Network Simulator NS2 Tutorial

Network Simulation

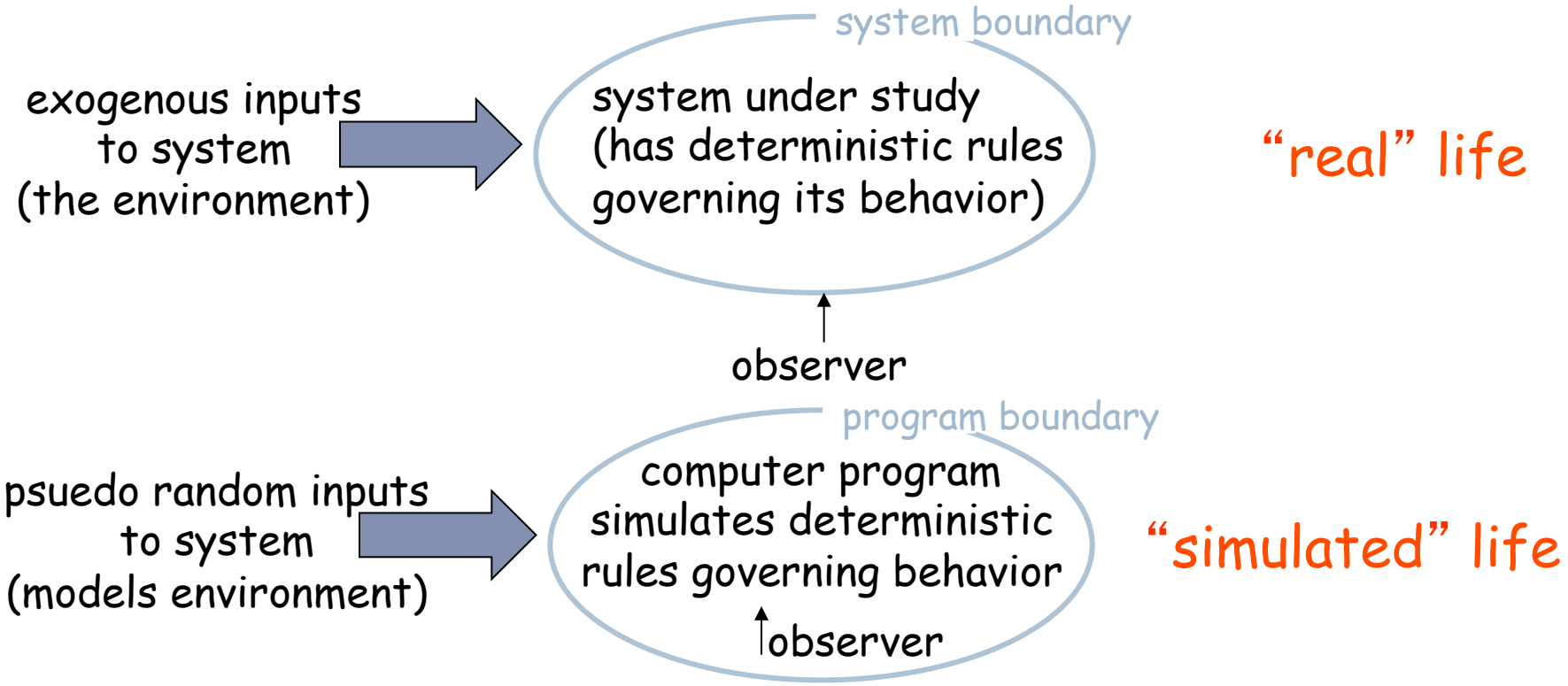
Motivation:

- Learn fundamentals of evaluating network performance via simulation

Overview:

- fundamentals of discrete event simulation
- ns-2 simulation

What is Simulation?



Why Simulation?

- Real-system not *available, is complex/costly or dangerous* (eg: space simulations, flight simulations)
- Quickly evaluate design *alternatives* (eg: different system configurations)
- Evaluate *complex functions* for which closed form formulas or numerical techniques not available

Simulation: Advantages/Drawbacks

- Advantages:
  - Sometimes cheaper
  - Find bugs (in design) in advance
  - Generality: over analytic/numerical techniques
  - Detail: can simulate system details at arbitrary level
- Drawbacks:
  - Caution: does model reflect reality
  - Large scale systems: lots of resources to simulate (especially accurately simulate)
  - May be slow (computationally expensive – 1 min real time could be hours of simulated time)
  - Art: determining right level of model complexity
  - Statistical uncertainty in results

The Evaluation Spectrum

- Numerical models
- Simulation
- Emulation
- Prototype
- Operational system

Programming a simulation

What's in a simulation program?

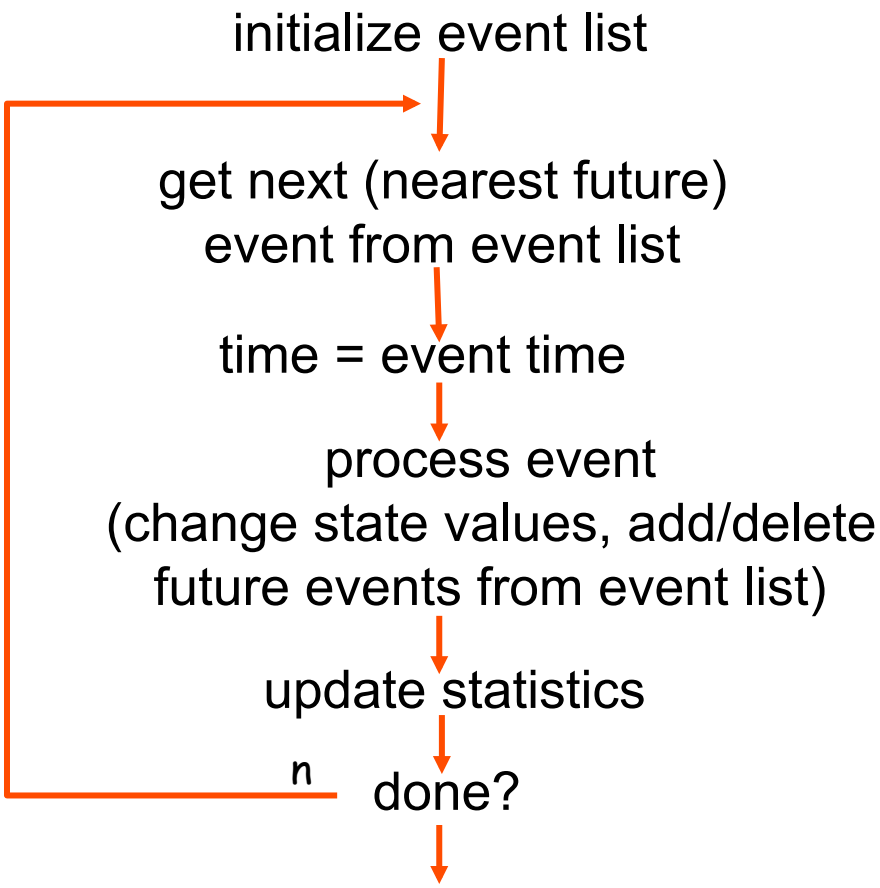
- *Simulated time:* internal (to simulation program) variable that keeps track of simulated time
- *System "state":* variables maintained by simulation program define system "state"
  - E.g., may track number (possibly order) of packets in queue, current value of retransmission timer
- *Events:* points in time when system changes state
  - Each event has associate *event time*
    - E.g., arrival of packet to queue, departure from queue
    - Precisely at these points in time that simulation must take action (change state and may cause new future events)
  - Model for time between events (probabilistic) caused by external environment

Simulator Structure

- Simulation program maintains and updates list of future events: event list

- Need:
  - Well defined set of events
  - For each event: simulated system action, updating of event list

Simulator Block Diagram



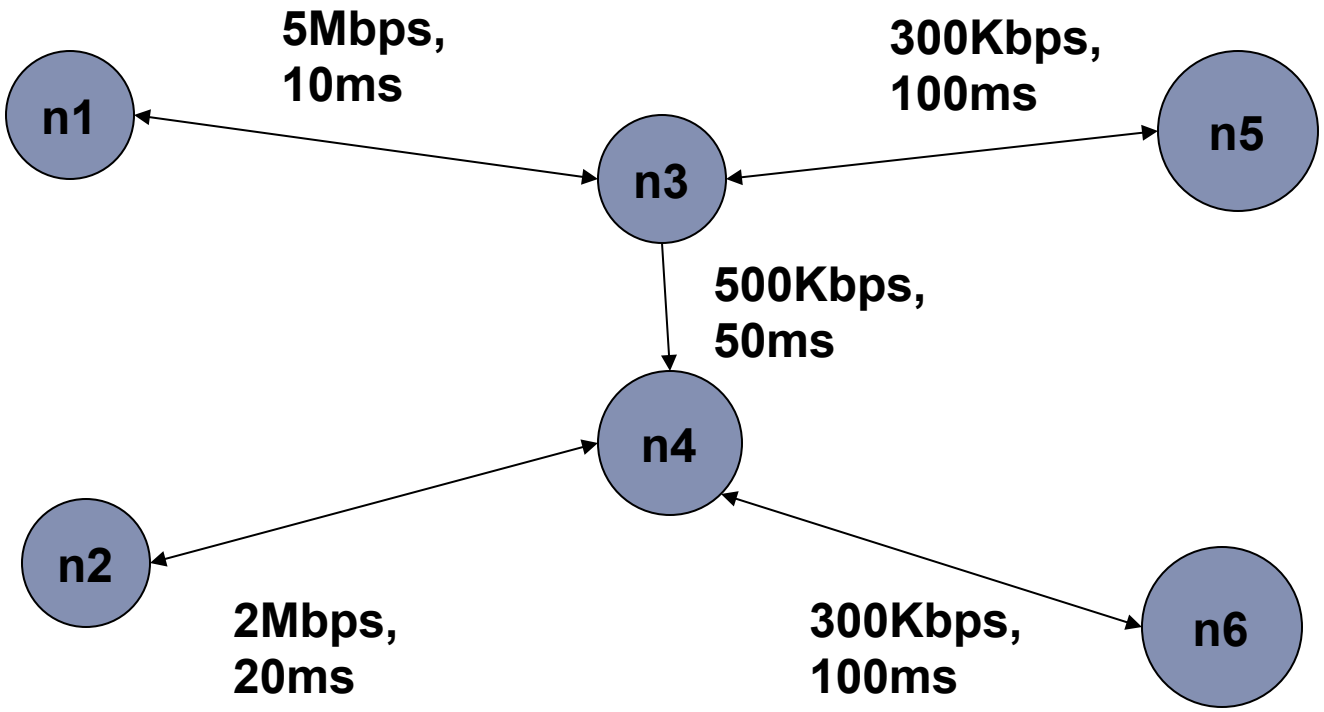
NS2 Outline

- What is it?
- How do I get it?
- How do I use it?
- How do I add to it?
- Documentation

What is NS2?

- Network Simulator
- A package of tools that simulates behavior of networks
  - Create Network Topologies
  - Log events that happen under any load
  - Analyze events to understand the network behavior

Creating Topologies

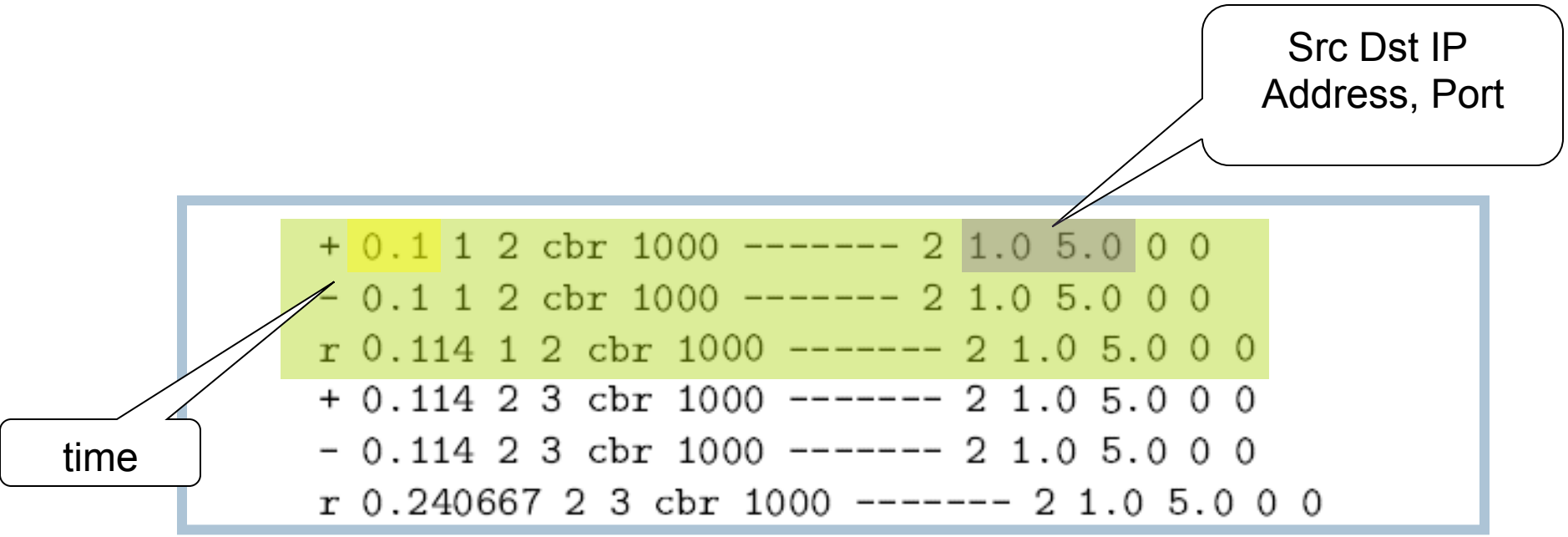


Creating Topologies

- 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.



Observing Network Behavior

- NAM:
  - Network Animator
  - A visual aid showing how packets flow along the network
- We' ll see a demo..

How Do I get NS2?

- NS already Installed for us at testbed
  - /usr/local/bin/ns2
- NAM already installed at
  - /usr/local/bin/nam

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

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.

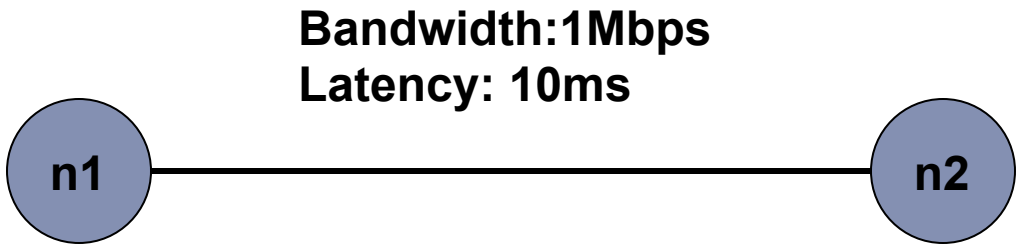
Basic otcl

```
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
```

=> can easily make variations of existing things (TCP, TCP/Reno)

A simple Example – Creating the topology



Creating the topology

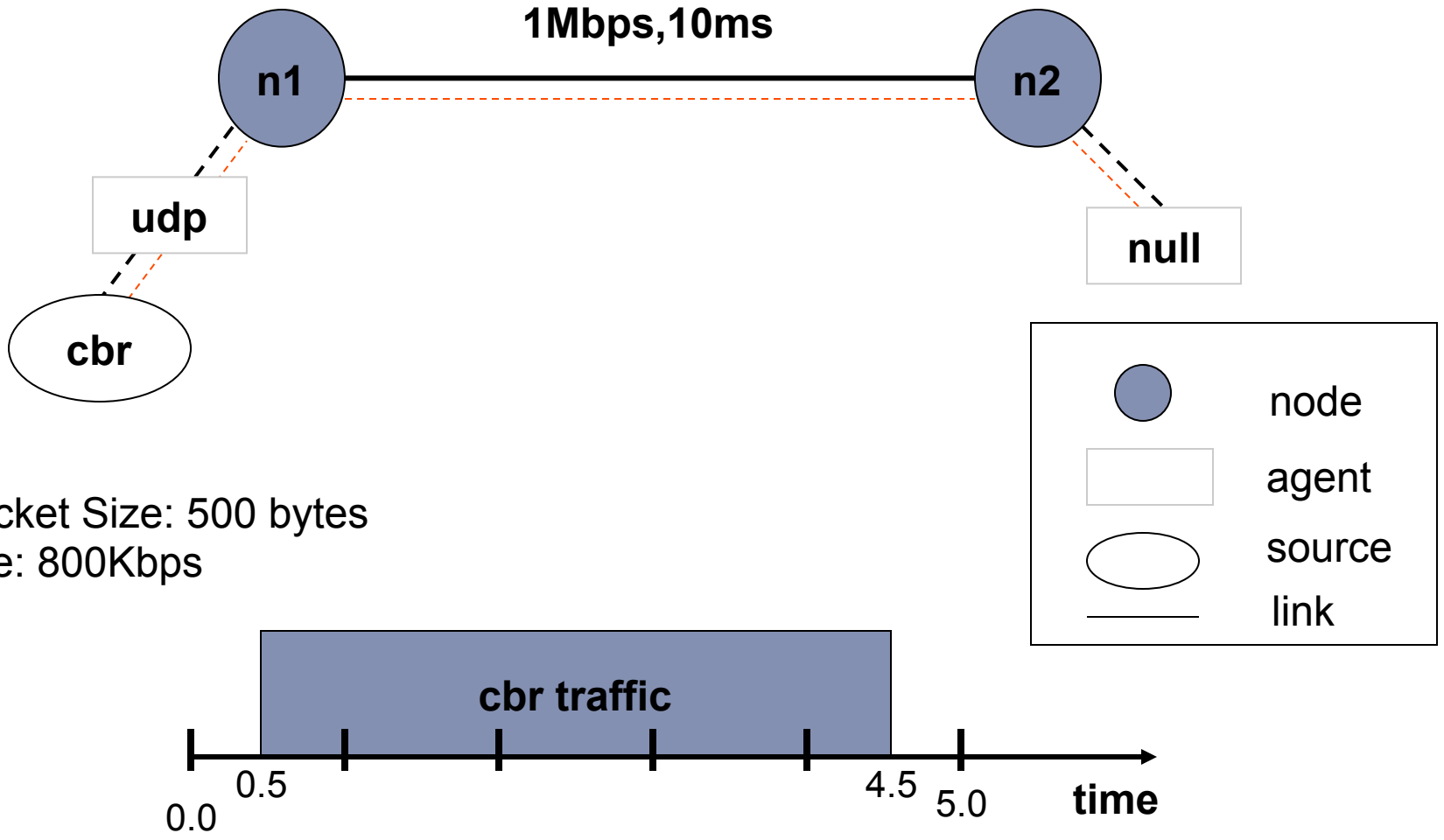
```
#create a new simulator object
set ns [new Simulator]

#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
```

Demo

Adding traffic



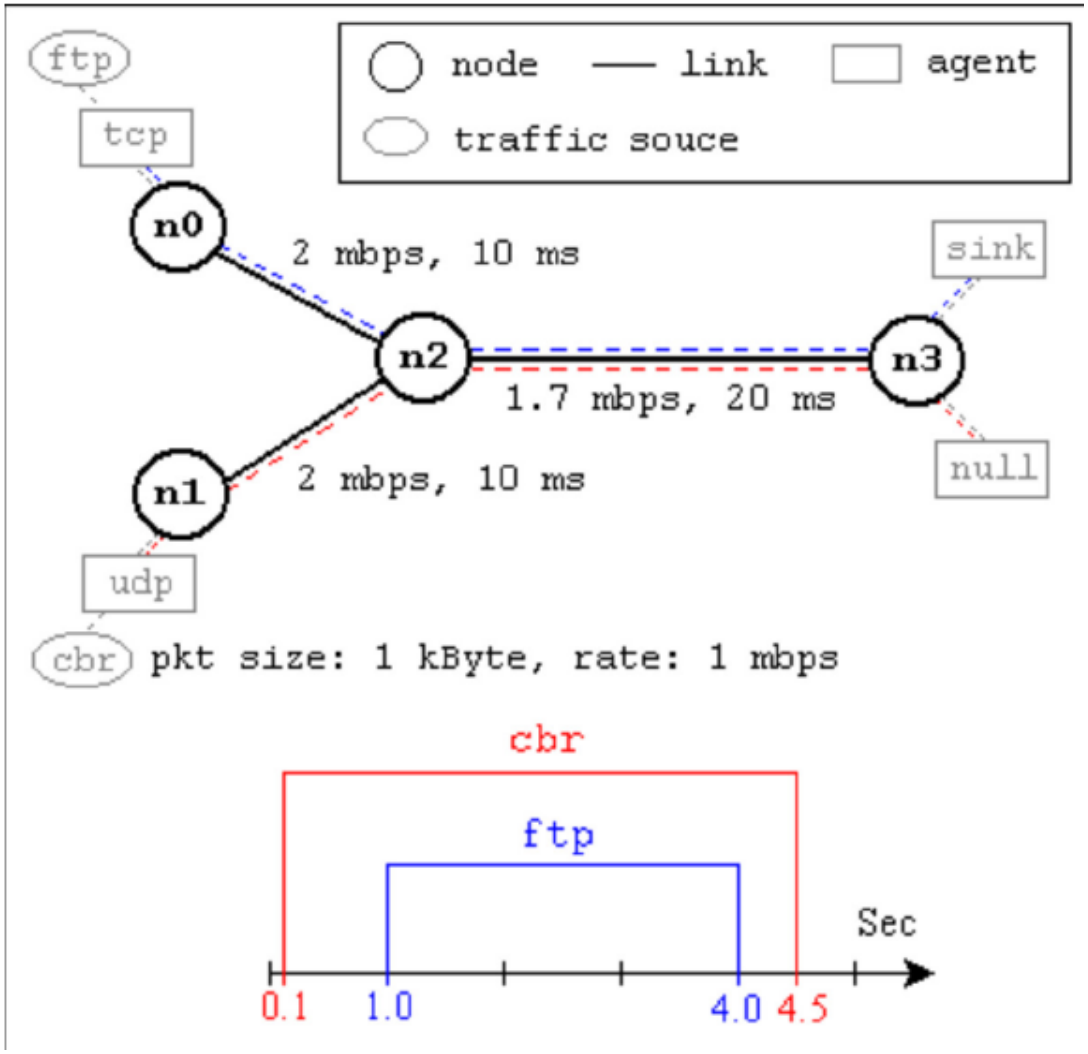
Putting it together..

```
#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
```

Demo

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
```

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.

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

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
```

Demo

Trace Analysis -- ns-simple-trace.tcl

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→nam Interface

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

nam Interface: Color

- Color mapping

```
$ns color 40 red
$ns color 41 blue
$ns color 42 chocolate
```

- Color ↔ flow id association

```
$tcp0 set fid_ 40  ;# red packets
$tcp1 set fid_ 41  ;# 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: Misc

- Annotation
  - Add textual explanation to your simulation

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

- Set animation rate

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

Outline

- What is it?
- How do I get it?
- How do I use it?
- Documentation

Documentation – NS2 Documentation

- NS2 Manual
  - Information about Otcl interpreter, C++ class hierarchy, parameters for various protocols
  - http://www.isi.edu/nsnam/ns/doc/index.html
  - Very detailed, useful when looking for something specific, like:
    - What are the shadowing models available for wireless? How do I select them?
    - How do I make my routing strategy to be Distance Vector routing?

Documentation – NS2 documentation

- NS2 Tutorial by Marc Greis
  - http://www.isi.edu/nsnam/ns/tutorial/index.html
  - Good starting point for understanding the overall structure of NS2
  - Examples:
    - What is the relation between c++ classes and Otcl classes?
    - basic info on instantiating NS2 instance, tcl scripting

Documentation – NS2 Documentation

- NS2 for beginners
  - http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf
  - More detailed than Marc Greis' Tutorial
  - More info on getting it up and running – rather than internals
  - Examples:
    - What does each line of a tcl script do?
    - Most common examples of trace formats that are useful

Documentation – Tcl Documentation

- Tcl Tutorial
  - http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html
- Tcl Manual
  - All commands and their explanation
  - http://www.tcl.tk/man/tcl8.6/TclCmd/contents.htm