Computer Network Lab (17CSL57)

NS2 Overview

Dr. Ram P Rustagi
Dept of CSE, KSIT
KRP-KSGI
rprustagi@ksit.edu.in

NS2 Resources

- NS2 tutorial
  - https://www.isi.edu/nsnam/ns/tutorial/index.html
  - http://www.engr.iupui.edu/~dskim/tutorials/print.php?article=ns2
- NS2 man page
  - `man ns`
  - https://ee.lbl.gov/ns/man.html
- Example tcl programs
  - **http://nile.wpi.edu/NS/simple_ns.html**
    - Default Lab01 used by most VTU affiliated colleges
  - https://www.cs.virginia.edu/~cs757/slidespdf/cs757-ns2-tutorial-exercise.pdf
  - [google guru]

Logistics

- Logistics of FDP Workshop
- Each session has two components
  - Theory session explaining the concepts (~45 mins)
  - Hands on session practice of concepts (~45 minutes)
- Theory session
  - Discuss key networking concept used in lab exercises
  - Discuss lab experiment design and implementation
- Hands on session
  - Practice basic concepts
  - Perform basic lab exercise
  - Result Analysis of lab exercise
    - Very important to assimilate the key concepts
  - Design at least 2 variations of lab exercises
    - Analyze and compare the results with basic exercise

Overview

- General Requirements
- Linux commands overview
- NS2 overview
- Defining nodes, links, application
- Traffic generation
- Animation visualization
- Trace file analysis

Basics of Linux Usage

- Launching Terminal window
  - `Ctrl + Alt + T`
  - Click on Terminal icon
- Launching multiple terminal window
  - `Ctrl + Alt + T`
  - Right click on *Terminal* icon
- Shut down
  - In terminal
    - `sudo shutdown -h now`
  - GUI: Select the sequence of shutdown menus

Basic Commands

- `ls` (`ls -l`)
  - list of files
- `pushd` <dir>; `popd`
- `mkdir` [`-p`] <dir>
- `more` <filename>;
  - `less` is more efficient
- `cat` <filename>
- `man` <command>;
- `man -k` <key>
- `Ctrl+C` (or `^C`) to terminate the running program
  - `Ctrl-Z` (or `^Z`) suspends, doesn't terminate
- `exit`
  - to exit from a window

NS2 in CN Lab

- Using Linux platform
  - Ubuntu 16.04 LTS (or 18.04, 14.04)
- Using Windows : More complicated
  - Recommended to install VirtualBox
  - Install Ubuntu `16.04` LTS in VirtualBox
  - Install `ns2`, `nam` and `xgraph` in Ubuntu
  - Might have issues in installing `nam`
    - May result in core dump on nam invocation
    - Need to reinstall correct version.

NS2 Installation

- On Ubuntu 16.04 LTS
  - `sudo apt install ns2`
- Installing `nam`
  - installation (`sudo apt install nam`) gives core dump
  - Download `nam_1.15-10-ubuntu14_amd64.deb`
    - `https://accsindia-my.sharepoint.com/:u:/g/personal/rprustagi_accsindia_org/ETq01F-AsLJPgh5XSyhBpJUBozbE31d-wPzcg_UlU3sgpw?e=dVfumI`
    - `sudo dpkg --install nam_1.15-10-ubuntu14_amd64.deb`
- On Ubuntu 12.04
  - `sudo apt-get install ns2`
  - `sudo apt-get install nam`

ns2 related commands

- ns <progname>.tcl
  - ns <dir/progname>.tcl
- nam <progname>.nam
  - nam <dir/progname>.nam
- xgraph <graphdata>.xgr

NS2 Architecture

- The simulator
  - `ns` (i.e. ns2)
- **N**etwork **A**ni**M**ator
  - `nam`
  - To visualize `ns2` execution in graphical way
  - Shows topology and pkt flows, pkt drops
- Xgraph: Graph for traffic flow (`xgraph`)
- NS2 execution
  - Pre-processing:
    - Topology creation, traffic generation
  - Post Processing
    - Analysis of trace files (using scripts, programs)
      - e.g. `shell`, `awk`, `perl`, `python` etc.

Overview of NS2 Usage

- NS2 : Network Simulator
  - A discrete event simulator
  - Simple model - single threaded
- Focused on modeling network protocols
  - Wired, wireless, satellite
  - TCP, UDP, multicast, unicast
  - Web, telnet, ftp
  - Ad hoc routing, sensor networks
  - Infrastructure: stats, tracing, error models, etc

NS Model

- Discrete event simulation
  - In network, each activity is an event
  - Simulator has list of events
  - Simulator takes one event at a time, runs it to completion, then next event
  - Each event happens in an instance of virtual time (real life event could be much longer)
  - Single thread model
    - No concurrency.

NS2 Software Structure

- *C++* for packet processing
  - Fast to run, mimics network objects
  - Provides full control on network events
  - Slow to write (new) and debug
- *Otcl* for controlling the simulation
  - Setup, configuration, etc.
  - Fast to write, slow to execute
- Lab experiment exercises (01 to 06):
  - Mostly dealing with *otcl*

Tcl Basics - Variables, Expressions

- Syntax has to be taken care of
  - SPACE needs to be used for separation
- Variable declaration
  - `set` <var> <value>
    - `set course "CSL57"`
  - `puts` "expression with evaluations"
    - `puts "Welcome to NS2 learning"`
  - variable usage is with `$` prefix
    - e.g. `puts "$course"`
- Expressions
  - declared in square brackets

```
[ expr 5 * 5 ]
[ $x * $x ]
```

Tcl Basics - procedures

Space is important

- Procedures

```
proc pow {base exp} {
   set result 1
   while {$exp > 0} {
      set result [expr $result * $base]
      set exp [expr $exp - 1]
   }
return $result
}
```

- Invocation of procedure

```
set chesssquares [pow 2 6]
```

- Example program

```
ns exponential.tcl
```

Tcl Basics - if … else

- Control flows: `if` and nested `if`

```
proc findmax {a b c} { # find max of 3 nums
if {$a < $b} {
   if {$b < $c} {
      set max $c
   } else {
      set max $b
   }
} else {
  if {$a < $c} {
     set max $c
  } else {
     set max $a
  }
return $max
}
```

- example program `findmax.tcl`

Tcl Basics - Loops

- Control flows - While and For loop

```
# print all integers up to n
# Using while loop
set N 5
set ii 1
while {$ii <= $N} {
   puts "$ii"
   set ii [expr $ii +1]
}
# Using for loop
for {set i 1} {$i <= $N} {incr i} {
   puts "$ii"
}
```

- Example program: `loops.tcl`

Tcl Basics - Variable Indirect Access

- Indirect access of variables

```
for {set i 1} {$i <= 3} {incr i} {
 set n$i [$i * $i]
}
for {set j 1} {$j <= 3} {incr j} {
 puts "[set n[set j]]"
}
```

- Example program: `indirect.tcl`

NS2 Basic Program Structure

- Create the event scheduler
- Turn on tracing
- Create network topology
- Create transport connections
- Generate traffic
- Start simulator
- Run network animation

Event Scheduler

- Creating scheduler

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

- Define termination time (by default : seconds)

```
set endtime 5.0
```

- Procedure finish

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

Tracing and Animation

- Enable all packet tracing
  - Tracing is log of events which is processed to determine outcome of simulation

```
set trfilename “simpleudp.tr"
$ns trace-all [open $trfilename w]
```

  - Can do partial tracing as well
- Enable Animation of events
  - Used for graphic visualization of events

```
set namfilename “simpleudp.nam”
$ns namtrace-all [open $namfilename w]
```

  - Can do partial tracing as well

Creating Network Topology

- Network consists of nodes and links (simple/duplex)
- Create two nodes and connect via a duplex link

```
set n1 [$ns node]
set n2 [$ns node]
#$ns <link_type> <n1> <n2> <bandwidth> <delay> <qtype>
$ns duplex-link $n1 $n2 1Mb 100ms DropTail
#$ns simplex-link $n1 $n2 1Mb 100ms DropTail
```

  - 2 similar simplex link (opp direction) = duplex link
- Qtype indicates which packet to be dropped
  - `DropTail`: Drop last pkt received on congestion
  - `RED`: Random early detection method
  - `SFQ` : Stochastic Fair Queueing
  - `FQ`(Fair Queueing), `DRR` (Deficit Round Robin)

Creating Network Topology

- Label the nodes (for display in animation)

```
$n1 label "sender"
$n2 label "receiver"
```

- Color the nodes (for display in animation)

```
$n1 color "red"
$n2 color "blue"
```

Create Applications (Agents)

- Create applications (agents)
  - Associate agents (applications) to nodes
  - One agent can be attached to only one node
  - Multiple agents can be attached to same node
- Identify underlying protocol
  - TCP, UDP, ICMP
- TCP, UDP are used by applications e.g. `ftp`, `telnet`
- ICMP used by applications e.g. `ping`
- For traffic transmission, needs sender and receiver
  - Receiver is *sink*(TCP), *Null* (UDP), *recv*(ICMP)
  - Sender is CBR(UDP), FTP(TCP), *ping*(ICMP)
- Simulator needs to know connectivity for traffic
  - From which sender to which receiver.

Create Applications - UDP…

- For each traffic, create one sender/receiver agent.
- Create UDP agents (source)

```
set udps [new Agent/UDP]
$ns attach-agent $n1 $udps
```

- Create UDP Receiver/ traffic sink

```
set udpr [new Agent/Null]
$ns attach-agent $n2 $udpr
```

- Define traffic characteristics e.g. CBR, pkt size, intervals…

```
set cbrs [new Application/Traffic/CBR]
$cbrs set packetSize_ 1000 #Trans delay
$cbrs set interval_ 50ms
#$cbrs set interval_ 0.05
```

Create Applications - UDP

- Associate traffic characteristics to the sender.
  - Note: Receiver has no role in characteristics
  - UDP is one way traffic (no acks) and thus consumes pkt

```
$cbrs attach-agent $udps
```

- Connect source application (UDP Sender) to receiver application (UDP Receiver) agent

```
$ns connect $udps $nullr
```

- Define time events when application starts and stops

```
$ns at 0.0 "$cbrs start"
$ns at 1.0 "$cbrs stop"
```

  - A UDP packet will be generated starting at 0.5s, with an interval of 500ms, pkt size being 500 bytes
  - This needs to be verified in the trace log.

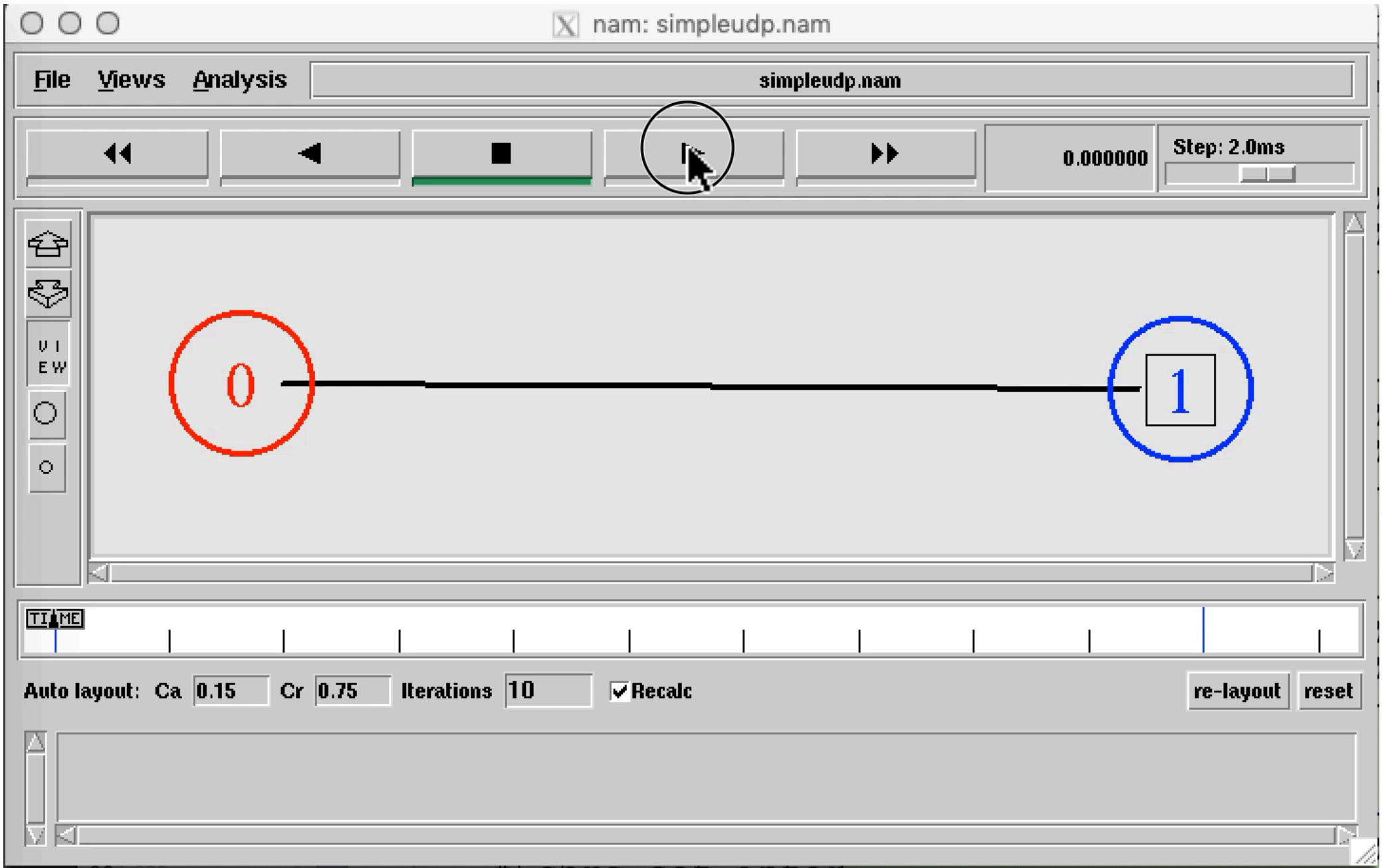
Start Applications

- Define the end time when simulation should end.
  ```
  $ns at 1.1 "finish"
  ```
- Start simulator i.e. define simulation start event
  ```
  $ns run
  ```
- Run the program
  - `ns simpleudp.tcl`
- Analysis of results
  - On completion, trace file `simpleudp.tr`, and
  - Animation activity file `simpleudp.nam`.
  - Size of these files will depend upon duration of the simulation and events generated.

Application Analysis

- Run the simulation
  - `nam simpleudp.nam`
- Tune the event occurence rate and study its impact on simulation progress
  - Move around the node graphically
  - For wire connected node, there is no impact.
  - On wireless nodes, there will be impact.

Animation SimpleUDP.mov



Analyzing Trace file

- Contents of trace file: following 12 fields
- `$1` Event : 4 values:
  +(Enqueue), −(dequeue), `r`(Recv), `d`(dropped)
- `$2` Time
- `$3` From Node - Input node of the link
- `$4` To node - output node of the link
- `$5` Pkt type - e.g CBR, TCP, Ping
- `$6` Pkt size - size in bytes
- `$7` Flags -
- `$8` Flow id - Flow identifier mostly for IPv6
- `$9` Src addr
- `$10` Dstn addr
- `$11` Seq num of network layer protocol pkt
- `$12` pkt id - Unique id of packet

Trace File: `simpleudp.tr`

```
$1 $2     $3 $4 $5  $6     $7       $8  $9  $10 $11 $12
+ 0       0  1 cbr 1000 -------- 0 0.0 1.0 0  0
- 0       0  1 cbr 1000 -------- 0 0.0 1.0 0  0
+ 0.05    0  1 cbr 1000 -------- 0 0.0 1.0 1  1
- 0.05    0  1 cbr 1000 -------- 0 0.0 1.0 1  1
+ 0.1     0  1 cbr 1000 -------- 0 0.0 1.0 2  2
- 0.1     0  1 cbr 1000 -------- 0 0.0 1.0 2  2
r 0.108   0  1 cbr 1000 -------- 0 0.0 1.0 0  0
+ 0.15    0  1 cbr 1000 -------- 0 0.0 1.0 3  3
- 0.15    0  1 cbr 1000 -------- 0 0.0 1.0 3  3
r 0.158   0  1 cbr 1000 -------- 0 0.0 1.0 1  1
+ 0.2     0  1 cbr 1000 -------- 0 0.0 1.0 4  4
- 0.2     0  1 cbr 1000 -------- 0 0.0 1.0 4  4
:
:
```

Hands on 1:

- Repeat the Experiment with following variations and analyze trace file and study the network animations
- Ex 1a: `n1→n2` and `n2→n1`
  - Generate UDP traffic in both directions (2 srcs, 2 dstns)
- Ex 1b:

  `n1→n2` : change the pkt size to 2000 bytes

  `n2→n1` : change the pkt size to 1500 bytes
- Ex 1c: for link `n1↔n2`, change
  - Link bandwidth to 500Kbps (from 1Mbps)
  - Propagation delay to 50ms
- Ex 1d: Configure parameters as per your choice, and
  - Study network behaviour.

Summary

- Nodes
- Links
- Application /agents
- Connecting sender and receiver
- Traffic generation (UDP)
- Events
- Trace file
- Animation