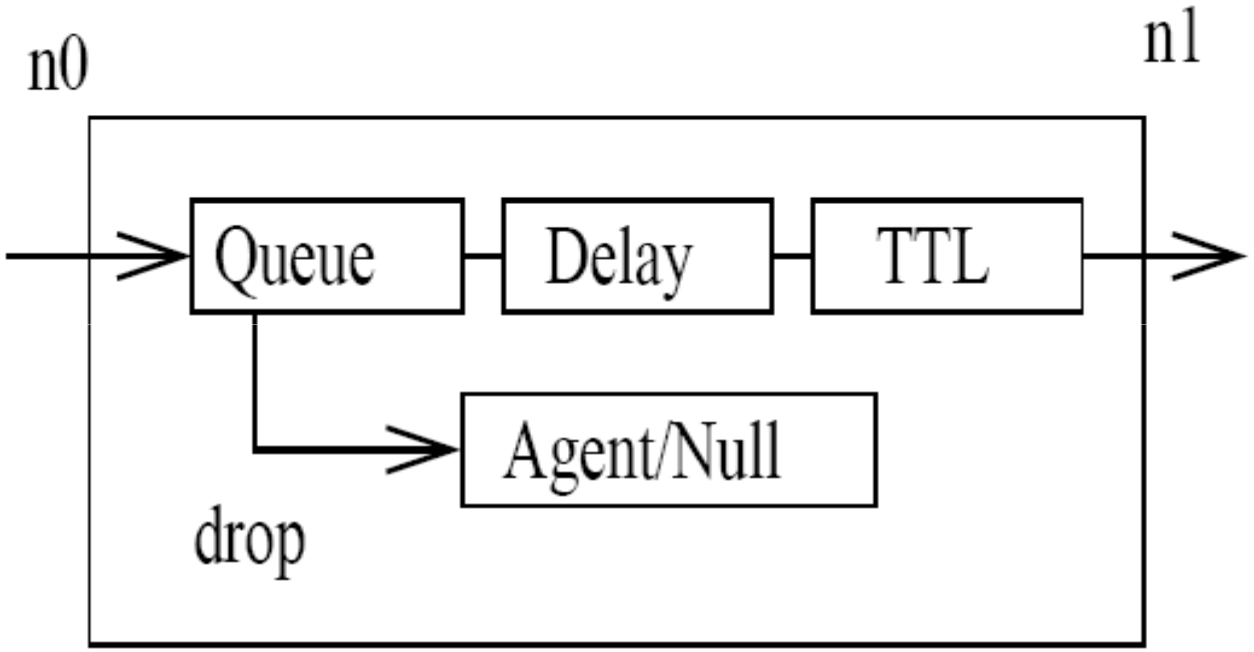
```
set src [new Application/Traffic/Exponential]
set src [new Application/Traffic/Pareto]
$src attach-agent $udp
```



Structure of a Unicast Node



Structure of a Simplex Link



Creating Connection: TCP

- TCP

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

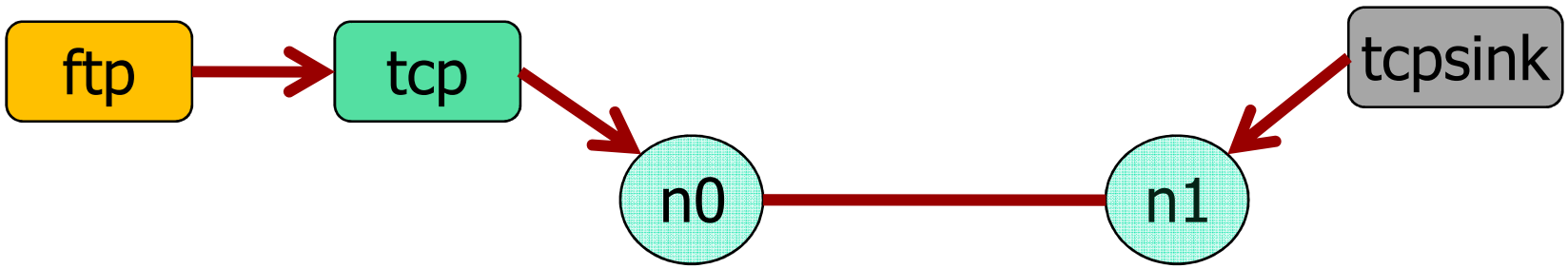
Creating Traffic: On Top of TCP

- FTP

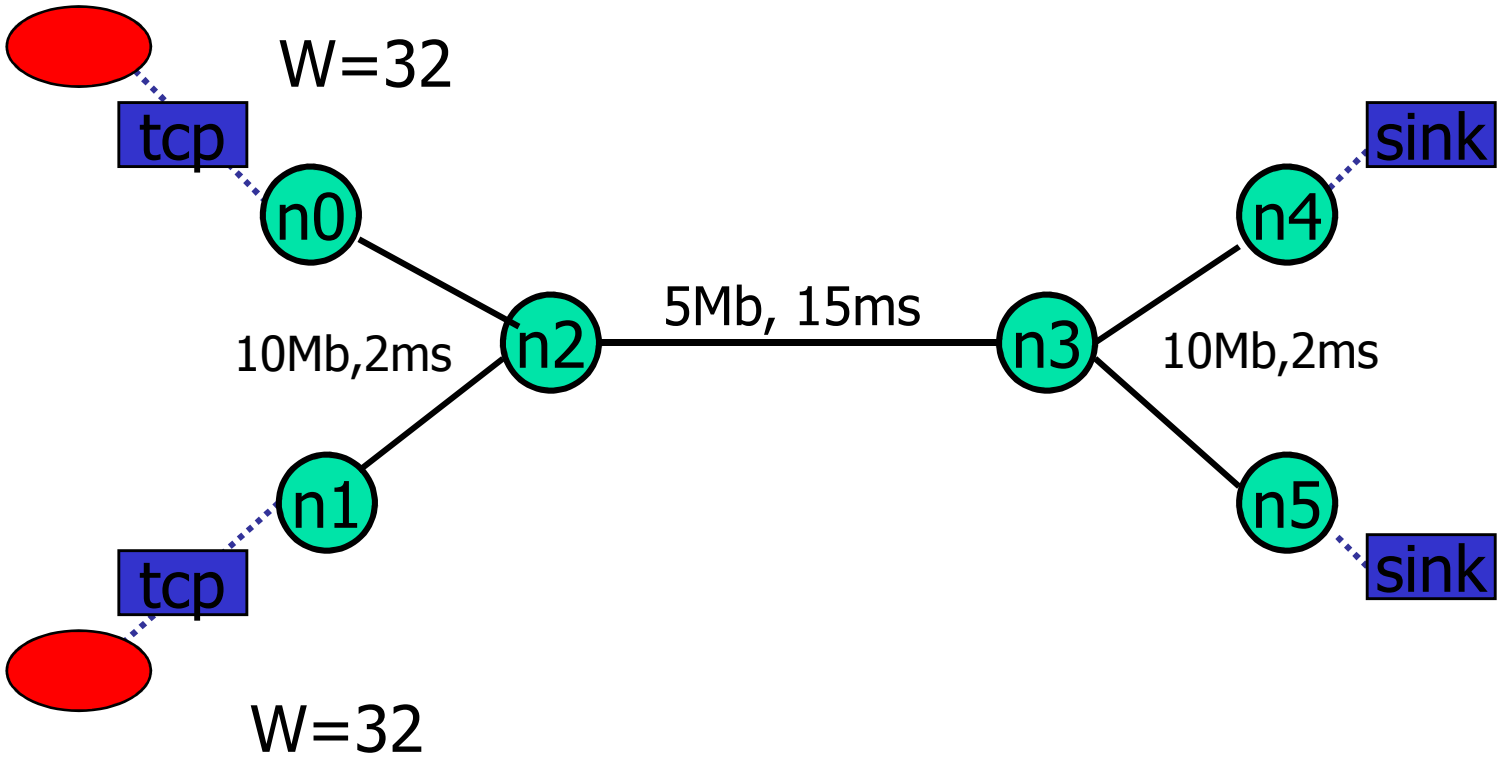
```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

- Telnet

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



Example


W=32
tcp
n0
10Mb,2ms
n2
5Mb, 15ms
n3
n4
sink
10Mb,2ms
n1
n5
tcp
sink
W=32

TCP Agents

- ns has several variants of TCP available:
  - -- Agent/TCP/Tahoe a ``tahoe'' TCP sender
  - -- Agent/TCP/Reno a ``Reno'' TCP sender
  - -- Agent/TCP/NewReno Reno with a modification
  - -- Agent/TCP/Sack1 TCP with selective repeat (follows RFC2018)
  - -- Agent/TCP/Vegas TCP Vegas
  - -- Agent/TCP/Fack Reno TCP with ``forward acknowledge ment''
- The oneway TCP receiving agents currently supported are:
  - -- Agent/TCPSink TCP sink with one ACK per packet
  - -- Agent/TCPSink/DelAck TCP sink with configurable delay per ACK
  - -- Agent/TCPSink/Sack1 selective ACK sink (follows RFC2018)
  - -- Agent/TCPSink/Sack1/DelAck Sack1 with DelAck
- The twoway experimental sender currently supports only a Reno form of TCP:
  - -- Agent/TCP/FullTcp

TCP Agent Parameters

- Agent/TCP set tcpTick— 0.1 ;# timer granularity in sec (.1 is NONST ANDARD)
- Agent/TCP set maxrto— 64 ;# bound on RTO (seconds)
- Agent/TCP set dupacks— 0 ;# duplicate ACK counter
- Agent/TCP set ack— 0 ;# highest ACK received
- Agent/TCP set cwnd— 0 ;# congestion window (packets)
- Agent/TCP set awnd— 0 ;# averaged cwnd (experimental)
- Agent/TCP set ssthresh— 0 ;# slowstat threshold (packets)
- Agent/TCP set rtt— 0 ;# rtt sample
- Agent/TCP set srtt— 0 ;# smoothed (averaged) rtt
- Agent/TCP set rttvar— 0 ;# mean deviation of rtt samples
- Agent/TCP set backoff— 0 ;# current RTO backoff factor
- Agent/TCP set maxseq— 0 ;# max (packet) seq number sent

Tracing

- Trace packets on all links
- #Open the **NAM trace** file

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

#Open the **Trace file**

```
set tf [open out.tr w]
$ns trace-all $tf
```

- Must appear immediately after creating scheduler
- Turn on tracing on specific links

```
$ns trace-queue $n0 $n1
$ns namtrace-queue $n0 $n1
```

- Event tracing (support TCP right now)
  - Record "event" in trace file: `$ns eventtrace-all`

Trace file

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
```

Monitoring

- Queue monitor

```
set qmon [$ns monitor-queue $n0 $n1 $q_f $sample_interval]
```

  - Get statistics for a queue

```
$qmon set pdrops_
```

  - Record to trace file as an optional

```
29.000000000000142 0 1 0.0 0.0 4 4 0 1160 1160 0
```

- Flow monitor

```
set fmon [$ns_ makeflowmon Fid]
$ns_ attach-fmon $slink $fmon
$fmon set pdrops_
```

nam Tracing

- Variable tracing in nam

  Agent/TCP set nam_tracevar_ true

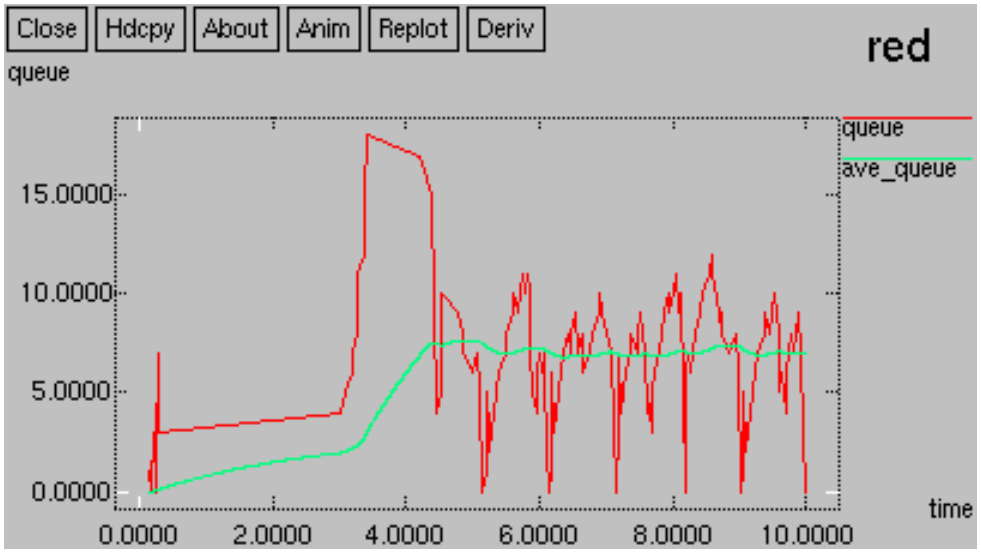
```
$tcp tracevar srtt_
$tcp tracevar cwnd_
```

- Monitor agent variables in nam

```
$ns add-agent-trace $tcp $tcp
$ns monitor-agent-trace $tcp
$srm0 tracevar cwnd_
……
$ns delete-agent-trace $tcp
```

Visualization Tools

- nam-1 (Network AniMator Version 1)
  - Packet-level animation
  - Well supported by ns
- xgraph
  - Conversion from ns trace to xgraph format



xgraph

- The *xgraph* program draws a graph on an X display given data read from either data files.
  - To run it: xgraph dataFileName
- You can save the hardcopy of the graph as a postscript file.
- Xgraph is available on gamma:
  - /opt/ns-allinone-2.30/xgraph-12.1

nam

- Basic visualization
  - Topology layout
  - Animation control
  - Synchronous replay
- Fine-tune layout
- TCP/SRM visualization
- Editor: generate ns simulation scripts

ns→nam Interface

- Color
- Node manipulation
- Link manipulation
- Topology layout
- Protocol state
- Misc

nam Interface: Color

- Color mapping

```
$ns color 1 red
$ns color 2 blue
$ns color 3 chocolate
```

- Color ↔ flow id association

```
$tcp0 set fid_ 1  ;# red packets
$tcp1 set fid_ 2  ;# blue packets
```

nam Interface: Nodes

- Color

```
$node color red
```

- Shape (can't be changed after sim starts)

```
$node shape box      ;# circle, box, hexagon
```

- Marks (concentric "shapes")

```
$ns at 1.0 "$n0 add-mark m0 blue box"
$ns at 2.0 "$n0 delete-mark m0"
```

- 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 "abced"
```

- Dynamics (automatically handled)

```
$ns rtmodel Deterministic {2.0 0.9 0.1} $n0 $n1
```

- Asymmetric links not allowed

nam Interface: Topo 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

nam Interface: Protocol State

- Monitor values of agent variables

```
$ns add-agent-trace $srm0 srm_agent0
$ns monitor-agent-trace $srm0
$srm0 tracevar C1_
$srm0 tracevar C2_
# … …
$ns delete-agent-trace $tcp1
```

nam Interface: Misc

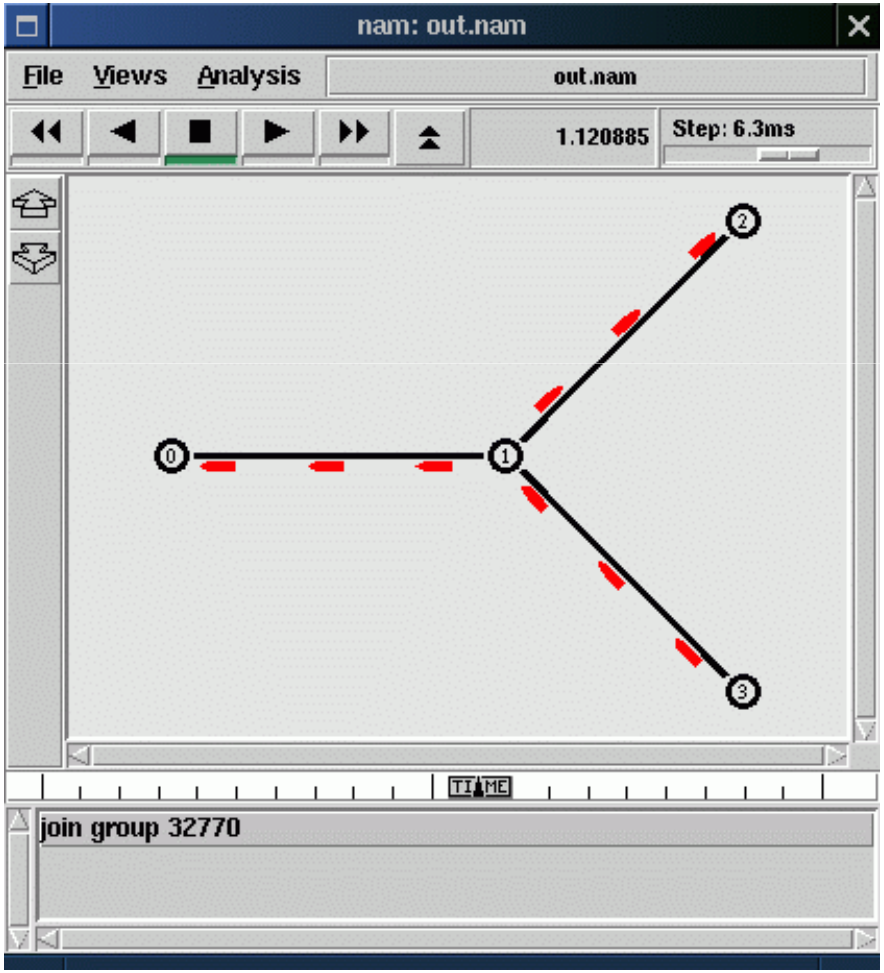
- Annotation
  - Add textual explaination to your sim

```
$ns at 3.5 "$ns trace-annotate \“packet drop\""
```

- Set animation rate

```
$ns at 0.0 "$ns set-animation-rate 0.1ms"
```

nam


nam: out.nam
File
Views
Analysis
out.nam
1.120885
Step: 6.3ms
0
1
2
3
TIME
join group 32770

Summary: Generic Script Structure

```
set ns [new Simulator]
# [Turn on tracing]
# Create topology
# Setup packet loss, link dynamics
# Create routing agents
# Create:
#   - multicast groups
#   - protocol agents
#   - application and/or setup traffic sources
# Post-processing procs
# Start simulation
```