What is ns?

- Network simulator
- A discrete event simulator
- 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 --- what is it good for?

Used to:
- Evaluate performance of existing network protocols.
- Prototyping and evaluation of new protocols.
- Large-scale simulations not possible in real experiments.

ns

How does it work:
- Event-driven simulator
  - Model world as events
  - Simulator has list of events
  - Process: take next one, run it, until done
  - Each event happens in instant of virtual time, but takes arbitrary real time
- Single thread of control
- Packet level
Ns models

- Traffic/applications
  - CBR, FTP, telnet, web
- Routing/Queuing
  - Drop-tail, FQ, SFQ, RED, DRR
  - Wired routing, adhoc routing etc
- Transport
  - TCP (variants), UDP, multicast (SRM)

ns - software structure

- Object oriented (C++, OTcl) – code reuse
- Scalability + Extensibility
  - Control/"data" separation
  - Split C++/OTcl object
- C++ for packet-processing (fast to run)
- OTcl for control - (fast to write)
  - Simulation setup and configuration

otcl and C++: The Duality

Outline

- Overview
- Tcl, OTcl basics
  - ns basics
  - Extending ns
  - ns internals
**Tcl basics**

```tcl
proc fact {x} {
    set ret 1
    if {$x > 2} {
        for {set i 1} {$i <= $x} {incr i} {
            set ret [expr $i * $ret]
        }
    }
    puts "factorial of $x is $ret"
} fact 5 \rightarrow factorial of 5 is 120
```

**Basic otcl**

```otcl
Class mom
    mom instproc init (age) {
        $self instvar age_
        set age_ $age
    }
    mom instproc greet () {
        $self instvar age_
        puts "$age_ years old mom: How are you doing?"
    }

set a [new mom 45] $a greet
```

**Tcl basics**

```tcl
proc fact {x} {
    set ret 1
    if {$x > 2} {
        for {set i 1} {$i <= $x} {incr i} {
            set ret [expr $i * $ret]
        }
    }
    puts "factorial of $x is $ret"
} fact 5 \rightarrow factorial of 5 is 120
```

**Basic otcl - inheritance**

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

set a [new mom 45] $a greet
```

- $ for de-referencing
- Spaces - important
- {} defines a block
- set, puts
- proc definition: proc name args body
- instead of single class declaration multiple definitions
- instproc adds class methods
- instvar adds instance variable, and brings them to the local scope
- Self - this in Java, C++
- all methods virtual (as in Java)
Outline

• Overview
• Tcl, OTcl basics
• ns basics
  • Extending ns
  • ns internals

Basic structure of ns-scripts

• Creating the event scheduler
• [Tracing]
• Creating network topology
• Creating Transport Layer - Agents
• Creating Applications - Applications
• Events!

Creating Event Scheduler

• Create scheduler
  • set ns [new Simulator]

• Schedule event
  • $ns at <time> <event>
  • <event>: any legitimate ns/tcl commands

• Start scheduler
  • $ns run

“Hello World” in ns

```
simple.tcl
set ns [new Simulator]
$ns at 1 "puts \"Hello World!\""
$ns at 1.5 "exit"
$ns run
```
bovik@gs19% ns simple.tcl
Hello World!
bovik@gs19%
Creating Network

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

- Links & Queuing
  - $ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>
  - Queue type: DropTail, RED, CBQ, FQ, SFQ, DRR

Routing + traffic

- Unicast
  - $ns rproto <type>
  - <type>: Static, Session, DV

- Multicast support also.

- Traffic
  - Simple two layers: transport and application.
  - Transport: TCP, UDP etc.
  - Applications: web, ftp, telnet etc.

Transport Layer

Class Agent

Agent/UDP  Agent/TCP (=Tahoe)

Other TCP flavors

The transport layer: UDP

- UDP
  - set udp [new Agent/UDP]
  - set null [new Agent/NULL]

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

  - $ns connect $udp $null
The transport layer: TCP

- TCP
  - set tcp [new Agent/TCP]
  - set tcpsink [new Agent/TCPSink]
- $ns attach-agent $n0 $tcp
- $ns attach-agent $n1 $tcpsink
- $ns connect $tcp $tcpsink

Transport Layer

Class Agent

- Agent/UDP
- Agent/TCP (=Tahoe)
- Other TCP flavors
- Agent/TCP/FuTCP

Application Layer

Class Application

- (Simulated Applications) (on top of TCP)
- (Traffic generators) (on top of UDP)

Creating Traffic: On Top of TCP

FTP
- set ftp [new Application/FTP]
- $ftp attach-agent $tcp
- $ns at <time> "$ftp start"

Telnet
- set telnet [new Application/Telnet]
- $telnet attach-agent $tcp
Creating Traffic: On Top of UDP

- CBR
  - set src [new Application/Traffic/CBR]
- Exponential or Pareto on-off
  - set src [new Application/Traffic/Exponential]
  - set src [new Application/Traffic/Pareto]
- Trace driven traffic
  - Inter-packet time and packet-size

Attaching a traffic source

- set cbr [new Application/Traffic/CBR]
- $cbr$ attach-agent $udp$
- $ns$ at <time> “$cbr$ start”

Tracing

Trace packets on all links:
- set [open out.tr w]
- $ns$ trace-all $f$
- $ns$ flush-trace
- close $f$

More Tracing

- Tracing specific links
  - $ns$ trace-queue $n0$ $n1$ $f$
- Tracing variables
  - set cwnd_chan_ [open all.cwnd w]
  - $tcp$ trace cwnd_
  - $tcp$ attach $cwnd_chan$
Controlling object parameters

- Almost all ns objects have parameters
  - ex. Application/Traffic/Exponential has rate and packetSize
- set parameters in OTcl
  - set etraf [new Application/Traffic/Exponential]
  - set etraf rate_1Mb
  - set etraf packetSize_1024

Putting it all together

- set ns [new Simulator]
  - set n0 [$ns node]
  - set n1 [$ns node]
  - $ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
  - $ns trace-queue $n0 $n1 $!
  - set tcp [$ns create-connection TCP $n0 $TCPSink $n1 1.0]
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp
  - $ns at 0.2 "$ftp start"
  - $ns at 1.2 "exit"
  - $ns run

Creating Topology

Creating Transport layer

Creating Applications

Schedule Events

nam – the network animator

set nf [open out.nam w]
$ns namtrace-all $nf
...
exec nam out.nam &

ns “components”

- ns, the simulator itself
- nam, the Network AniMator
  - Visualize ns output
  - GUI input simple ns scenarios
- Pre-processing:
  - Traffic and topology generators
- Post-processing:
  - Simple trace analysis, often in Awk, Perl, or Tcl
Network Dynamics: Link failures

- `$ns rtmodel-at <time> <up|down> $n0 $n1`
- `$ns rtmodel Trace <config_file> $n0 $n1`
- `$ns rtmodel <model> <params> $n0 $n1`  
  `<model>`: Deterministic, Exponential

Issues in Simulations

- Suppose you want to study the way TCP sources share a bottleneck link...
- Which topology?
- Which traffic sources?  
  Background Traffic?
- When to start sources?
- What else affects results?

Another practical issue: Memory

- `$ns/tcl/ex/cmcast-150.tcl`:
  - 150 nodes, 2200 links => 53MB
  - 2420 nodes, 2465 links => 800MB
- Avoid `trace-all`
- Use arrays for a sequence of variables
  - Instead of `n$i`, say `n($i)`

Outline

- Overview
- Tcl, OTcl basics
- ns basics
- Extending ns
  - New protocols, functionality
  - ns internals
Making changes to ns – where???

Where would you implement
• one-time configuration variables
• complex procedures
• per packet action

ns directory structure

New component purely in Otcl

New component in C++

• Create C++ class, fill in methods
• Define otcl linkage
• Write otcl code (if any)
• Build (and debug)
Outline

- Overview
- Tcl, OTcl basics
- ns basics
- Extending ns
- ns internals

How does linkage work?

- how to access Tcl variables from C++
- how is C++ object created from interpreter
  * ....

TclObject: Hierarchy and Shadowing

TclObject: Hierarchy and Shadowing

TclObject

- Example
  set tcp [new Agent/TCP]
  => how is corresponding C++ object created?
  $tcp set window_ 500
  => how is corresponding C++ variable set?
  $tcp advance 5000
  => how is C++ procedure called?
TclObject::bind()

- Link C++ member variables to otcl object variables
- C++
  ```cpp
tcpAgent::TcpAgent()
  {
    bind("window ", &wind);
    ...
  } -

  - bind_time(), bind_bool(), bind_bw()
  - otcl
  $tcp set window 200
```

TclObject::command()

- Implement otcl methods in C++
- Trap point: otcl method cmd()
- Send all arguments after cmd() call to TclObject::command()
**TclObject::command()**

- **otcl**
  
  ```
  set tcp [new Agent/TCP]
  $tcp advance 10
  ```

- **C++**

  ```
  int TcpAgent::command(int argc, const char* const* argv) {
    if (argc == 3) {
      if (strcmp(argv[1], "advance") == 0) {
        int newseq = atoi(argv[2]);
        ... return(TCL_OK);
      }
    }
    return (Agent::command(argc, argv));
  }
  ```

**TclObject**

- **Example**

  ```
  set tcp [new Agent/TCP]
  => how is corresponding C++ object created?
  $tcp set window_ 500
  => how is corresponding C++ variable set?
  $tcp advance 5000
  => how is C++ procedure called?
  ```

**TclObject: Creation and Deletion**

**TclClass**

```
Static class TcpClass : public TclClass {
  public:
  TcpClass() : TclClass("Agent/TCP") {}
  TclObject* create(int, const char* const*) {
    return (new TcpAgent());
  }
} class_tcp;
```
**Class Tcl**

- Singleton class with a handle to Tcl interpreter
  - While writing C++ code
- Usage
  - Invoke otcl procedure
  - Obtain otcl evaluation results
  - Pass a result string to otcl
  - Return success/failure code to otcl

```cpp
Class Tcl

Tcl tcl = Tcl::instance();

Passing results to the interpreter:
if (strcmp(argv[1], "now") == 0) {
    tcl.resultf("%g", clock());
    return TCL_OK;
}

if (strcmp(argv[1], "helloworld") {
    tcl.evalc("puts stdout Hello World");
    return TCL_OK;
}
```

**Class TclCommand**

- C++ implementation of global otcl commands

```cpp
class RandomCommand : public TclCommand {
public:
    RandomCommand() : TclCommand("ns-random") {} 
    virtual int command(int argc, const char*const* argv) {
        Tcl tcl = Tcl::instance();
        if (argc == 1) {
            sprintf(tcl.buffer(), "%u", Random::random());
            tcl.result(tcl.buffer());
        }
    }

    virtual int evalc(const char* cmd) {
        return tcl.evalc(cmd);}
};
```

**Summary**

- Root of ns-2 object hierarchy
- bind(): link variable values between C++ and OTcl
- command(): link OTcl methods to C++ implementations
- TclClass: Create and initialize TclObject's
- Td: C++ methods to access Tcl interpreter
- TclCommand: Standalone global commands
- EmbeddedTcl: ns script initialization
Useful Resources

- Marc Greis ns-2 tutorial
  http://www.isi.edu/nsnam/ns/tutorial/
- Tcl/Tk http://www.tcl.tk/man/tcl8.4/
- NS http://www.isi.edu/nsnam/ns/
- http://nile.wpi.edu/NS/