ns-2 Introduction
Dr Osman Ghazali
Credits

• Some of this material is based on, variously:
  – ns-2 documentation
  – NS2 Online Tutorial

• Any mistakes certainly comes from me 😊
Outline

- Background
- Usage
- Simulation
- Fundamentals
- Infrastructure
- Debugging
Outline

• Background

• Usage
• Simulation
• Fundamentals
• Infrastructure
• Debugging
Useful links

- **ns-2 web page**
  [http://www.isi.edu/nsnam/ns/](http://www.isi.edu/nsnam/ns/)

- **nam web page**
  [http://www.isi.edu/nsnam/nam/](http://www.isi.edu/nsnam/nam/)

- **ns-2 tutorial**

- **ns-2 workshops**

- **ns-2 manual**
  [http://www.isi.edu/nsnam/ns/ns-documentation.html](http://www.isi.edu/nsnam/ns/ns-documentation.html)
ns-2, the Network Simulator

• A *discrete event simulator* modelling *network protocols*
  - Packets, links, queues, protocols
  - Visualizer (*NAM*)
  - Trace playback
  - Error models

• There are alternatives!
  - Experimentation
  - Analysis
  - Other simulators

- ✅ detail
- ✗ expense, scale
- ✅ understanding
- ✗ limited detail
- ✅ detail within niche
- ✗ niches, limited reuse
History

• 1989: REAL by Keshav
• 1995: ns by Floyd, McCanne at LBL
• 1997: ns-2 by the VINT project (Virtual InterNetwork Testbed) at LBL, Xerox PARC, UCB, USC/ISI
• Now: ns-2.29 maintained at USC/ISI
  – ns-2.30 pending release
Components

- ns-2.29
- tcl8.4.11
- tk8.4.11
- tclcl-1.17
- otcl-1.11
- nam-1.11

- tcl
  - doc
  - ns-tutorial
  - ...
  - models

- ex
  - test
  - lib
  - ...

- C++

...and, of course, your simulation script!
Components

• *ns-2*, the simulator itself
  – Specify simulation, generate traces
  – Depends on Tcl/Tk, OTcl, TclCL
• *nam*, the network animator
  – Animate traces from simulation
  – GUI for constructing simple simulations
• Pre-processing
  – Traffic, topology generation
• Post-processing
  – Analyse trace output with *awk*, etc
Status

• *ns-2.34 (Pending)*, October 2008
  – ~330 kLOC C/C++, ~250 kLOC Tcl/OTcl
  – ~100 test suites and 100+ examples
  – ~400 pages of ns manual

• Platform support
  – FreeBSD, Linux, Solaris, Windows, Mac

• User base
  – >1k institutes (50 countries), >10k users
  – ~300 posts/month to ns-users@isi.edu
  – ns-users@mash.cs.berkeley.edu
    • majordomo@mash.cs.berkeley.edu
    • subscribe ns-users yourname@address
Outline

• Background

• Usage

• Simulation

• Fundamentals

• Infrastructure

• Debugging
Using ns

Define problem → Simulation model → Execute simulation → Post-process results → Extend simulator
Using ns

• Create simulation
  – Describe network, protocols, sources, sinks
  – Interface via OTcl which controls C++

• Execute simulation
  – Simulator maintains event list (packet list), executes next event (packet), repeats until done
  – Events happen instantly in virtual time but could take arbitrarily long real time
  – Single thread of control, no locking, races, etc

• Post-process results
  – Scripts (awk, perl, python) to process text output
  – No standard library but some available on web
Languages

- **C++ for data**
  - Per-packet processing, the core of ns
  - Fast to run, detailed, complete control

- **OTcl for control**
  - Simulation description
  - Periodic or triggered actions
  - Manipulating existing C++ objects
  - Faster to write and change
Basic Tcl

# Variables:
set x 10
set x
puts “x is $x”

# Functions and expressions:
set y [pow x 2]
set y [expr x*x]

# Control flow:
if {$x > 0} { return $x } else {
    return [expr -$x] }
while { $x > 0 } {
    puts $x
    incr x -1
}
for {set i 0} {$i<10} {incr i} {
    puts “$\i == $i”
}

# Procedures:
proc pow {x n} {
    if {$n == 1} { return $x }
    set part [pow x [expr $n-1]]
    return [expr $x*$part]
}

# Files:
set file [open “nstrace.txt” w]
set line [gets $file]
puts -nonewline $file “hello!”
close $file
Basic OTcl

**Class** Person

# constructor:
Person **instproc** init {age} {
    $self instvar age_
    set age_ $age
}

# method:
Person **instproc** greet {} {
    $self instvar age_
    puts “age $age_: Hello!”
}

# subclass:

Class Child -**superclass** Person
Child **instproc** greet {} {
    $self instvar age_
    puts “age $age_ kid: Wassup!”
}

set a [new Person 45]
set b [new Child 15]
$a greet
$b greet
C++/OTcl Linkage

- **OTcl creates objects in the simulator**
  - Objects are *shared* between OTcl and C++ by default
  - Access the OTcl interpreter via class `Tcl`
    ```
    Tcl &tcl = Tcl::instance();
    tcl.eval(...); tcl.evalc(""); tcl.evalf("",...);
    tcl.result(...); res = tcl.result();
    ```
  - `ns2` also keeps hashtable of every TclObject
    ```
    tcl.enter()/.lookup()/.remove()
    ```

- **Variables are, by default, *unshared***
  - Pure C++ or OTcl variables
  - Bindings can be established by the compiled constructor
    ```
    bind(); bind_delay(); { bind("rtt_", &t_rtt_); t_rtt_ = 10; }
    ```

Evaluate strings in interpreter

Return results to interpreter

Access results from interpreter

Couple C++ object instance variable `t_rtt_` with OTcl object instance variable `rtt_`
C++/OTcl Linkage

• Commands and methods
  – For all TclObject, ns creates cmd{} instance procedure to access compiled methods
  – Consider $o distance? <agentaddr>
    • Invokes distance?{} instance procedure of $o
  – If this doesn’t exist, call parent TclObject unknown{} method
  – ...which calls $o cmd distance? <agentaddr>
  – ...which calls C++ <T>::command() method
struct hdr_ping {
    char ret; // required by PacketHeaderManager
    double send_time; double rcv_time; int seq;

    static int offset_; // required by PacketHeaderManager
    inline static int offset() { return offset_; }
    inline static hdr_ping* access(const Packet* p) { return (hdr_ping*) p->access(offset_); }
};

class PingAgent : public Agent {
public:
    PingAgent();
    int seq; // a send sequence number like in real ping
    virtual int command(int argc, const char*const* argv);
    virtual void recv(Packet*, Handler*); 
};
PingAgent::PingAgent() : Agent(PT_PING), seq(0) {
    bind("packetSize_", &size_);
}

int PingAgent::command(int argc, const char*const* argv) {
    if (argc == 2) {
        if (strcmp(argv[1], "send") == 0) {
            Packet* pkt = allocpkt();
            hdr_ping* hdr = hdr_ping::access(pkt);
            hdr->ret = 0; hdr->seq = seq++;
            hdr->send_time = Scheduler::instance().clock();
            send(pkt, 0); // Agent::send()
            return (TCL_OK);
        } else if (/* etc */) else { /* etc */ }
    }
    return (Agent::command(argc, argv));
}
Outline

- Background
- Usage
- Simulation
  - Model. Class hierarchy.
- Fundamentals
- Infrastructure
- Debugging
From Network to Simulation

Application, Agent & Node

Node

Link

Node

Link

Node

Link

Node

Link

Node

Link

Node
Class Hierarchy (partial)

- **TclObject**
  - Receive packets and transmit to `target_`
  - Basis for `Agents` and `Links` (~`Queue` + `Delay`)

- **NsObject**

- **Connector**

- **Queue**

- **Delay**

- **Agent**

- **Trace**

- **Classifier**

- **AddressClassifier**
  - Table of \( n \) slots each pointing to a \texttt{TclObject}
  - `classify()` identifies destination slot for packet
  - `AddressClassifier`, `PortClassifier` found within `Nodes`

- **Reno**

- **SACK**
Outline

- Background
- Usage
- Simulation
- Fundamentals
- Infrastructure
- Debugging
Components

• Four major types
  – Application
    • Communication instigator
  – Agent
    • Packet generator/consumer
  – Node
    • Addressable entity
  – Link
    • Set of queues
Applications (Ch.36, p.323)

- Poke the agent to which they’re attached
  - Introduction of Process class
    - C.f. documentation!
    - Models any entity that is capable of receiving, requesting or processing data

- Two basic types:
  - class Process : public TclObject
    [ns-2.29/common/ns-process.h]
  - class Application : public Process
    [ns-2.29/apps/app.h]
  - class TrafficGenerator : public Application
    [ns-2.29/tools/trafgen.h]
Applications

• Application
  - Assumes attached to a TCPAgent
  - start(), stop(), send(), recv()
  - E.g. class TelnetApp
    [ns-2.20/apps/telnet.h]
    • Schedule agent_->sendmsg() calls based on exponential interarrival timer

• TrafficGenerator
  - Assumes attached to a UDPAgent
  - init(), next_interval()
  - E.g. class POO_Traffic
    [ns-2.29/tools/pareto.cc]
    • Schedule bursts of packets (Pareto on-off source)
Agents

- **TCP/UDP stacks**
  - `timeout()`, `send()`, *etc*
    - Allocate and schedule packets
  - `recv()`, *etc*
    - Callback to `app_` to notify of data
- **Subtype of Connector**
  - class `Agent` : public Connector
    - `ns-2.29/common/agent.h, ...	tcl/lib/ns-agent.tcl`
  - class `TcpAgent` : public `Agent`
    - class `FullTcpAgent` : public `TcpAgent`
  - class `UdpAgent` : public `Agent`
Nodes

- Addressable entity built from classifiers
  - Distributes incoming data to agents
  - Distributes outgoing data to links
- Simplest unicast case has address and port classifiers, others may have more
- Node, LanNode, etc derived from ParentNode

[ns-2.29/common/{parentnode,node}.h]
Links (Ch.6, p.62)

- An OTcl amalgam of C++ objects
  
  [ns-2.29/tcl/lib/ns-link.tcl]
  
  - class SimpleLink - superclass Link
  
  - Simulator instproc duplex-link { n1 n2 bw delay type args }

- More complex links may have complex link delay/bw characteristics...
  
  [ns-2.29/link/delay.h]
  
  - class LinkDelay : public Connector

- ...and multiple queue_ elements
  
  [ns-2.29/queue/{queue.h,red.h,...}]
  
  - class Queue : public Connector
  
  - class RedQueue : public Queue
$\text{ns } \text{duplex-link } n0 \ n1 \ 5\text{Mb} \ 2\text{ms} \ \text{DropTail}$
Packets

next_

hdrlen_

data_

bits_

accessdata()

userdata()

access(int)
bits()

IP header

TCP header

RTP header

Packet

next_

hdrlen_

data_

bits_
Packets

- **Derived from base** class Event
  - Other derived class is at-event (OTcl)
- **Packets are set of headers plus data**
  - Default is to include *all headers of all types in all packets*, giving >3kB *per-packet!*
  - Turn off unnecessary headers *before* creating simulator object (*common always required*)
  - *remove-packet-header* AODV ARP …, *or*
  - *remove-all-packet-headers* add-packet-header IP TCP …
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PacketHeader/LRWPAN</td>
<td>216</td>
</tr>
<tr>
<td>PacketHeader/XCP</td>
<td>64</td>
</tr>
<tr>
<td>PacketHeader/Lms</td>
<td>56</td>
</tr>
<tr>
<td>PacketHeader/PGM</td>
<td>16</td>
</tr>
<tr>
<td>PacketHeader/PGM_SPM</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/PGM_NAK</td>
<td>16</td>
</tr>
<tr>
<td>PacketHeader/Pushback</td>
<td>4</td>
</tr>
<tr>
<td>PacketHeader/NV</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/LDP</td>
<td>40</td>
</tr>
<tr>
<td>PacketHeader/MPLS</td>
<td>20</td>
</tr>
<tr>
<td>PacketHeader/rtProtoLS</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/Ping</td>
<td>32</td>
</tr>
<tr>
<td>PacketHeader/TFRC</td>
<td>56</td>
</tr>
<tr>
<td>PacketHeader/TFRC_ACK</td>
<td>64</td>
</tr>
<tr>
<td>PacketHeader/Diffusion</td>
<td>192</td>
</tr>
<tr>
<td>PacketHeader/RAP</td>
<td>24</td>
</tr>
<tr>
<td>PacketHeader/AODV</td>
<td>808</td>
</tr>
<tr>
<td>PacketHeader/SR</td>
<td>720</td>
</tr>
<tr>
<td>PacketHeader/TORA</td>
<td>32</td>
</tr>
<tr>
<td>PacketHeader/IMEP</td>
<td>512</td>
</tr>
<tr>
<td>PacketHeader/ARP</td>
<td>32</td>
</tr>
<tr>
<td>PacketHeader/MIP</td>
<td>32</td>
</tr>
<tr>
<td>PacketHeader/IPinIP</td>
<td>4</td>
</tr>
<tr>
<td>PacketHeader/LL</td>
<td>32</td>
</tr>
<tr>
<td>PacketHeader/Mac</td>
<td>40</td>
</tr>
<tr>
<td>PacketHeader/Encap</td>
<td>4</td>
</tr>
<tr>
<td>PacketHeader/HttpInval</td>
<td>4</td>
</tr>
<tr>
<td>PacketHeader/MFTP</td>
<td>64</td>
</tr>
<tr>
<td>PacketHeader/SRMEXT</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/SRM</td>
<td>16</td>
</tr>
<tr>
<td>PacketHeader/aSRM</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/mcastCtrl</td>
<td>20</td>
</tr>
<tr>
<td>PacketHeader/CtrMcast</td>
<td>12</td>
</tr>
<tr>
<td>PacketHeader/rtProtoDV</td>
<td>4</td>
</tr>
<tr>
<td>PacketHeader/GAF</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/Snoop</td>
<td>24</td>
</tr>
<tr>
<td>PacketHeader/SCTP</td>
<td>8</td>
</tr>
<tr>
<td>PacketHeader/TCPA</td>
<td>16</td>
</tr>
<tr>
<td>PacketHeader/TCP</td>
<td>80</td>
</tr>
<tr>
<td>PacketHeader/IVS</td>
<td>32</td>
</tr>
<tr>
<td>PacketHeader/RTP</td>
<td>12</td>
</tr>
<tr>
<td>PacketHeader/Message</td>
<td>64</td>
</tr>
<tr>
<td>PacketHeader/Resv</td>
<td>16</td>
</tr>
<tr>
<td>PacketHeader/TCP_QS</td>
<td>12</td>
</tr>
<tr>
<td>PacketHeader/UMP</td>
<td>16</td>
</tr>
<tr>
<td>PacketHeader/Src_rt</td>
<td>76</td>
</tr>
<tr>
<td>PacketHeader/IP</td>
<td>28</td>
</tr>
<tr>
<td>PacketHeader/Common</td>
<td>104</td>
</tr>
<tr>
<td>PacketHeader/Flags</td>
<td>9</td>
</tr>
</tbody>
</table>
A Simple Topology
Outline

• Background
• Usage
• Simulation
• Fundamentals
• Infrastructure
• Debugging
Addressing

- Two modes: *default* and *hierarchical*
  
  \[\text{ns-2.29/tcl/lib/ns-address.tcl}\]
  
  - Default: 32 bits address, 32 bits port, 1 bit multicast
  - Hierarchical: *default* and *specific*
    
    - Default: 3 levels, 10/11/11 bits per level
    - Specific:
      \[
      \text{\$ns set-address-format hierarchical <nlevels> <bits in level1>}
      \]

- Nodes are automatically assigned addresses
  
  - But if you want to generate e.g. simulations with subnet structure, do it yourself
Links have assigned weights
  - Default to 1
    - $\text{ns cost n0 n1 cost}$

Three supported types:
  - Static
    - Simple Dijkstra, computed at start-of-day
  - Session
    - Simple Dijkstra, computed at each topology change
  - Distance Vector (DV)
    - RIP-like: periodic & triggered updates, split horizon, poison reverse
  - Link state “highly experimental”
    [ns-2.29/linkstate/]
Routing

- For dynamic ("session") routing, need a failure model: there are four
  - `$ns rtmodel Trace <config_file> $n0 $n1`
  - `$ns rtmodel Exponential {<params>} $n0 $n1`
  - `$ns rtmodel Deterministic {<params>} $n0 $n1`
  - `$ns rtmodel-at <time> up|down $n0 $n1`

- You can also define your own
Maths

• **Support classes** \([\text{ns-2.29/tools/}]\)

• class **Integrator**
  - Simple linear interpolation integrator

• class **Samples**
  - Tracks set of sample points (count, sum, sum^2, mean, var)

• class **Random, <type>RandomVariable**
  - Wrapper around RNG
  - Uniform, exponential, pareto, normal, lognormal generators

• class **RNG**
  - Implementation of pseudo-random number generator with a period of \(2^{31}-2\)

  » (possibly; documentation and rng.h disagree on this point)
Tracing (Ch.24, p.223)

- Packet tracing and event tracing
  - Objects that are inserted between nodes
  - Insert from OTcl using `trace`{} method
    - `$ns trace-all $file`, or
    - `[$ns link $n0 $n1] trace $file`

- Monitoring
  - Record counters of interest for all packets or on a per-flow basis

- ...or roll your own
Tracing

• *ns* and *nam* trace file formats
  - *ns* also supports `show_tcphdr_` variable controlling display of TCP flags

• *ns* format

<table>
<thead>
<tr>
<th>op</th>
<th>ts</th>
<th>prevhop</th>
<th>nexthop</th>
<th>type</th>
<th>size</th>
<th>flags</th>
<th>flowid</th>
<th>src</th>
<th>dst</th>
<th>seqno</th>
<th>pktid</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1.102</td>
<td>0</td>
<td>1</td>
<td>tcp</td>
<td>40</td>
<td>------</td>
<td>0</td>
<td>0.0</td>
<td>1.0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• Enque [+] , dequeue [-], receive [r], drop [d]

• Types are

• Flags are
  - ECN-CE [E], -ECT [N], -CE [C], -CWR [A]; plus
  - priority [P] and TCP fast start [F]


Outline

• Background
• Usage
• Simulation
• Fundamentals
• Infrastructure
• Debugging
Debugging

• Split object model is a PITA
• **Use Don Lib’s Tcl Debugger and gdb**
  - See the *ns-2* documentation for details
  - **Use** `call Tcl::instance().eval()` to bounce between C++ and OTcl
  - `$ns gen-map{}`, prints all objects

• Memory debugging is probably the biggest issue
  - **Use** *dlmalloc*
Debugging

• OTcl leaks: it does *not* garbage collect
  – set ns [new Simulator]
    for {set i 0} {$i < 500} {incr i} {
      set a [new RandomGenerator/Constant]
    }
  – This will generate \(~500\) objects
  – Free things yourself \((\text{delete } \$a)\)

• Conservation
  – Avoid \($ns \text{ trace-all } \$f\)
    • Allocates trace objects on all links, 14kB/link
  – Remove unnecessary packet headers
  – Use arrays for sequences
  – Avoid naming objects unnecessarily
  – Use `\texttt{delay\_bind()}` *rather than* `\texttt{bind()}` in C++ objects
Debugging

- **OTcl leaks: it does *not* garbage collect**
  - `set ns [new Simulator]
    for {set i 0} {$i < 500} {incr i} {
      set a [new RandomGenerator/Constant]
    }
  - This will generate ~500 objects
  - Free things yourself (`delete $a`)

- **Conservation**
  - Avoid `$ns trace-all $f`
  - Remove unnecessary packet headers
    - Reduces from ~3kB/packet to ~100B/packet
  - Use arrays for sequences
  - Avoid naming objects unnecessarily
  - Use `delay_bind()` rather than `bind()` in C++ objects
Debugging

- **OTcl leaks: it does *not* garbage collect**
  - set ns [new Simulator]
    for {set i 0} {$i < 500} {incr i} {
      set a [new RandomGenerator/Constant]
    }
  - This will generate ~500 objects
  - Free things yourself (delete $a)

- **Conservation**
  - Avoid $ns trace-all $f
  - Remove unnecessary packet headers
  - Use arrays for sequences
    - for {set i 0} {$i<500} {incr i} {set n$1 [$ns node]}
    - vs.
      for {set i 0} {$i<500} {incr i} {set n($1) [$ns node]}
    - Saves ~40B/variable
  - Avoid naming objects unnecessarily
  - Use delay_bind() rather than bind() in C++ objects
Debugging

• OTcl leaks: it does *not* garbage collect
  - set ns [new Simulator]
    for {set i 0} {$i < 500} {incr i} {
      set a [new RandomGenerator/Constant]
    }
  - This will generate ~500 objects
  - Free things yourself (delete $a)

• Conservation
  - Avoid $ns trace-all $f
  - Remove unnecessary packet headers
  - Use arrays for sequences
  - Avoid naming objects unnecessarily
    - Saves ~80B/variable
  - Use delay_bind() rather than bind() in C++ objects
Debugging

• OTcl leaks: it does *not* garbage collect
  - set ns [new Simulator]
     for {set i 0} {$i < 500} {incr i} {
       set a [new RandomGenerator/Constant]
     }
  - This will generate ~500 objects
  - Free things yourself ((delete $a)

• Conservation
  - Avoid $ns trace-all $f
  - Remove unnecessary packet headers
  - Use arrays for sequences
  - Avoid naming objects unnecessarily
  - Use `delay_bind()` rather than `bind()` in C++ objects
    • Changes memory requirements to per-class rather than per-instance
Summary

• Background

• Usage

• Simulation
  – Model. Class hierarchy.

• Fundamentals

• Infrastructure

• Debugging
Thank You