Outline

- Goals
- Discrete Event Simulation
- Basic ns-2
- Examples
What is ns-2?

- Discrete event simulator
- Models network protocols
  - Wired, wireless, satellite
  - TCP, UDP, multicast, unicast
  - Web, Telnet, FTP
  - Ad-hoc routing, sensor networks
  - Infrastructure: stats, tracing, error models, etc.
- Multiple levels of detail in one simulator

Why simulate?

- To examine protocol in controlled environment
- Repeatable experiments
- Alternatives:
  - Experimentation: operational details, but limited scale, limited flexibility
  - Analysis: can provide deeper understanding, but ignores implementation details
ns-2 components

- **ns**: the Network Simulation engine
  - executes tcl scripts containing simulation setup and events
- **nam**: the Network AniMator
  - visualize ns output

Discrete Event Simulation

- model world as events
  - maintain queue of events, ordered by time
  - maintain [virtual time]
  - repeat:
    - extract event at head, set [virtual time]=event’s time
    - process it
    - if processing generates another event, add it to queue
- each event takes predefined amount of virtual time, arbitrary amount of real time
  - having a slow CPU makes simulation run slower (in real time), but doesn’t change result
Discrete Event Simulation

OTcl and C++: The Duality

oTcl overview

- programming language used for setting up simulation environment
  - object oriented
  - interpreted (slow)
- Used for
  - Setting up topology
  - Placing agents
  - Injecting events
  - Configuring tracing

Examples:
- **variables**
  - set x 10
  - puts “x is $x”
- **expressions**
  - set y [pow x 2]
  - set y [expr x+x*3]
- **control**
  - if ($x>0) { return $x } else { return [expr -$x] } 
  - while ($x >0) { puts $x
    - set x [eval x+1] }
Basic ns-2

1. Create the simulator object
   set ns [new Simulator]
2. Set up the network topology
   set <var> [\$ns node]
3. Define traffic patterns
   $\ns \text{<link-type>} \text{<node1>} \text{<node2>} \text{<bandwidth>} \text{<delay>} \text{<queuetype>}
4. Define agents
   set <var> [\new Agent/<agent type>]
   $\ns \text{attach-agent} \text{<node variable>} \text{<agent variable>}
   $\ns \text{connect} \text{<agent variable>} \text{<agent variable>}
5. Define the trace files and monitors
   $\ns \text{namtrace-all } \text{namtracefile}
6. Schedule simulation events
   $\ns \text{at} \text{<time>} \text{<event>}
7. Run the simulator
   $\ns \text{run}
4. Define Agents

`# Create a TCP agent and attach it to node n0`
`sset tcp0 [new Agent/TCP]`
`$ns attach-agent $n0 $tcp0`

`# Create a TCP sink agent and attach it to node n2`
`sset sink [new Agent/TCPSink]`
`$ns attach-agent $n2 $sink`

`# Connect both agents`
`s$ns connect $tcp0 $sink`

`# create an FTP source`
`sset ftp [new Application/FTP]`
`$ftp setmaxpkts_ 1000`
`$ftp attach-agent $tcp0`

```
ftp
```
```
tcp
```
```
n0 n1 n2
cp-sink
```

5. Define Trace Files and Monitors

`# open the nam trace file`
`sset nam_trace_fd [open tcp_tahoe.nam w]`
`s$ns namtrace-all $nam_trace_fd`

`# define a 'finish' procedure`
`proc finish {} {
    global ns nam_trace_fd trace_fd
    # close the nam trace file`
`s$ns flush-trace`
    close $nam_trace_fd`
    # execute nam on the trace file`
    `exit 0`
`}`

```
ftp
```
```
tcp
```
```
n0 n1 n2
cp-sink
```

---

Basic ns-2

Basic ns-2
6. Schedule Simulation Events
$ns at 0.0 "$ftp start"
$ns at 10.0 "$ftp stop"
$ns at 10.1 "finish"

7. Run the simulation
#Run the simulation
$ns run

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

# Create three nodes
set n0 [new Node]
set n1 [new Node]
set n2 [new Node]

# Create link between the nodes
$ns duplex-link $n0 $n1 4Mb 10ms DropTail
$ns duplex-link $n2 $n1 1Mb 10ms DropTail
$ns queue-limit $n1 $n2 10

# Create a TCP agent and attach it to node n0
set tcp0 [new TCP]
$ns attach-agent $n0 $tcp0

# Create a TCP sink agent and attach it to node n2
set sink [new TCPSink]
$ns attach-agent $n2 $sink

# Connect both agents
$ns connect $tcp0 $sink

# Create an FTP source
set ftp [new Application/FTP]
$ftp set maxpkts_ 1000
$ftp attach-agent $tcp0

# Inject starting events
$ns at 0.0 "$ftp start"
$ns at 10.0 "$ftp stop"
$ns at 10.1 "finish"

# Run the simulation
$ns run
```
nam visualization demo

- Now I show an instance of nam running the example on the previous slide...

We just saw this...  Now I will show you this...

tcl script (specification of experiment) \(\rightarrow\) ns-2 \(\rightarrow\) trace file (output) \(\rightarrow\) nam

Analysis

- Other commands for configuring nam not discussed here
Questions?

Project 2 - Introduction
Overview

- Learning goals
- Project structure
- Submission details

Goals

- Investigating the behavior of TCP
  - Effects of different window sizes on TCP dataflow
  - Appreciate the different variants of TCP
  - TCP with non-TCP flows

- Queueing
  - Droptail vs. RED
Project Structure

- Project is divided into two main tasks
  - Task 1:
    - 4 questions
    - 10 points
  - Task 2:
    - 8 questions
    - 30 points

Task 1

- Initial network configuration

- 2 TCP flows
  - Observe what happens when the window sizes change
Results

- Trace file: typically has the format (time, val)
- Nam file format:
  - For example: event r –t 10 –s 1 –d 2 –p “tcp”
- How do we access events?
  - awk '$1== "r" && $7==3 {a += $11} END {print a}'

Task 1: modifications

- You will need to modify the script to create the following topology:

  ![Topology Diagram]

- A third TCP flow is added
Task 2

- You are given several “black-box” (unknown) components:
  - tcp_A; sink_A
  - tcp_B; sink_B
  - tcp_C; sink_C
  - Queue_A
  - Queue_B
- You must figure out what these components implement (give reasons)

Task 2

- The black-box components DO NOT work with the normal NS2 simulator
- A new “bns” executable has been provided
  - Run the executable on solaris
    - Type: “> ./bns myscript.tcl”
- Scripts
  - Scripts that simulate each component are provided
  - Can also write your own scripts using the black-box components if result from scripts are not clear
Task 2: Details

- TCP
  - We will look at the following variants
    • Tahoe: slow-start + fast-retransmit
    • Reno: fast recovery
    • SACK: selective acknowledgements
  - There are 3 black box tcp’s and 3 scripts: tcp_X.tcl
  - To answer the questions you will need to:
    • Plot curves with xgraph or gnuplot
    • Answer some questions

Details cont’

- Fast TCP
  - Background
    • Version of TCP for large bandwidth-delay links
    • Increases the congestion window faster & avoids decreasing it unnecessarily
  - Investigate link sharing with regular TCP Reno
  - There are two scripts: Reno_Fast_tcp1.tcl & Reno_Fast_tcp2.tcl
Queueing

- Properties of RED vs. DropTail buffers
  - Provided with Queue_A & Queue_B
  - The scripts Queue_A.tcl and Queue_B.tcl simulate 10 tcp flows competing over 1 link
  - You can write your own scripts to do further testing

Submission

- The following will be required:
  - 2 Pdf files (task1.pdf & task2.pdf ) for written answers
  - Plots: paste images into the pdf files
  - Modified tcl scripts (where specified)
- Grading Scheme: see bottom of the project spec