What is ns?

- **Network simulator**
- VINT Project -- Virtual Internet Testbed
- Collaboration of many institutes
  - AT&T
  - UC Berkeley
  - XEROX PARC
  - ETH
- **Goals**

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.

How does it work:

- Event-driven simulator
- Packet level
- Link layer and up

Development Status

Current status:

- 100K lines of C++ code
- 70K lines of otcl code
- 20K lines of documentation

otcl and C++: The Duality

Current status:

<table>
<thead>
<tr>
<th>Otel / Tcl</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>tclcl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C++</td>
</tr>
</tbody>
</table>

Your ns-script
Basic otcl

```tcl
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
```

- instead of single class declaration multiple definitions
- instproc adds method
- instvar adds instance variable
- methods are always called through object ($self)
- init instproc instead of constructor

Basic otcl - inheritance

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

set b [new kid 15] $b greet
```

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

```tcl
simple.tcl
set ns [new Simulator]
$ns at 1 "puts "Hello World!"
$ns at 1.5 "exit"
$ns run

gs116% ns simple.tcl
Hello World!
gs116%
```

Creating Network

- Nodes
  - set n0 [$ns node]
  - set n1 [$ns node]
- Links & Queuing
  - $ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>
**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

**Two-way TCP**

- FullTcp connection
  - set tcp1 [new Agent/TCP/FullTcp]
  - set tcp2 [new Agent/TCP/FullTcp]
  - $ns attach-agent $n1 $tcp1
  - $ns attach-agent $n2 $tcp2
  - $ns connect $tcp1 $tcp2
  - $tcp2 listen

**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]

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 f[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

Putting it all together

- set ns [new Simulator]
- set n0 [new node]
- set n1 [new node]
  - $ns duplex-link $n0 $n1 1.5Mb
  - 10ms DropTail
  - $ns trace-queue $n0 $n1 $f

- set tcp [new create-connection TCP]
  - $n0 TCPSink $n1 $0

- set ftp [new Application/FTP]
  - $ftp attach-agent $tcp
  - $ns at 0.2 "$ftp start"
  - $ns at 1.2 "exit"

- $ns run

Is tracing all links always the best thing to do?
nam – the network animator

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

Computing routes

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

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 linkd => 800MB
```

- Avoid `trace-all`
- Use arrays for a sequence of variables
  - Instead of `n$i`, say `n($i)`

Basic ns-2: Not Covered

- mobile IP
- multicasting
- satellite
- emulation
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)

How does linkage work?

• how to access Tcl variables from C++
• how is C++ object created from interpreter
• ……

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: Hierarchy and Shadowing

- Link C++ member variables to otcl object variables
- C++
  TcpAgent::TcpAgent() {
    bind("window_", &wnd_);
    ...
  }
  *bind_time(), bind_bool(), bind_bw()
- otcl
  $tcp set window_ 200

TclObject::bind()

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

TclObject::command()

- otcl
  $tcp advance
  no such procedure
- 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
  
  ```tcl
  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?
  ```

**TclClass**

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

**Class Tcl**

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

```c
Tcl& tcl = Tcl::instance();
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

```c
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());
      }
    }
  }
}
```
## Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TclObject</strong></td>
<td>Root of ns-2 object hierarchy</td>
</tr>
<tr>
<td></td>
<td><code>bind()</code>: link variable values between C++ and OTcl</td>
</tr>
<tr>
<td></td>
<td><code>command()</code>: link OTcl methods to C++ implementations</td>
</tr>
<tr>
<td><strong>TclClass</strong></td>
<td>Create and initialize TclObject’s</td>
</tr>
<tr>
<td><strong>Tcl</strong></td>
<td>C++ methods to access Tcl interpreter</td>
</tr>
<tr>
<td><strong>TclCommand</strong></td>
<td>Standalone global commands</td>
</tr>
<tr>
<td><strong>EmbeddedTcl</strong></td>
<td>ns script initialization</td>
</tr>
</tbody>
</table>