CSE 222
Graduate Networking

Winter 2001

Lecture 9: ns Tutorial

Dmitrii Zagorodnov

Overview

- Introduction to Tcl
- Introduction to OTcl
- Introduction to ns
- ns installation
- Midterm project

Tcl History

- Tcl = Tool Command Language
- Simple, extendible, interpreted language
- Tk is an extension for building GUI applications
- Invented by John Ousterhout
  - 1988 : first version of Tcl and Tk
  - 1990 : presented at USENIX
  - 1993 : first workshop at Berkeley
  - 1994 - 1997 : Ousterhout at Sun
  - 1997 - present : Ousterhout at Scriptics
- Can be given a script or used interactively (tclsh)

Basic Syntax

- Scripts are *commands* separated by newlines or ;

  cmd arg arg ...
- Variables are referenced using $

  $var
- Commands are nested using [ ]

  cmd [cmd arg ...] arg
- Words are grouped into single argument using “” or { }

  puts “Hello, $name”

  while {$i < 10} { incr i }
- Backslash is used to quote special characters (e.g. **\n**)
- Pound character # is used for comments (note **;#**)

Common Commands

- Set a variable

```
set x 2
set y $x
set y
```

- Math expressions

```
expr 3 + $x
```

- Producing output

```
set file [open filename.txt w]
puts file "3 + \$x = [expr 3 + $x]"
```


String Processing

- Many operations available via string command

```
string length $str
string index $str $index
```

- Concatenate strings using append

```
append str a b c
```

- Format output using format

```
puts [format "%05d %.5f" 10 12.23454356]
```

- Strings can be compared with ==

```
if {$x == "foo"}
```

- Strings can be treated as lists

```
lindex "one two three" 1
```

Control Structures

if { ... } { ...} elseif { ... } else { ... }

while { $x > 0 } { ... }

foreach x { a b c d } { ... }

for {set i 0} {$i < 10} {incr i 1} { ... }

Procedures

- Procedures: *proc name arglist body*

  proc add {a b} {return [expr $a + $b]}
- Envoke just like a command

  add 4 5
- Single, global scope for procedure names
- Global variables must be brought into procedures

  proc foo { ... } { global ns ... }
- "Qualified names" can be used to reference a variable in any scope

  ::x

OTcl Classes

- Create a class with *Class* command

  Class Bagel

  Bagel abagel or **set abagel [new Bagel]**
- Methods are defined using *instproc*, instance variables using *set* or *instvar*. Use *$self* to refer to the object inside its methods.

```
Bagel instproc toast { } {
        $self instvar toasted
                ...
}
```

- Constructors/destructors are called *init* and *destroy*

Inheritance in OTcl

- Classes can be extended using *-superclass*

  Class SpreadableBagel -superclass Bagel
- One can trace the hierarchy using *info*

```
SpreadableBagel info heritage
> Bagel Object
```

- Command *next* arranges for the next method with the same name up the inheritance tree to be invoked

```
SpreadableBagel instproc init {args} {
        eval $self next $args
        ...
}
```

Network simulator ns

- Discrete event simulator geared for network traffic simulation
  - Specifically, for tuning TCP
- Simulation is described by an OTcl script
- Simulation output can be visualized in *nam* and *xgraph*
- Used in many papers (NOTE: SACK and Vegas experiments are in ns distribution)

Typical ns script

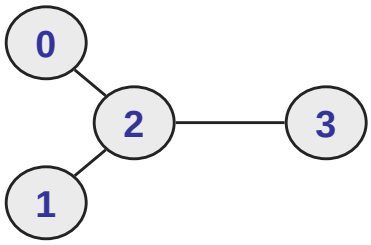
- Open trace files (by default everything is logged)
  - Careful, files can get big
- Build network topology
  - describe links, nodes, protocols, traffic sources/sinks
- New “agents” can be built by extending existing ones
- NOTE: Adding a brand new network protocol requires coding in C++ and recompilation of ns
- Schedule when traffic flows start and stop
- **$ns run**

Network Topology: Nodes and Links

```
set ns [new Simulator]

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
```



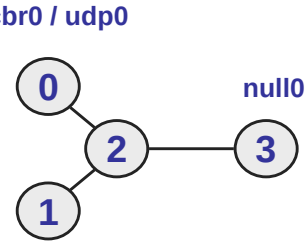
Network Topology: Agents and Applications

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
```



Scheduling Events

- Traffic flows can be started/stopped using *at*:

```
$ns at 0.5 “$cbr0 start”
$ns at 1.0 “$cbr0 stop”
```

- The script is typically terminated by a finish() proc

```
$ns at 5.0 “finish”
proc finish { } {
        global ns fd
        $ns flush-trace; close $fd
        exec nam out.nam &
        exit 0
}
```


Adding new agents

```
Class Agent/Message/Receiver -superclass
   Agent/Message

Agent/Message/Receiver instproc handle msg {
        $self instvar congested_
        if $congested_ {
                set congested_ 0
        } else {
                $self send "uncongested [lindex $msg 1]"
        }
}
```

ns Installation

- In order to use ns you can:
  - Use APE lab account (send mail to me if you don't have one)
  - Install ns-2 on your own machine (needs over 130Mb)
- Update or set 4 environment variables:
  - PATH
  - LD_LIBRARY_PATH
  - TCL_LIBRARY
  - NS
- Test your setup
  - ns $NS/ns-2/tcl/ex/simple.tcl
- Note: some scripts in example directory require it to be your current directory as they look for other files

Midterm Project

- Copy files from $NS/midterm into your own directory
- Two main parts:
  - Rate-based congestion control
  - Fair Queuing
- Turn in by email:
  - Write-up with answers to questions
  - Several graphs (in JPEG format)
- Assignment is up on 2/6/01 and due 2/18/01

For Next Time...

- Domain Name System
- Read and review Mockapetris and Dunlap paper from SIGCOMM ‘88
- Read section 9.1 in the textbook