Simulation Methodology

- Plan:
  - Introduce basics of simulation modeling
  - Define terminology and methods used
  - Introduce simulation paradigms
    - Time-driven simulation
    - Event-driven simulation
    - Monte Carlo simulation
  - Technical issues for simulations
    - Random number generation
    - Statistical inference

Time-Driven Simulation

- Time advances in fixed size steps
- Time step = smallest unit in model
- Check each entity to see if state changes
- Well-suited to continuous systems
  - e.g., river flow, factory floor automation
- Granularity issue:
  - Too small: slow execution for model
  - Too large: miss important state changes

Event-Driven Simulation (1 of 2)

- Discrete-event simulation (DES)
- System is modeled as a set of entities that affect each other via events (msgs)
- Each entity can have a set of states
- Events happen at specific points in time (continous or discrete), and trigger state changes in the system
- Very general technique, well-suited to modeling discrete systems (e.g, queues)

Event-Driven Simulation (2 of 2)

- Typical implementation involves an event list, ordered by time
- Process events in (non-decreasing) timestamp order, with seed event at t=0
- Each event can trigger 0 or more events
  - Zero: “dead end” event
  - One: “sustaining” event
  - More than one: “triggering” event
- Simulation ends when event list is null, or desired time duration has elapsed

Monte Carlo Simulation

- Estimating an answer to some difficult problem using numerical approximation, based on random numbers
- Examples: numerical integration, primality testing, WSN coverage
- Suited to stochastic problems in which probabilistic answers are acceptable
- Might be one-sided answers (e.g., prime)
- Can bound probability to some epsilon

Summary

- Simulation methods offer a range of general-purpose approaches for performance evaluation
- Simulation modeler must determine the appropriate aspects of system to model

Queueing Theory

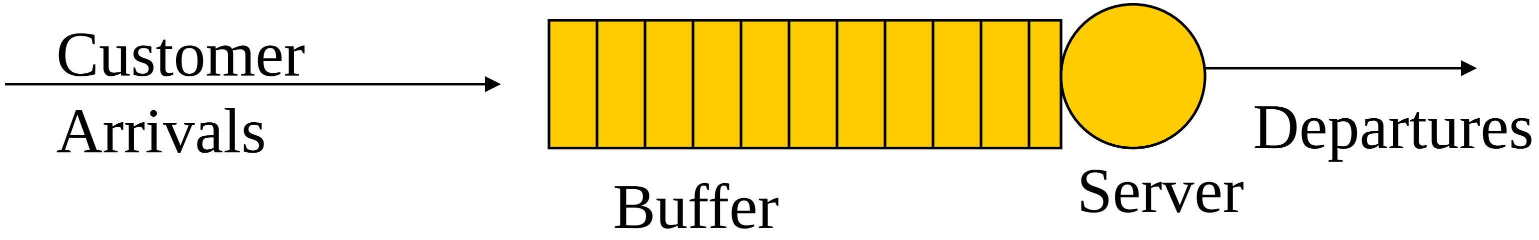
- Plan:
  - Introduce basics of Queueing Theory
  - Define notation and terminology used
  - Discuss properties of queuing models
  - Show examples of queueing analysis:
    - M/M/1 queue
    - Variations on the M/G/1 queue
    - Open queueing network models
    - Closed queueing network models

Queueing Theory Basics

- Queueing theory provides a very general framework for modeling systems in which customers must line up (queue) for service (use of resource)
  - Banks (tellers)
  - Restaurants (tables and seats)
  - Computer systems (CPU, disk I/O)
  - Networks (Web server, router, WLAN)

Queue-based Models

- Queueing model represents:
  - Arrival of jobs (customers) into system
  - Service time requirements of jobs
  - Waiting of jobs for service
  - Departures of jobs from the system
- Typical diagram:



Why Queue-based Models?

- In many cases, the use of a queuing model provides a quantitative way to assess system performance
  - Throughput (e.g., job completions per second)
  - Response time (e.g., Web page download time)
  - Expected waiting time for service
  - Number of buffers required to control loss
- Reveals key system insights (properties)
- Often with efficient, closed-form calculation

Caveats and Assumptions

- In many cases, using a queuing model has the following implicit underlying assumptions:
  - Poisson arrival process
  - Exponential service time distribution
  - Single server
  - Infinite capacity queue
  - First-Come-First-Serve (FCFS) discipline (also known as FIFO: First-In-First-Out)
- Note: important role of memoryless property!

Advanced Queueing Models

- There is TONS of published work on variations of the basic model:
  - Correlated arrival processes
  - General (G) service time distributions
  - Multiple servers
  - Finite capacity systems
  - Other scheduling disciplines (non-FIFO)
- We will start with the basics!

Queue Notation

- Queues are concisely described using the Kendall notation, which specifies:
  - Arrival process for jobs {M, D, G, …}
  - Service time distribution {M, D, G, …}
  - Number of servers {1, n}
  - Storage capacity (buffers) {B, infinite}
  - Service discipline {FIFO, PS, SRPT, …}
- Examples: M/M/1, M/G/1, M/M/c/c

The M/M/1 Queue

- Assumes Poisson arrival process, exponential service times, single server, FCFS service discipline, infinite capacity for storage, with no loss
- Notation: M/M/1
  - Markovian arrival process (Poisson)
  - Markovian service times (exponential)
  - Single server (FCFS, infinite capacity)

The M/M/1 Queue (cont'd)

- Arrival rate: λ (e.g., customers/sec)
  - Inter-arrival times are exponentially distributed (and independent) with mean $1 / \lambda$
- Service rate: μ (e.g., customers/sec)
  - Service times are exponentially distributed (and independent) with mean $1 / \mu$
- System load: $\rho = \lambda / \mu$

  $0 \leq \rho \leq 1$ (also known as utilization factor)
- Stability criterion: $\rho < 1$ (single server systems)

Queue Performance Metrics

- N: Avg number of customers in system as a whole, including any in service
- Q: Avg number of customers in the queue (only), excluding any in service
- W: Avg waiting time in queue (only)
- T: Avg time spent in system as a whole, including wait time plus service time
- Note: Little's Law: $N = \lambda T$

The M/D/1 Queue

- Assumes Poisson arrival process, deterministic (constant) service times, single server, FCFS service discipline, infinite capacity for storage, no loss
- Notation: M/D/1
  - Markovian arrival process (Poisson)
  - Deterministic service times (constant)
  - Single server (FCFS, infinite capacity)

The M/G/1 Queue

- Assumes Poisson arrival process, general service times, single server, FCFS service discipline, infinite capacity for storage, with no loss
- Notation: M/G/1
  - Markovian arrival process (Poisson)
  - General service times (must specify F(x))
  - Single server (FCFS, infinite capacity)

The G/G/1 Queue

- Assumes general arrival process, general service times, single server, FCFS service discipline, infinite capacity for storage, with no loss
- Notation: G/G/1
  - General arrival process (specify G(x))
  - General service times (must specify F(x))
  - Single server (FCFS, infinite capacity)

Queueing Network Models

- So far we have been talking about a queue in isolation
- In a queueing network model, there can be multiple queues, connected in series or in parallel (e.g., CPU, disk, teller)
- Two versions:
  - Open queueing network models
  - Closed queueing network models

Open Queueing Network Models

- Assumes that arrivals occur externally from outside the system
- Infinite population, with a fixed arrival rate, regardless of how many in system
- Unbounded number of customers are permitted within the system
- Departures leave the system (forever)

Closed Queueing Network Models

- Assumes that there is a finite number of customers, in a self-contained world
- Finite population; arrival rate varies depending on how many and where
- Fixed number of customers (N) that recirculate in the system (forever)
- Can be analyzed using Mean Value Analysis (MVA) and balance equations

PERFORMANCE EVALUATION

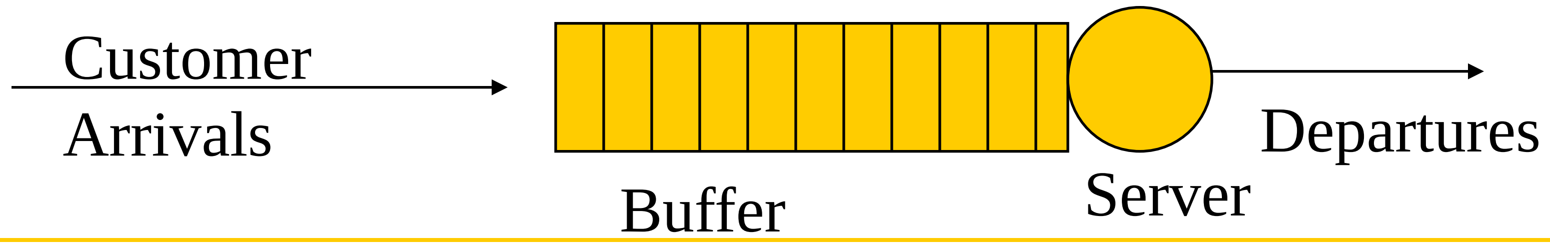
- Often in Computer Science you need to:
  - demonstrate that a new concept, technique, or algorithm is feasible
  - demonstrate that a new method is better than an existing method
  - understand the impact of various factors and parameters on the performance, scalability, or robustness of a system

PERF EVAL: THE BASICS

- There are three main methods used in the design of performance evaluation studies:
- Analytic approaches
  - the use of mathematics, Markov chains, queueing theory, Petri Nets, abstract models…
- Simulation approaches
  - design and use of computer simulations and simplified models to assess performance
- Experimental approaches
  - measurement and use of a real system

Analytical Example: Queueing Theory

- Queueing theory is a mathematical technique that specializes in the analysis of queues (e.g., customer arrivals at a bank, jobs arriving at CPU, I/O requests arriving at a disk subsystem, lineup at Tim Hortons)
- General diagram:



Queueing Theory (cont'd)

- The queueing system is characterized by:
  - Arrival process (M, G)
  - Service time process (M, D, G)
  - Number of servers (1 to infinity)
  - Number of buffers (infinite or finite)
- Example notation: M/M/1, M/D/1
- Example notation: M/M/∞ , M/G/1/k

Simulation Example: TCP Throughput

- Can use an existing simulation tool, or design and build your own custom simulator
- Example: ns-2 network simulator
- A discrete-event simulator with detailed TCP protocol models
- Configure network topology and workload
- Run simulation using pseudo-random numbers and produce statistical output

OTHER ISSUES

- Simulation run length
  - choosing a long enough run time to get statistically meaningful results (equilibrium)
- Simulation start-up effects and end effects
  - deciding how much to “chop off” at the start and end of simulations to get proper results
- Replications
  - ensure repeatability of results, and gain greater statistical confidence in the results given

Experimental Example: Benchmarking

- The design of a performance study requires great care in experimental design and methodology
- Need to identify
  - experimental <u>factors</u> to be tested
  - <u>levels</u> (settings) for these factors
  - performance <u>metrics</u> to be used
  - <u>experimental design</u> to be used

FACTORS

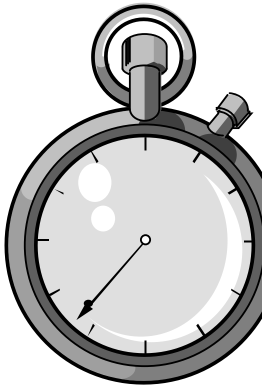
- Factors are the main "components" that are varied in an experiment, in order to understand their impact on performance
- Examples: request rate, request size, read/write ratio, num concurrent clients
- Need to choose factors properly, since the number of factors affects size of study

LEVELS

- Levels are the precise settings of the factors that are to be used in an experiment
- Examples: req size S = 1 KB, 10 KB, 1 MB
- Example: num clients C = 10, 20, 30, 40, 50
- Need to choose levels realistically
- Need to cover useful portion of the design space

PERFORMANCE METRICS

- Performance metrics specify what you want to measure in your performance study
- Examples: response time, throughput, pkt loss
- Must choose your metrics properly and instrument your experiment accordingly



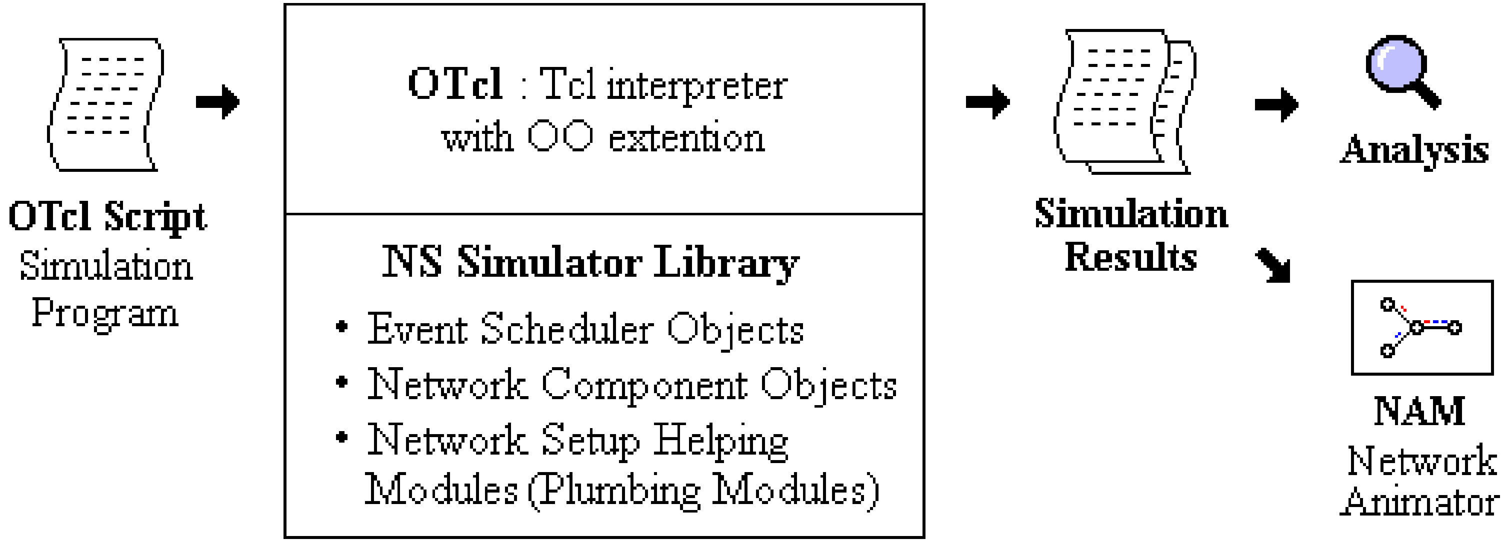
An Introduction to NS-2*

NS-2 Learning Resources

- Installation instructions
- Using related tools (nam, xgraph, etc)
- NS-2 official website and documentation
- Tutorials to get you started
- Sample coding exercises

What is ns?

- Object-oriented, discrete event-driven network simulator
- Written in C++ and OTcl
- By VINT: Virtual InterNet Testbed

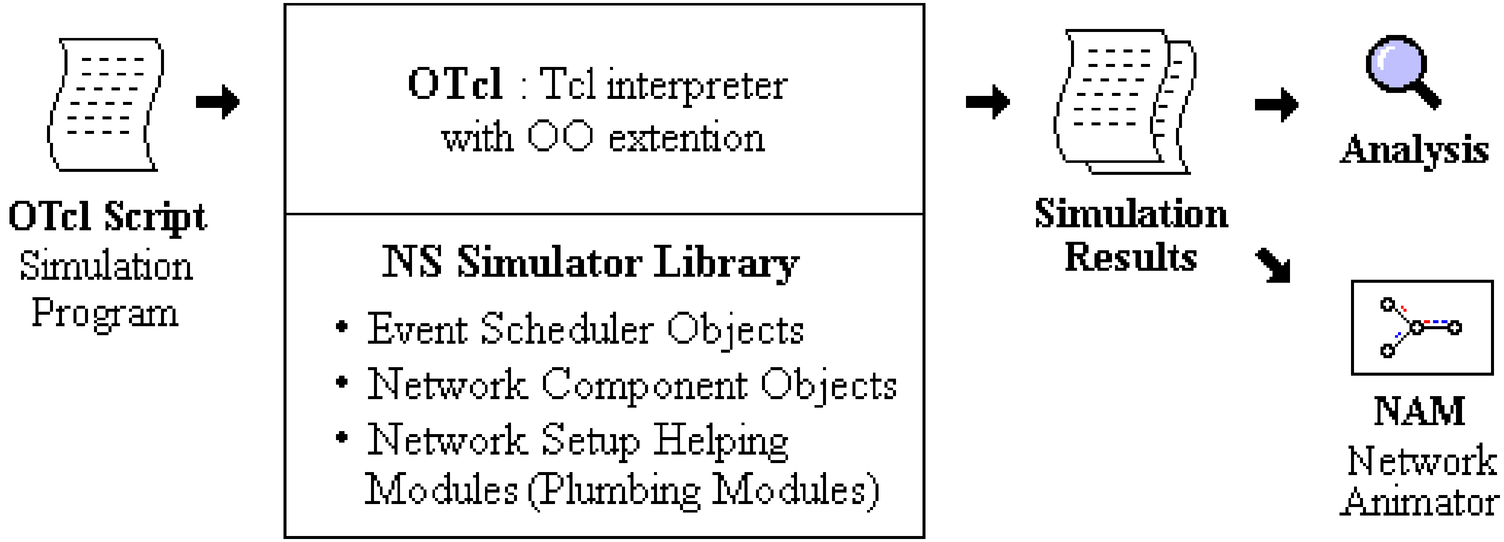


ns Architecture

- Separate data path and control path implementations.

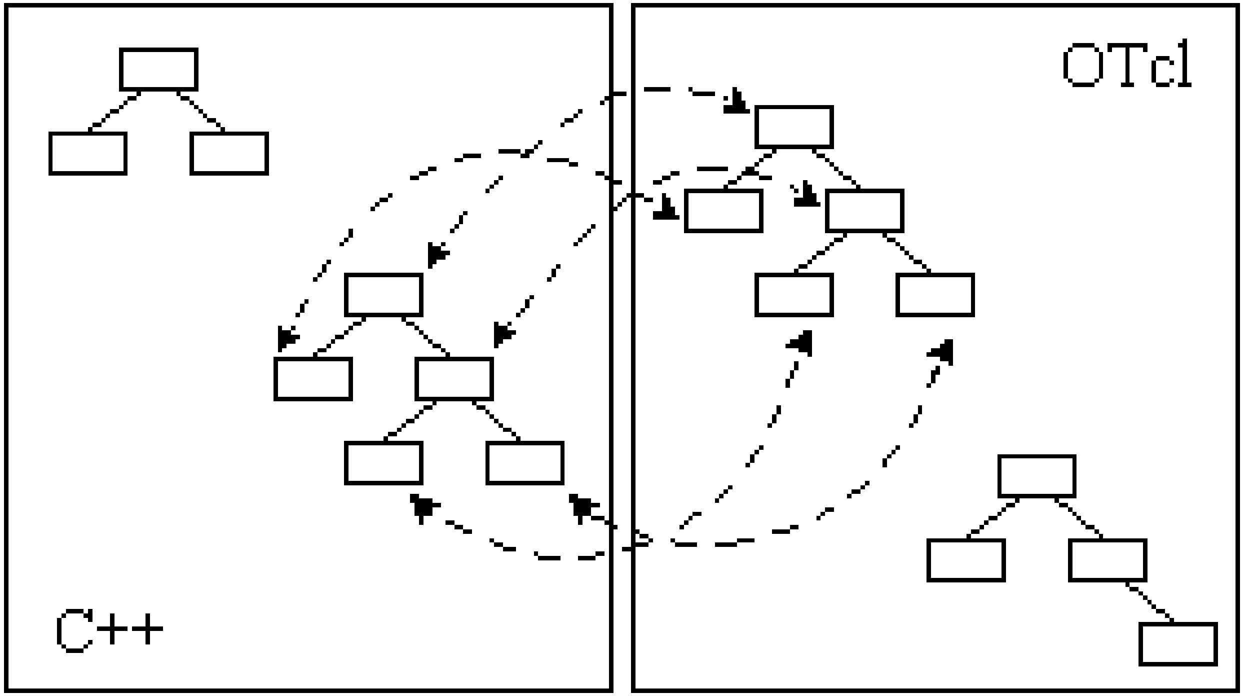
What is ns?

- Object-oriented, discrete event-driven network simulator
- Written in C++ and OTcl
- By VINT: Virtual InterNet Testbed

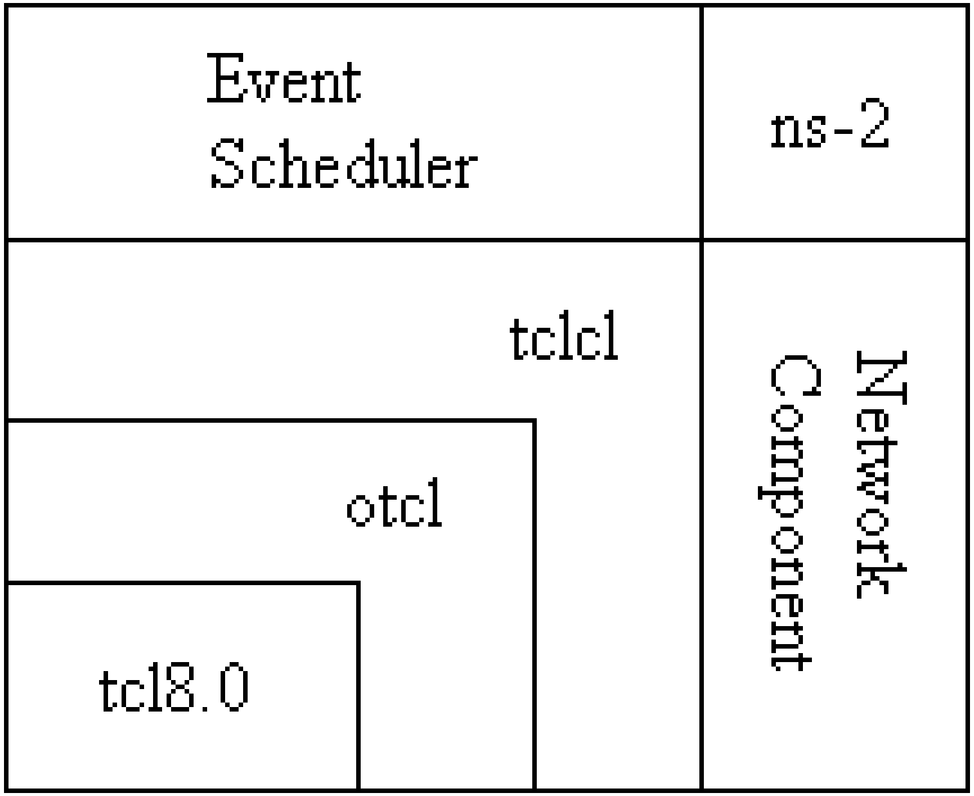


ns Architecture

- Separate data path and control path implementations.



ns Architecture



Hello World – Interactive mode

```
bash-shell$ ns
% set ns [new Simulator]
_o3
% $ns at 1 "puts \"Hello World!\""
1
% $ns at 1.5 "exit"
2
% $ns run
Hello World!
bash-shell$
```

Hello World – Batch mode

simple.tcl

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

```
bash-shell$ ns simple.tcl
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
```

Basic OTcl

```
Class Mom
Mom instproc greet {} {
   $self instvar age_
   puts "$age_ years old mom: How
   are you doing?"
}

Class Kid -superclass Mom
Kid instproc greet {} {
   $self instvar age_
   puts "$age_ years old kid:
   What's up, dude?"
}
```

```
set mom [new Mom]
$mom set age_ 45
set kid [new Kid]
$kid set age_ 15

$mom greet
$kid greet
```

```
45 year old mom say:
     How are you doing?
15 year old kid say:
     What's up, dude?
```

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
```

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)
```

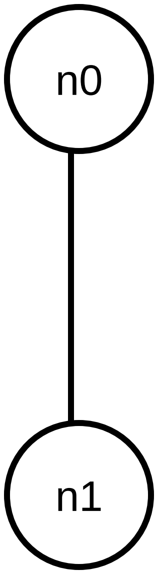
```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000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

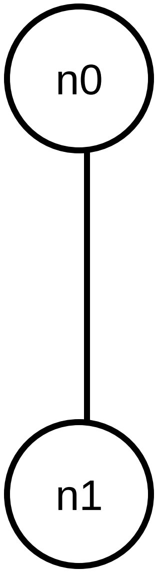
Step 3: Create network

- Two nodes, One link



Step 3: Create Network

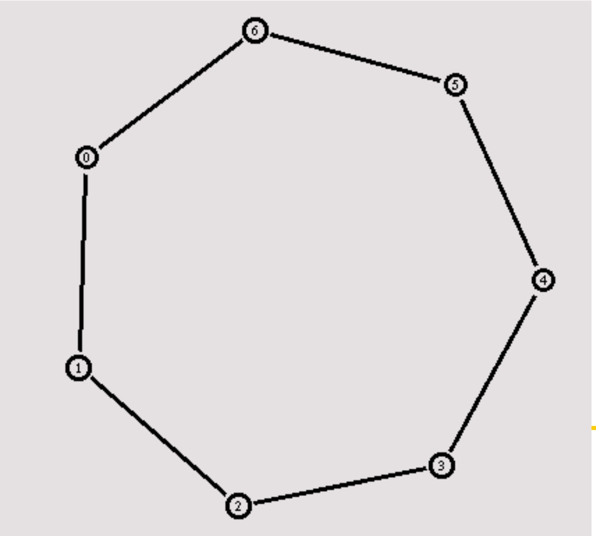
- Nodes
  - `set n0 [$ns node]`
  - `set n1 [$ns node]`



- 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

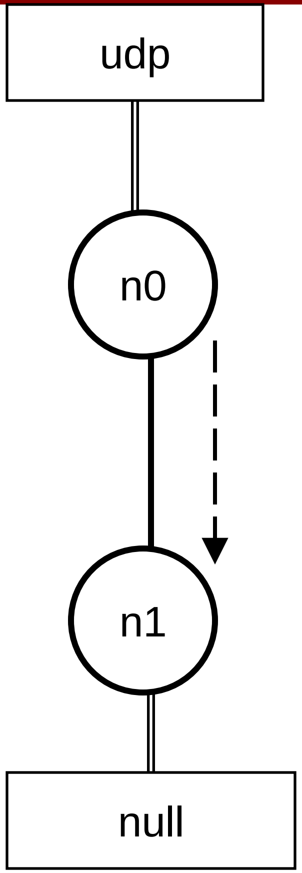
- Link failures
  - Hooks in routing module to reflect routing changes
- **`$ns rtmodel-at <time> up|down $n0 $n1`**

- For example:

```
$ns rtmodel-at 1.0 down $n0 $n1
$ns rtmodel-at 2.0 up $n0 $n1
```

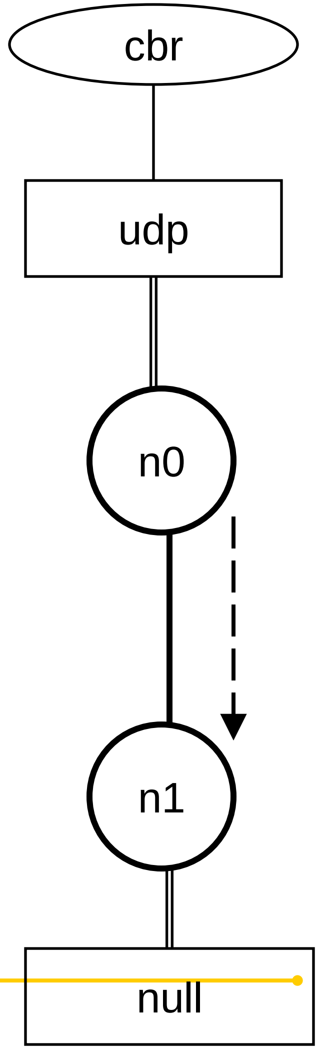
Step 5: Creating UDP connection

```
set udp [new Agent/UDP]
set null [new Agent/Null]

$ns attach-agent $n0 $udp
$ns attach-agent $n1
  $null

$ns connect $udp $null
```



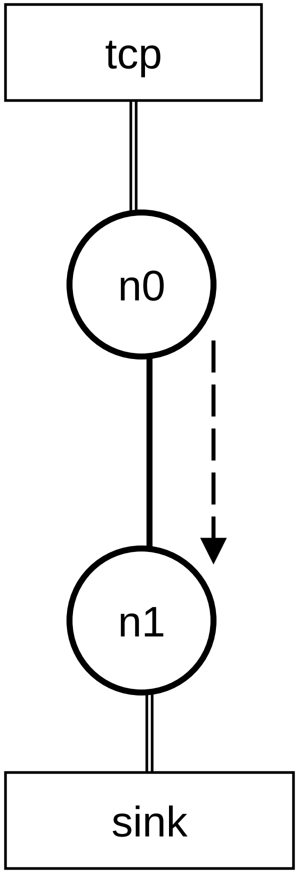
Step 6: Creating Traffic (On Top of UDP)

- CBR
  - `set cbr [new Application/Traffic/CBR]`
  - `$cbr set packetSize_ 500`
  - `$cbr set interval_ 0.005`
  - `$cbr attach-agent $udp`



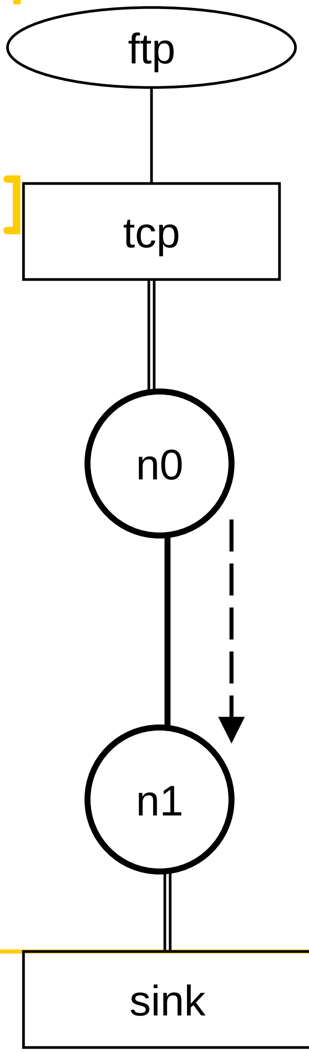
Creating TCP connection

```
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
```

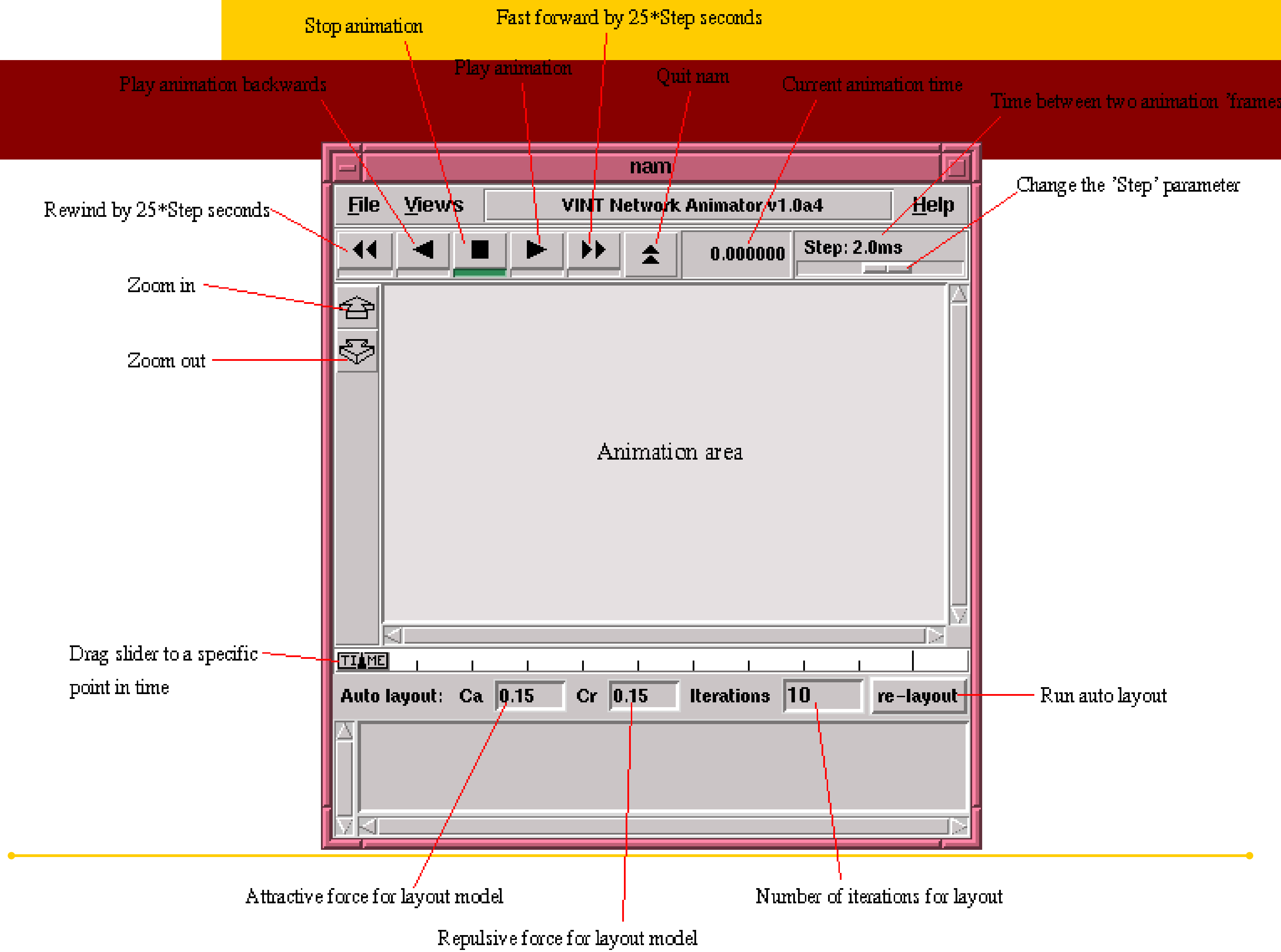
Recall: Generic Script Structure

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

Visualization Tools

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



Stop animation
Fast forward by 25*Step seconds
Play animation backwards
Play animation
Quit nam
Current animation time
Time between two animation 'frames'
nam
File
Views
VINT Network Animator v1.0a4
Help
Rewind by 25*Step seconds
0.000000
Step: 2.0ms
Change the 'Step' parameter
Zoom in
Zoom out
Animation area
Drag slider to a specific point in time
TIME
Auto layout: Ca 0.15 Cr 0.15 Iterations 10 re-layout
Run auto layout
Attractive force for layout model
Number of iterations for layout
Repulsive force for layout model

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"
```

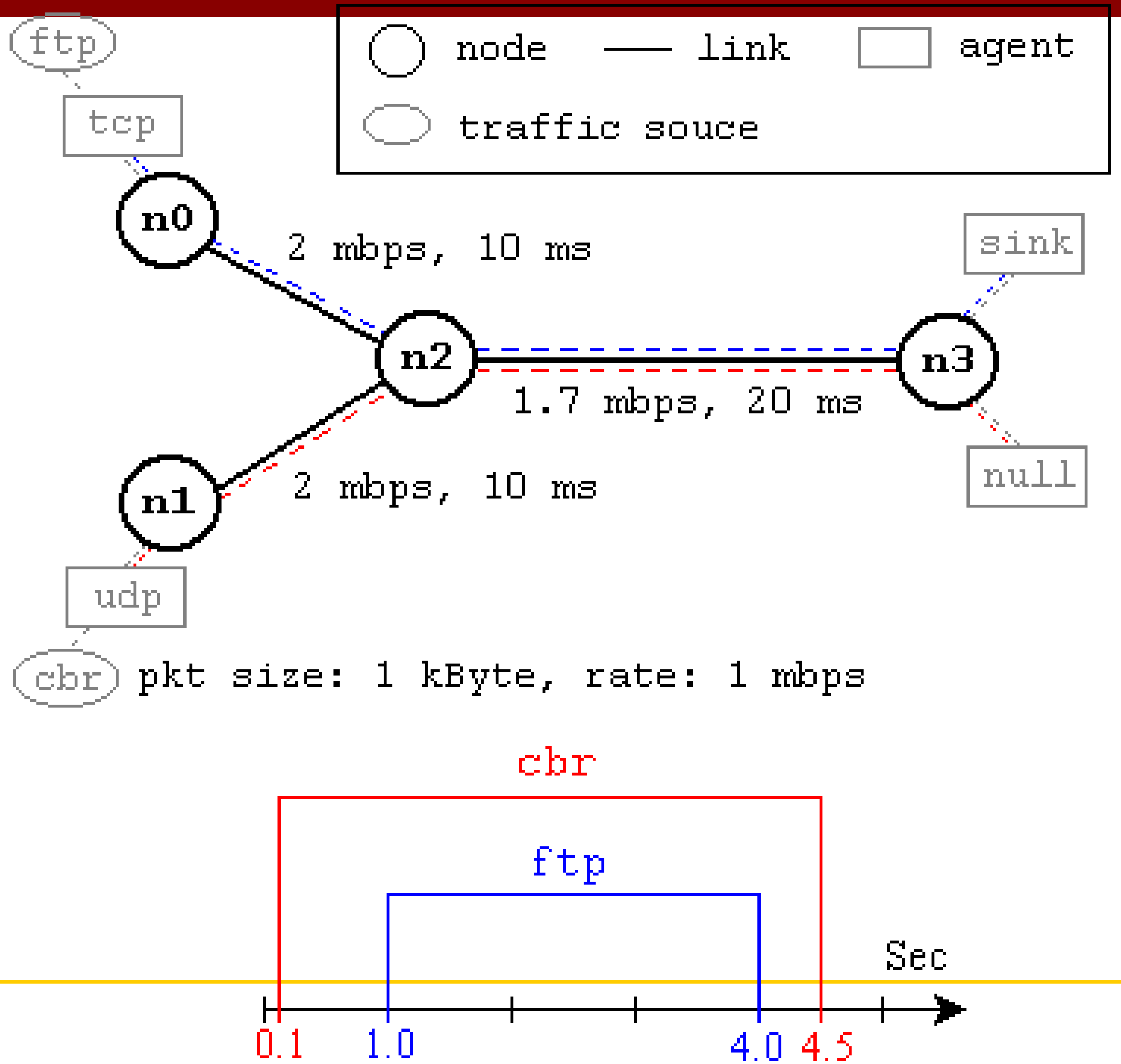
nam Interface: Topology Layout

- “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
```

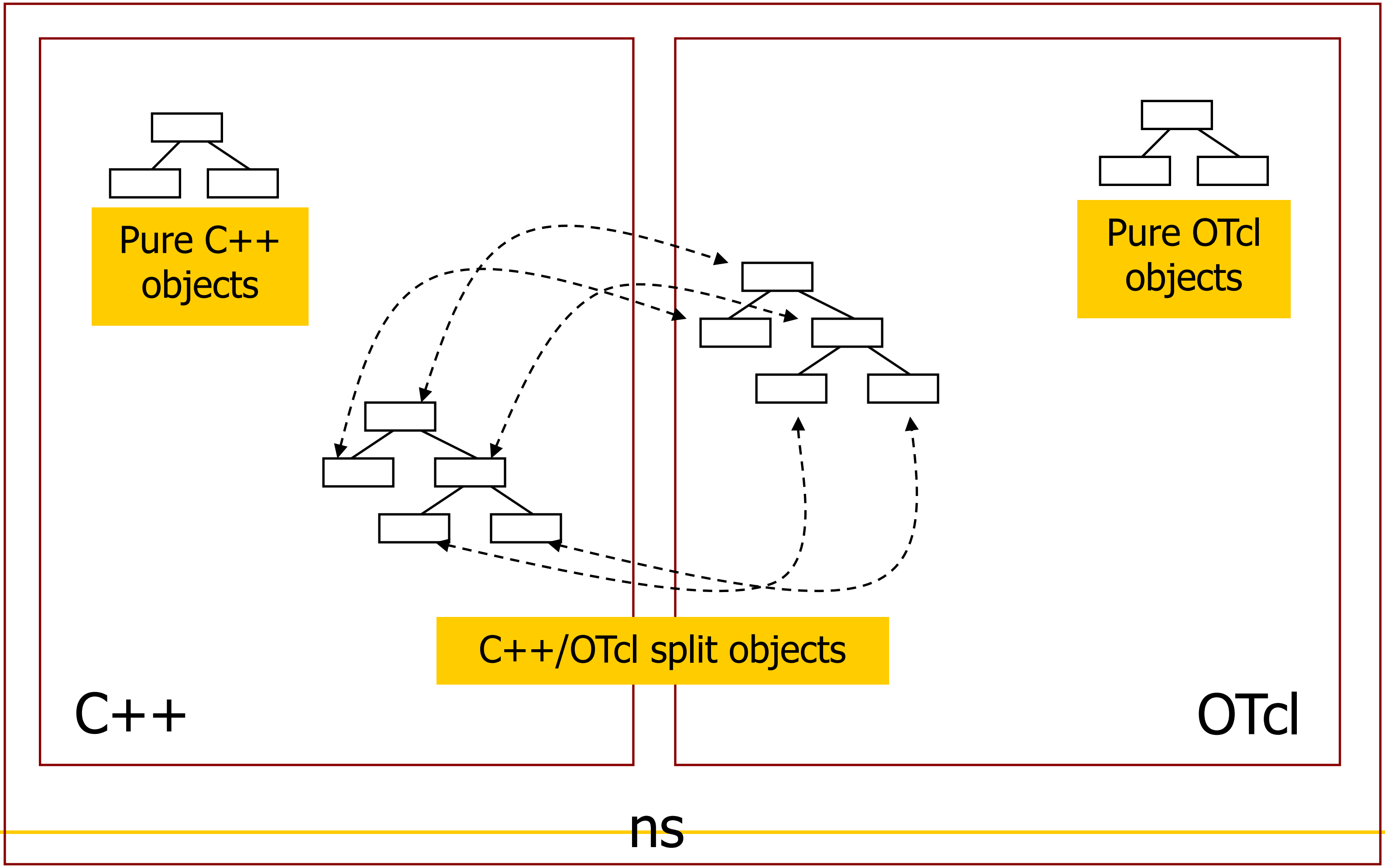
- If anything missing → automatic layout

Simulation Example



Part II: Extending ns

OTcl and C++: The Duality



Installation

- Unix variants
  - Download NS-allinone-2.27 package
  - Contains
    - TCL/TK 8.4.5
    - oTCL 1.8
    - Tclcl 1.15
    - Ns2
    - Nam -1

Installation

- After successful downloading and unzipping install allinone package , install NS by
  - install by calling ~/ns-allinone-2.27/install
- After successful installation , Validate the scripts by running ./validate in ~/ns-allinone-2.27/ns-2.27/
- Its now all set to work with NS

Code for simple topology

- Creating a Simulator Object
  - set ns [new Simulator]
- Setting up files for trace & NAM
  - set trace_nam [open out.nam w]
  - set trace_all [open all.tr w]
- Tracing files using their commands
  - $ns namtrace-all $trace_nam
  - $ns trace-all $trace_all

Code for simple topology

- Closing trace file and starting NAM
  - proc finish { } {
    - global ns trace_nam trace_all
    - $ns flush-trace
    - close $trace_nam
    - close $trace_all
    - exec nam out.nam &
    - exit 0 }

Code for simple topology

- Creating LINK & NODE topology
  - Creating NODES
    - set n1 [$ns node]
    - set n2 [$ns node]
    - set n3 [$ns node]
    - set n4 [$ns node]
    - set r1 [$ns node]
    - set r2 [$ns node]

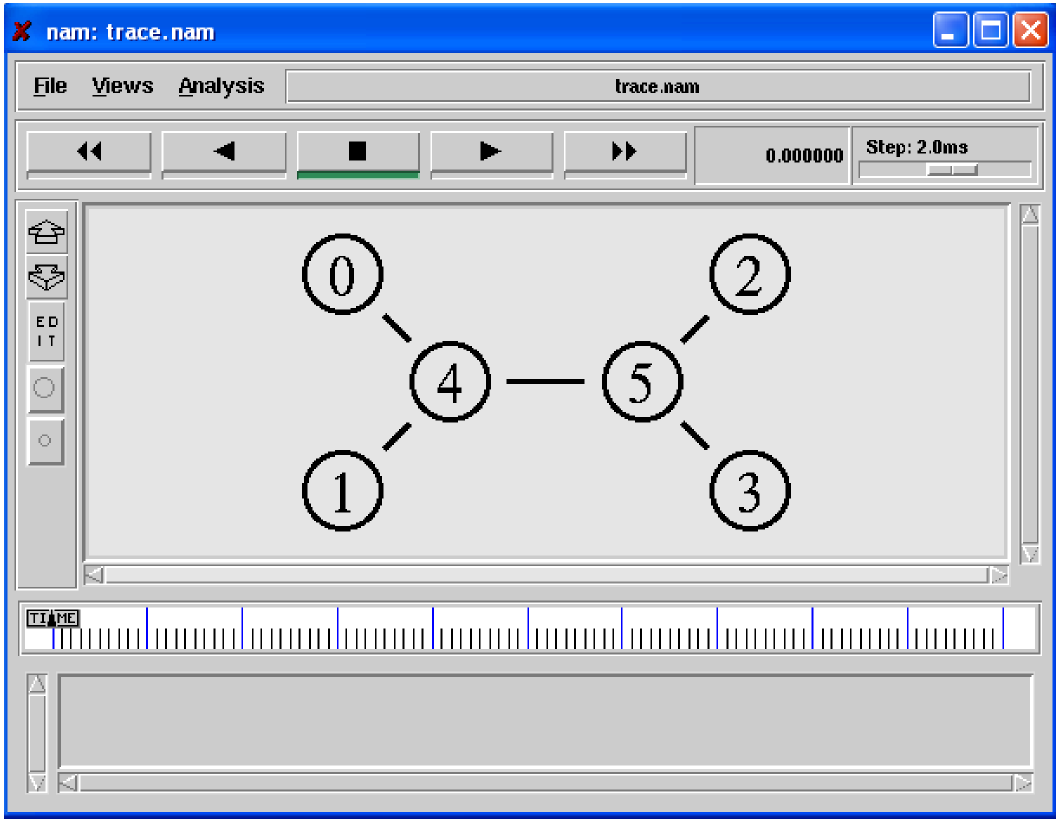
Code for simple topology

- Creating LINKS
  - $ns duplex-link $N1 $R1 2Mb 5ms DropTail
  - set DuplexLink0 [$ns link $N1 $R1]
  - $ns duplex-link $N2 $R1 2Mb 5ms DropTail
  - set DuplexLink1 [$ns link $N2 $R1]
  - $ns duplex-link $R1 $R2 1Mb 10ms DropTail
  - set DuplexLink2 [$ns link $R1 $R2]
  - $ns duplex-link $R2 $N3 2Mb 5ms DropTail
  - set DuplexLink3 [$ns link $R2 $N3]
  - $ns duplex-link $R2 $N4 2Mb 5ms DropTail
  - set DuplexLink4 [$ns link $R2 $N4]

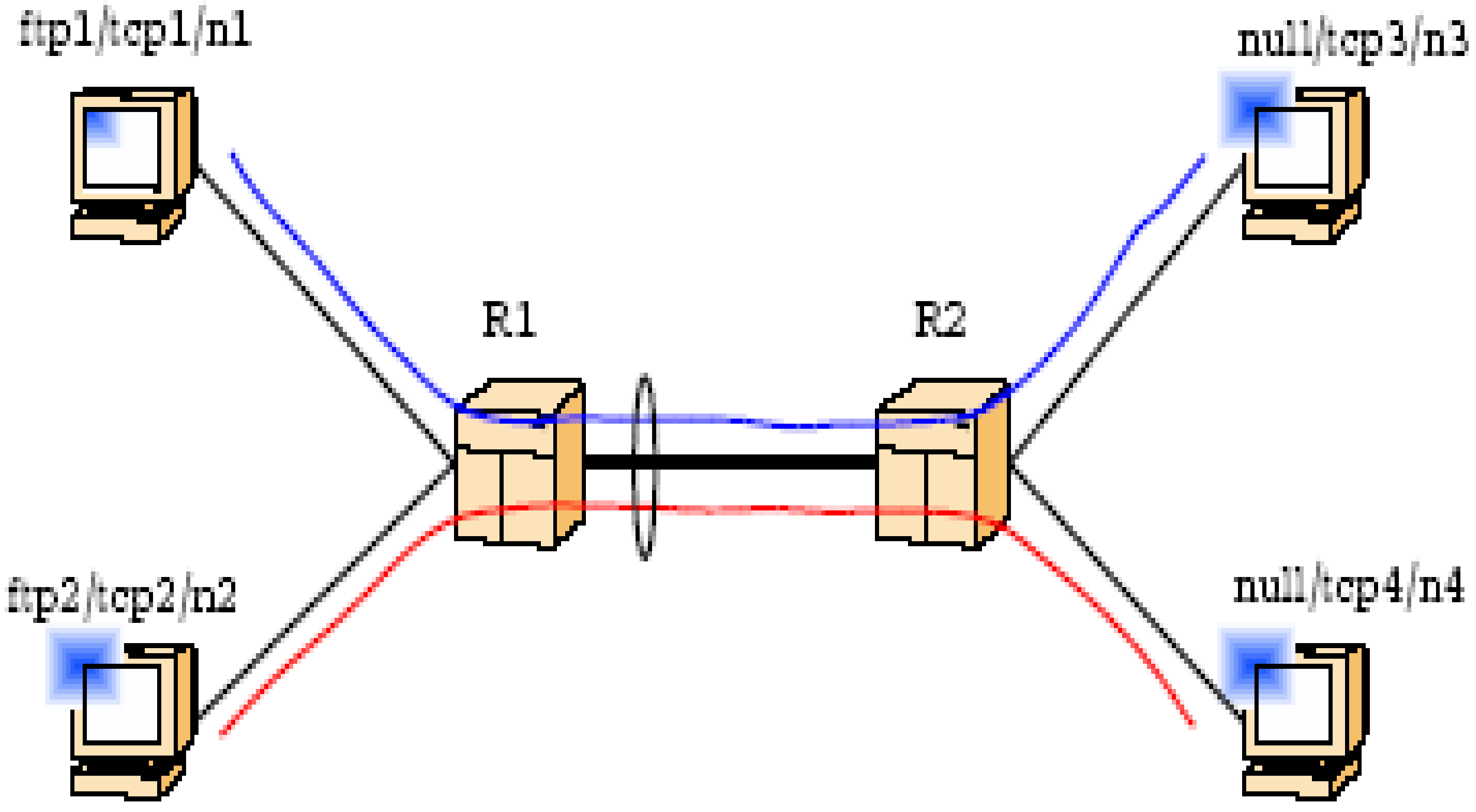
Code for simple topology

- Orientation of links
  - $ns duplex-link-op $N1 $R1 orient right-down
  - $ns duplex-link-op $N2 $R1 orient right-up
  - $ns duplex-link-op $R1 $R2 orient right
  - $ns duplex-link-op $R2 $N3 orient right-up
  - $ns duplex-link-op $R2 $N4 orient right-down

Final topology Generated



Traffic topology aimed at



Generating Traffic

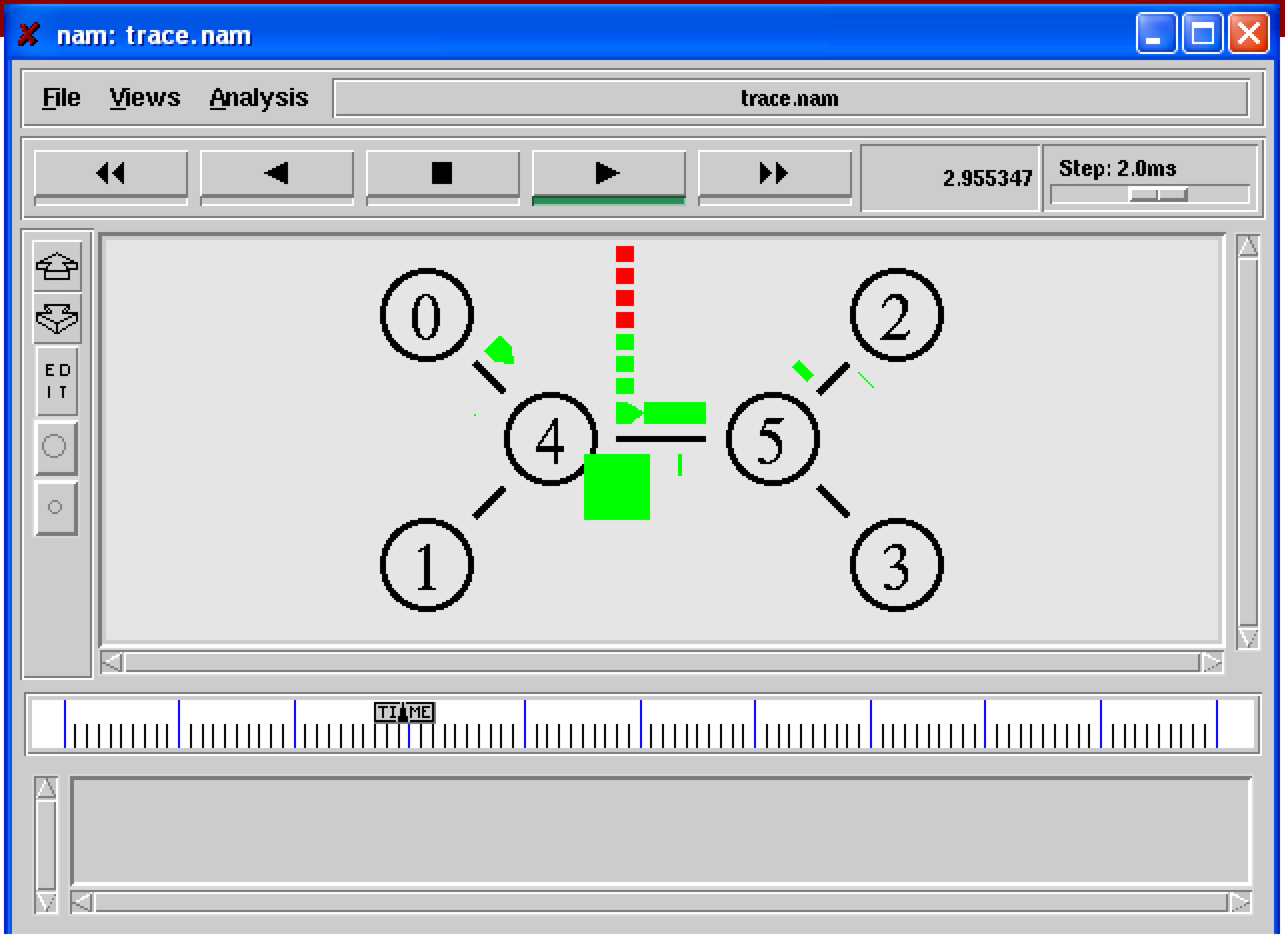
- Attaching AGENT TCP to NODE 1
  - set TCP1 [new Agent/TCP]
  - $ns attach-agent $N1 $TCP1
- Attaching AGENT TCP to NODE 2
  - set TCP2 [new Agent/TCP]
  - $ns attach-agent $N2 $TCP2
- Attaching AGENT TCP to NODE 3
  - set TCP3 [new Agent/TCPSink]
  - $ns attach-agent $N2 $TCP3
- Attaching AGENT TCP to NODE 4
  - set TCP4 [new Agent/TCPSink]
  - $ns attach-agent $N2 $TCP4

Generating Traffic

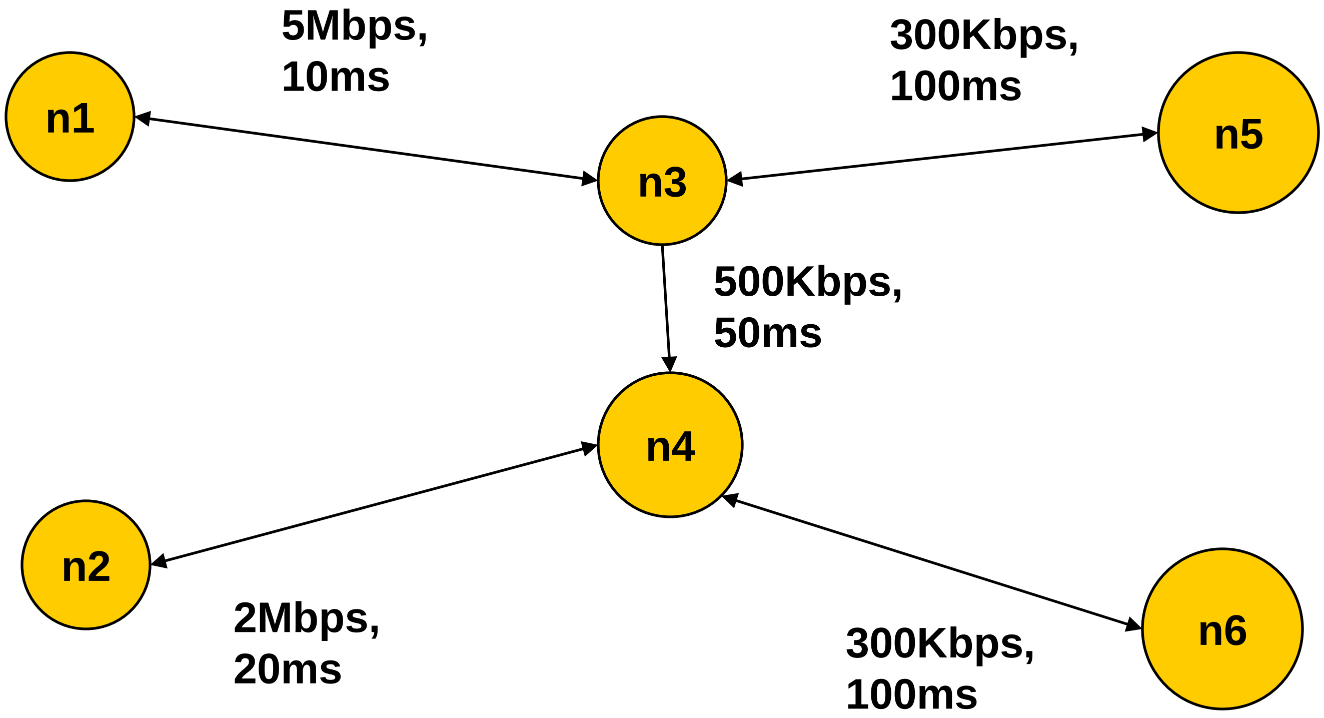
- Attaching Application (FTP)
  - set FTP0 [new Application/FTP]
  - set FTP1 [new Application/FTP]
  - $FTP0 attach-agent $TCP0
  - $FTP1 attach-agent $TCP1

Setting simulation times

- $ns at 0.5 "$FTP0 start"
- $ns at 0.5 "$FTP1 start"
- $ns at 10.0 "$FTP0 stop"
- $ns at 10.0 "$FTP1 stop“
- $ns at 10.0 “finish”
- Making NS run
  - $ns run


nam: trace.nam
File
Views
Analysis
trace.nam
2.955347
Step: 2.0ms
EDIT
0
1
2
3
4
5
TIME

Creating Topologies



Creating Topologies

- Nodes
  - Set properties like queue length, location
  - Protocols, routing algorithms
- Links
  - Set types of link – Simplex, duplex, wireless, satellite
  - Set bandwidth, latency etc.
- Done through tcl Scripts

Observing Network Behavior

- Observe behavior by tracing “events”
  - Eg. packet received, packet drop etc.

Src Dst IP Address, Port

```
+ 0.1 1 2 cbr 1000 ------- 2 1.0 5.0 0 0
- 0.1 1 2 cbr 1000 ------- 2 1.0 5.0 0 0
r 0.114 1 2 cbr 1000 ------- 2 1.0 5.0 0 0
+ 0.114 2 3 cbr 1000 ------- 2 1.0 5.0 0 0
- 0.114 2 3 cbr 1000 ------- 2 1.0 5.0 0 0
r 0.240667 2 3 cbr 1000 ------- 2 1.0 5.0 0 0
```

time

Observing Network Behavior

- NAM:
  - Network Animator
  - A visual aid showing how packets flow along the network

How Do I use it?

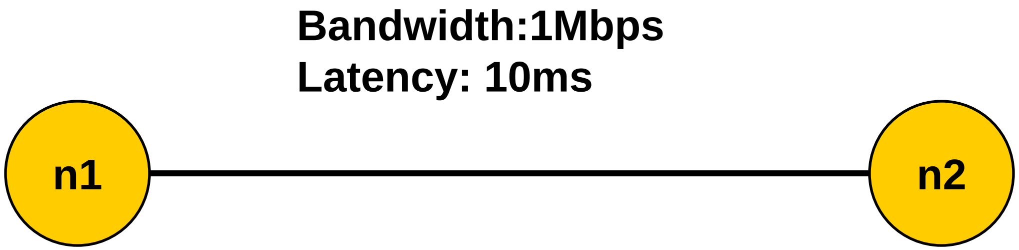
- Creating a Simple Topology
- Getting Traces
- Using NAM

Basics of using NS2

- Define Network topology, load, output files in Tcl Script
- To run,

  $ ns simple_network.tcl
- Internally, NS2 instantiates C++ classes based on the tcl scripts
- Output is in form of trace files

A simple Example – Creating the topology



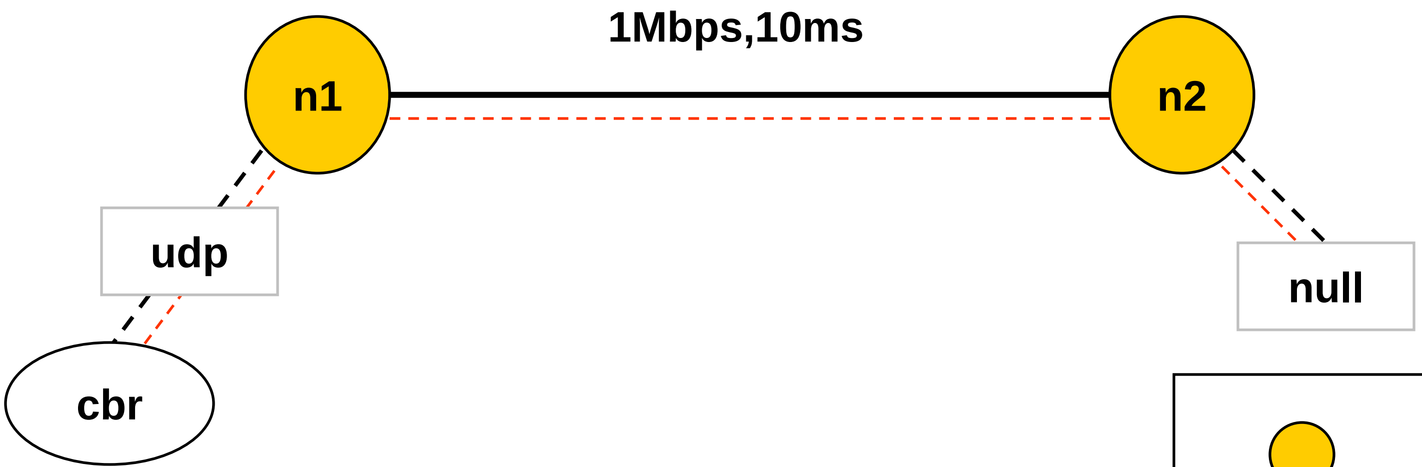
Creating the topology

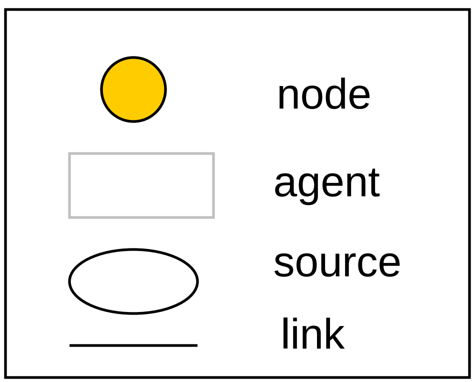
```
#create a new simulator object
set ns [new Simulator]

#open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace

    #close the trace file
    close $nf

    #execute nam on the trace file
    exec nam out.nam &

    exit 0
}
```

Creating the topology (Contd)

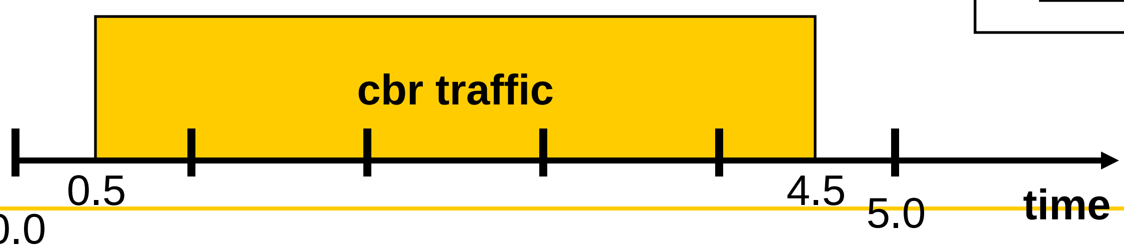
```
#create two nodes
set n0 [$ns node]
set n1 [$ns node]

#create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

Adding traffic



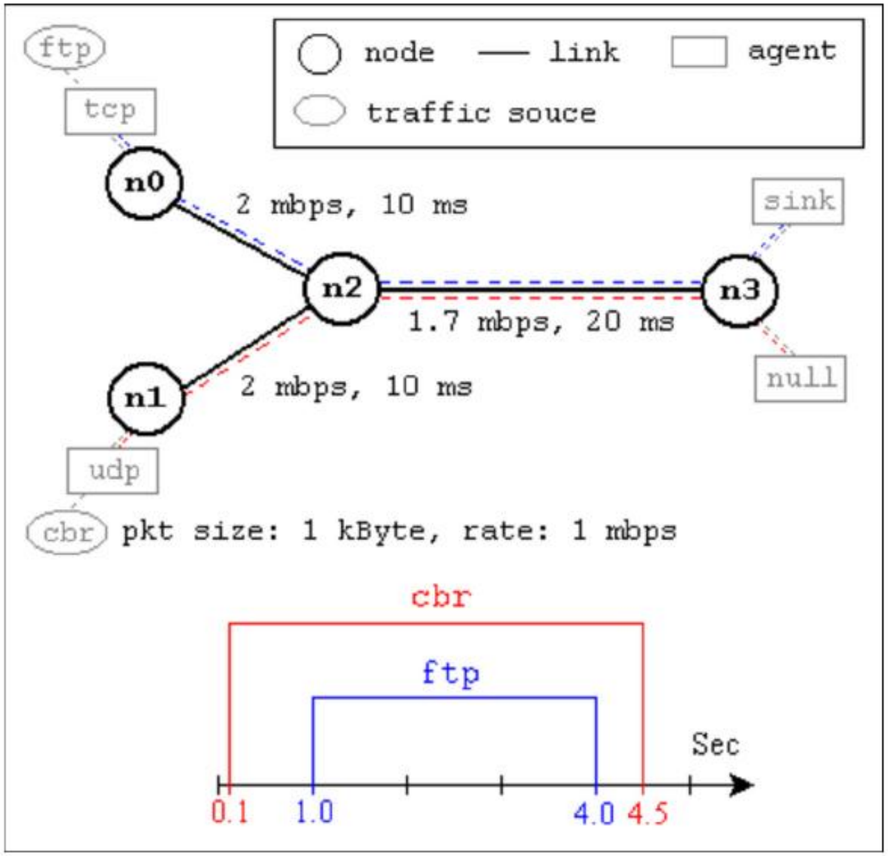


Packet Size: 500 bytes
rate: 800Kbps



```
#create a udp agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

#Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#create a Null agent(a traffic sink) and attach it to node n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

#Connect the traffic source to the sink
$ns connect $udp0 $null0

#Schedule events for CBR traffic
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

#call the finish procedure after 5 secs of simulated time
$ns at 5.0 "finish"

#run the simulation
$ns run
```

A second Scenario * (from NS by Example)



Taken from NS by Example
by [Jae Chung]
and
[Mark Claypool]

A second Example (From NS by Example)

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
```

A Second Scenario (Contd.)

```
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
```

A Second Scenario (Contd.)

```
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

A Second Scenario (Contd.)

```
#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
```

To create agents or traffic sources, we need to know the class names these objects (Agent/TCP, Agent/TCPSink, Application/FTP and so on).
This information can be found in the NS documentation.
But one shortcut is to look at the "ns-2/tcl/libs/ns-default.tcl" file.

A Second Scenario (Contd.)

```
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
```

A Second Scenario (Contd.)

```
#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```