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

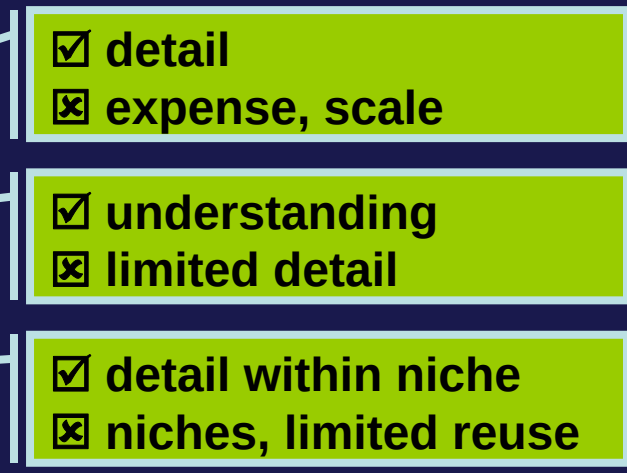
- Background
  - Links. Purpose. History. Components. Status.
- Usage
- Simulation
- Fundamentals
- Infrastructure
- Debugging

Useful links

- *ns-2* web page
  http://www.isi.edu/nsnam/ns/
- nam web page
  http://www.isi.edu/nsnam/nam/
- *ns-2* tutorial [ns-2.29/tutorial/]
  http://www.isi.edu/nsnam/ns/tutorial/index.html
- *ns-2* workshops
  http://www.isi.edu/nsnam/ns/ns-tutorial/index.html
- *ns-2* manual
  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

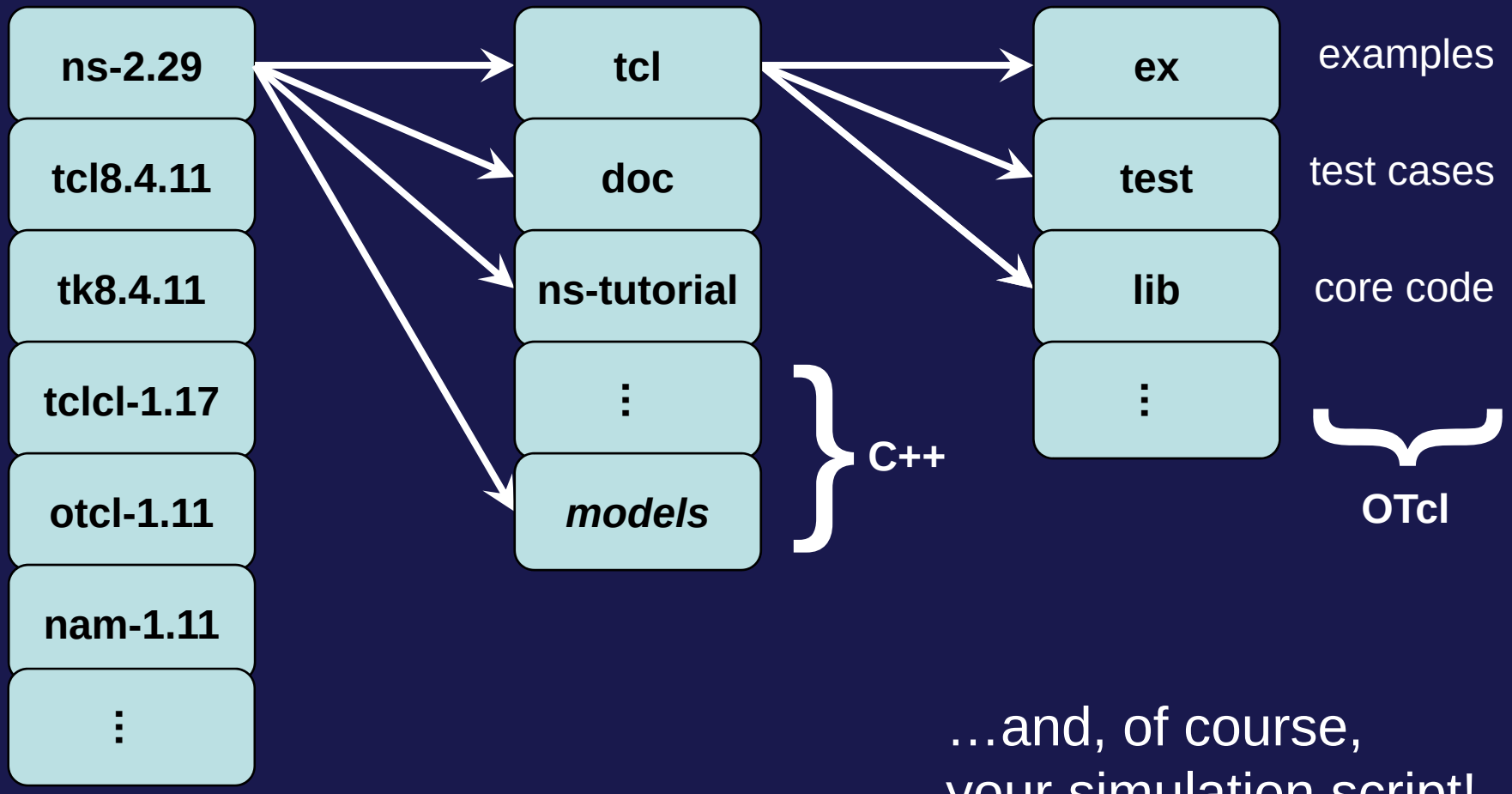


History

- 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



…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

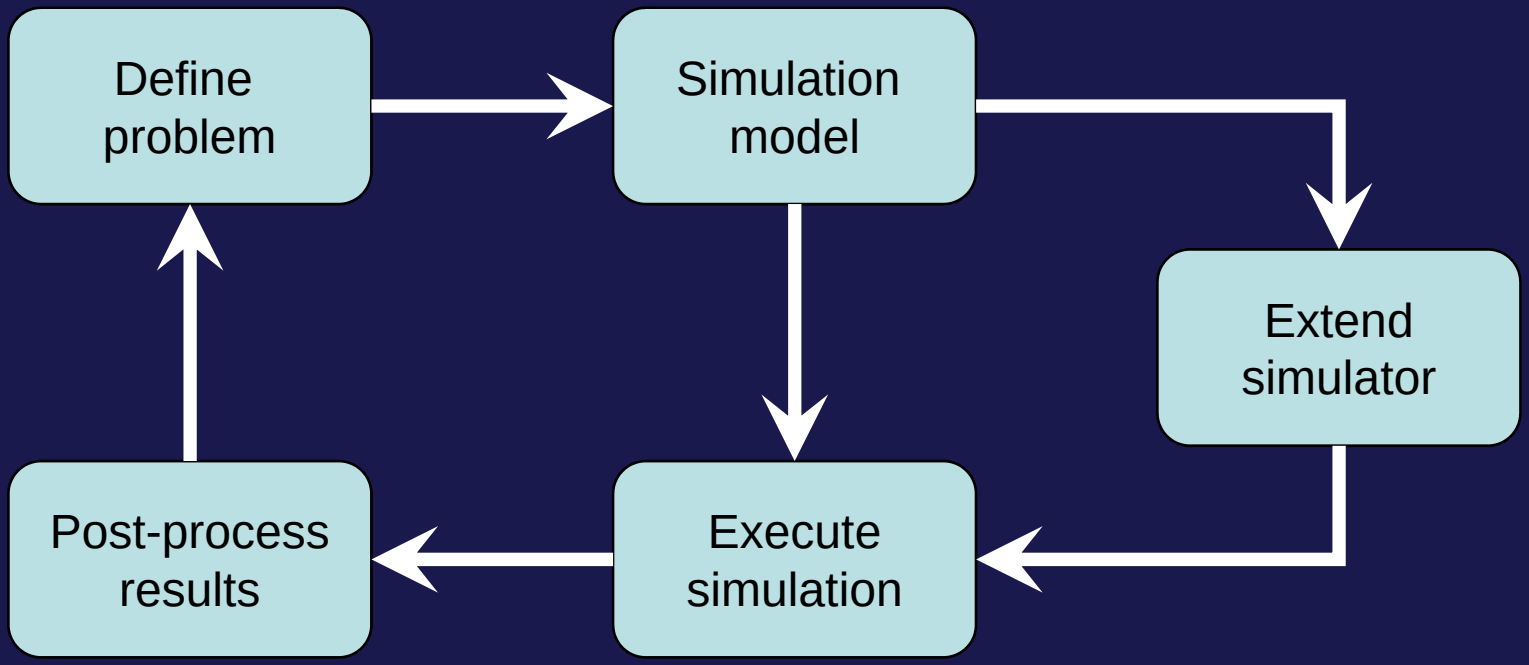
Status

- *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
  - Process. Split object model. Languages. Tcl/OTcl. Linkage.
- Simulation
- Fundamentals
- Infrastructure
- Debugging

Using *ns*


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

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 simul
  - Objects are *shared* between OTcl a
  - Access the OTcl interpreter via class
  - `Tcl &tcl = Tcl::instance();`
    `tcl.eval(…); tcl.evalc(“”); tcl.evalf(“”,…);`
    `tcl.result(…); res = tcl.result();`
  - so keeps hashtable o TclObject
    p(
- Variables are, by default, *unshared*
  - *P*
  - Bindings established by the compiled constructer
    `bind(); bind_d ay();`
    `{ 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

C++/OTcl Linkage

```
struct hdr_ping {
    char ret; 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));
}
```

Header management

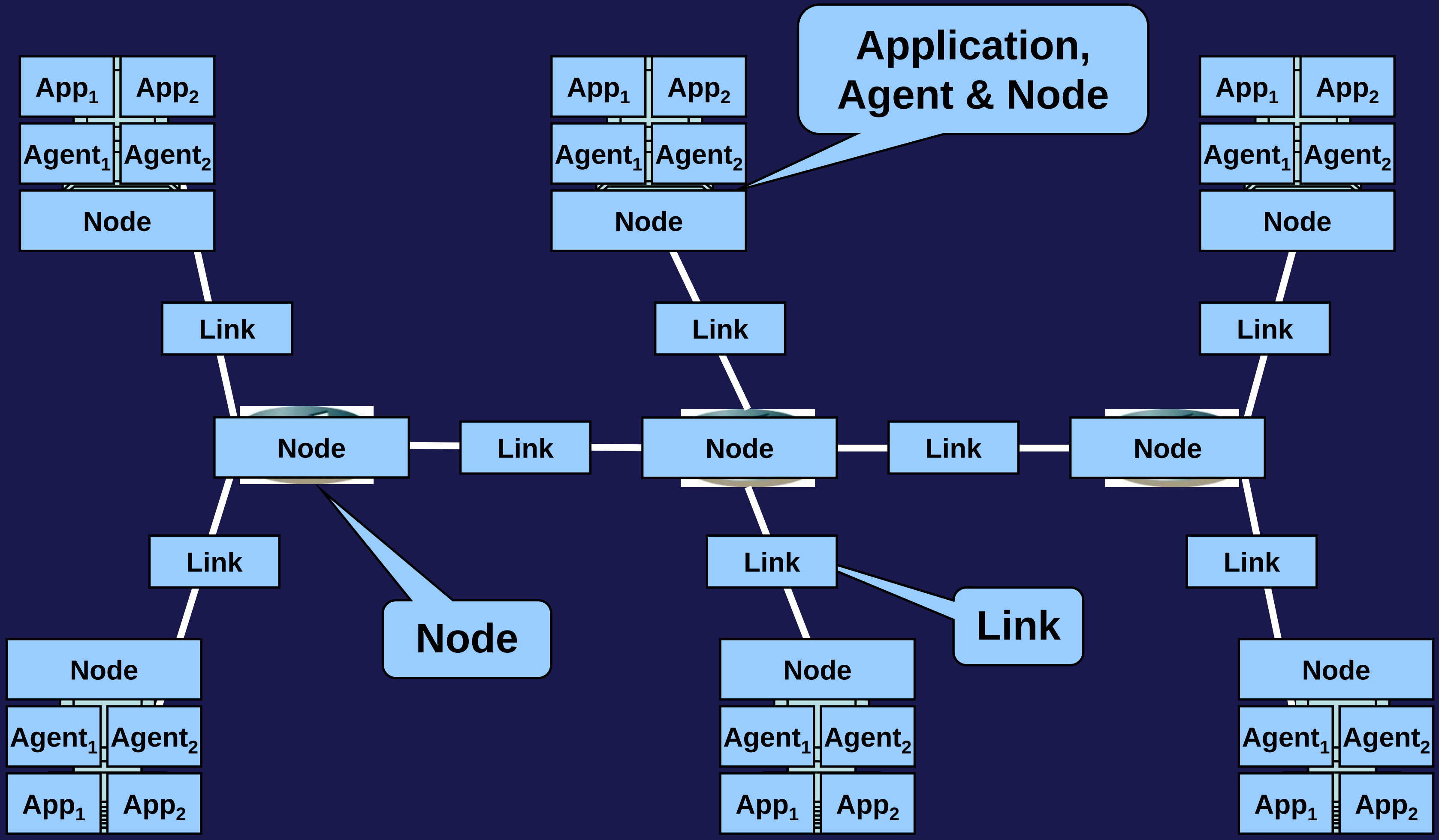
Constructor,
with bound variable

send method invoked in simulation; result returned

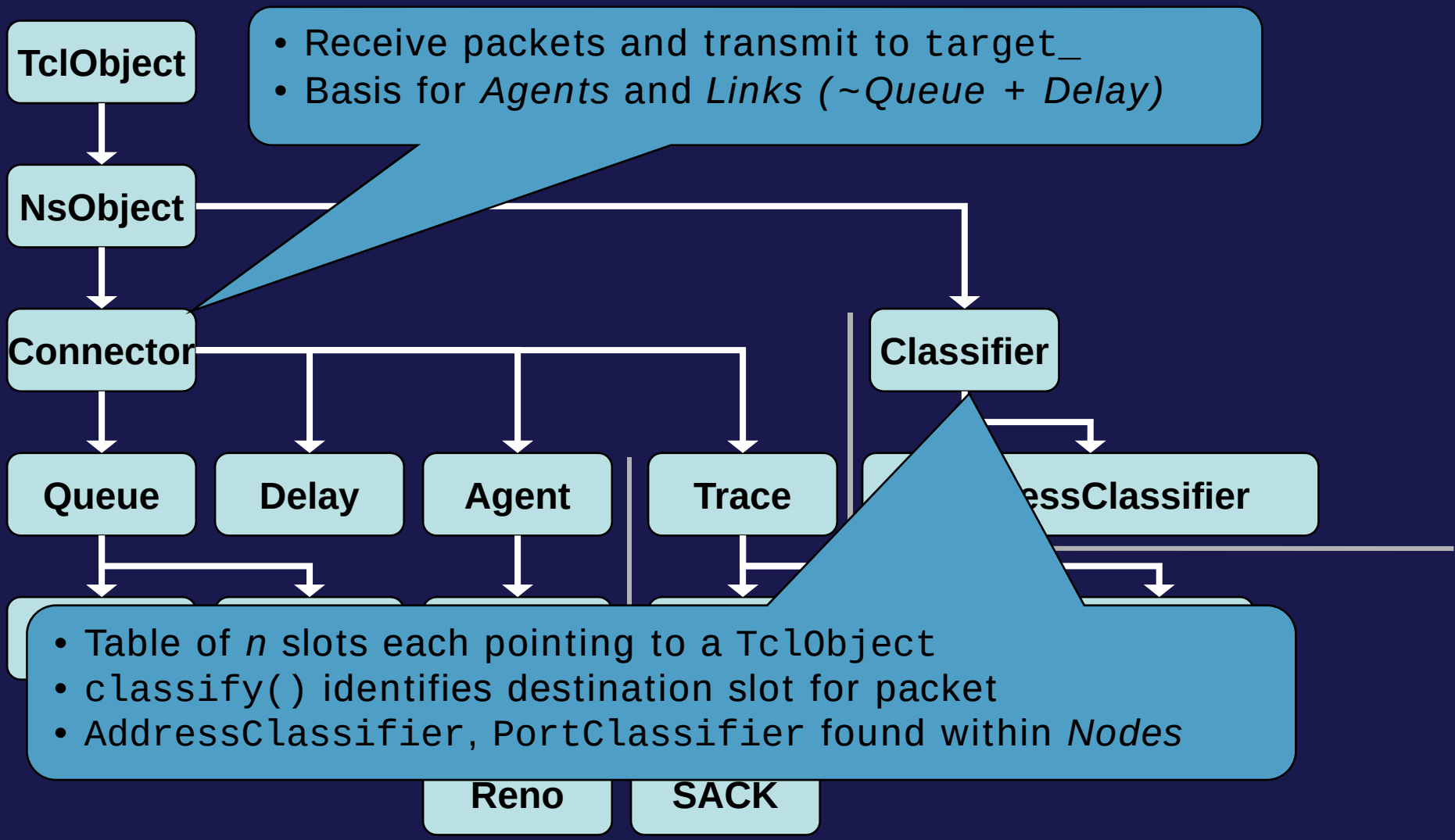
Outline

- Background
- Usage
- Simulation
  - Model. Class hierarchy.
- Fundamentals
- Infrastructure
- Debugging

From Network to Simulation



Class Hierarchy (partial)



Outline

- Background
- Usage
- Simulation
- Fundamentals
  - Applications. Agents. Nodes. Links. Packets. A simple topology.
- 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)

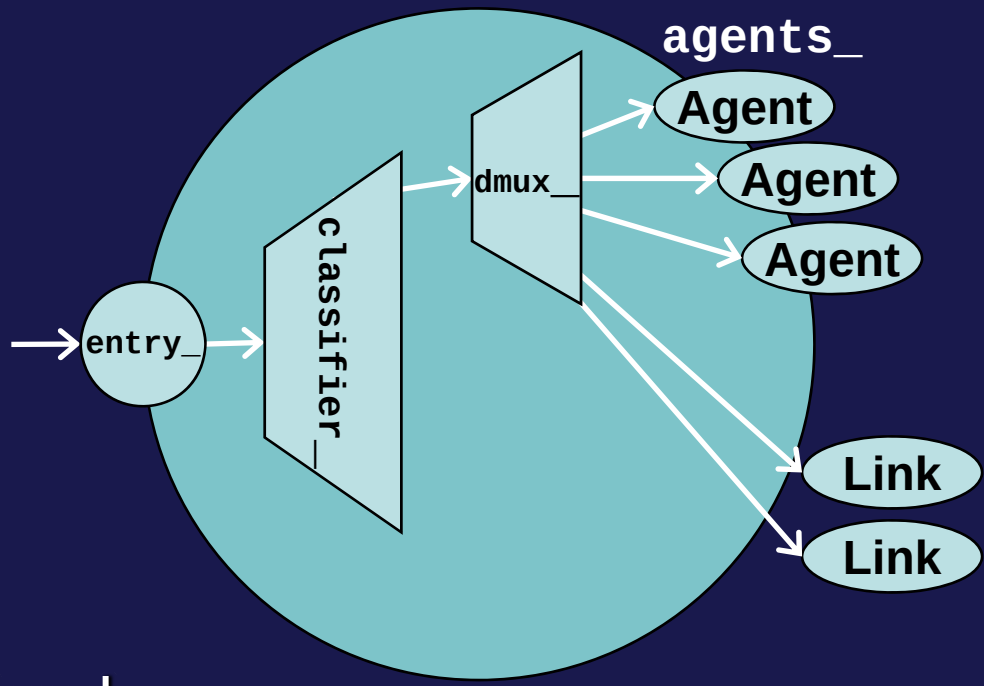
Agents

- 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

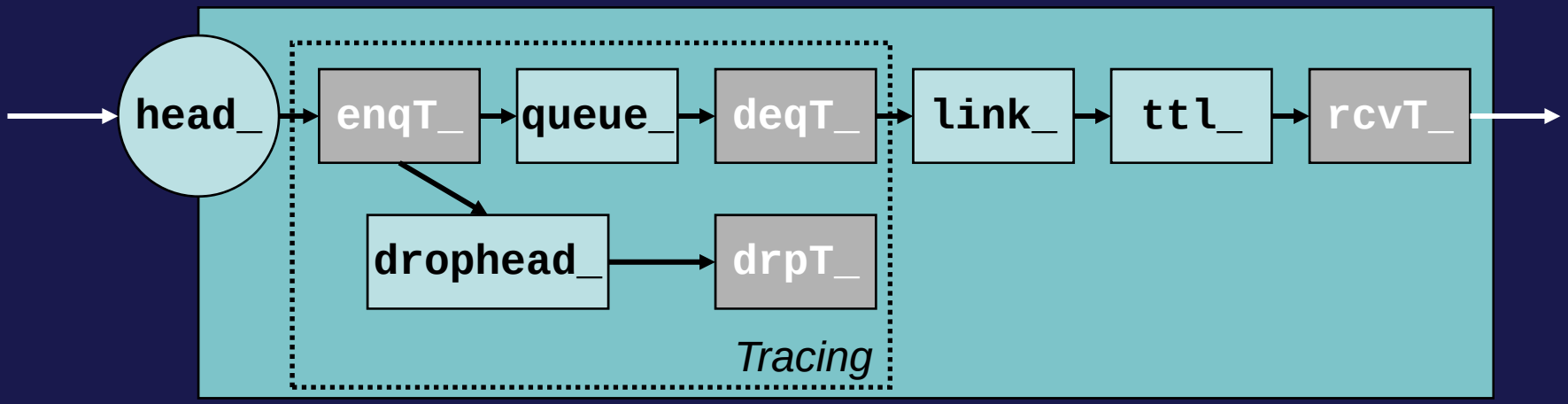
- Addressable entity built from classifiers
  - Distributes incoming data to agents
  - Distributes outgoing data to links
- Simplest unicast case has address and port classifiers, others may have more
- Node, LanNode, etc derived from ParentNode

  [ns-2.29/common/{parentnode,node}.h]

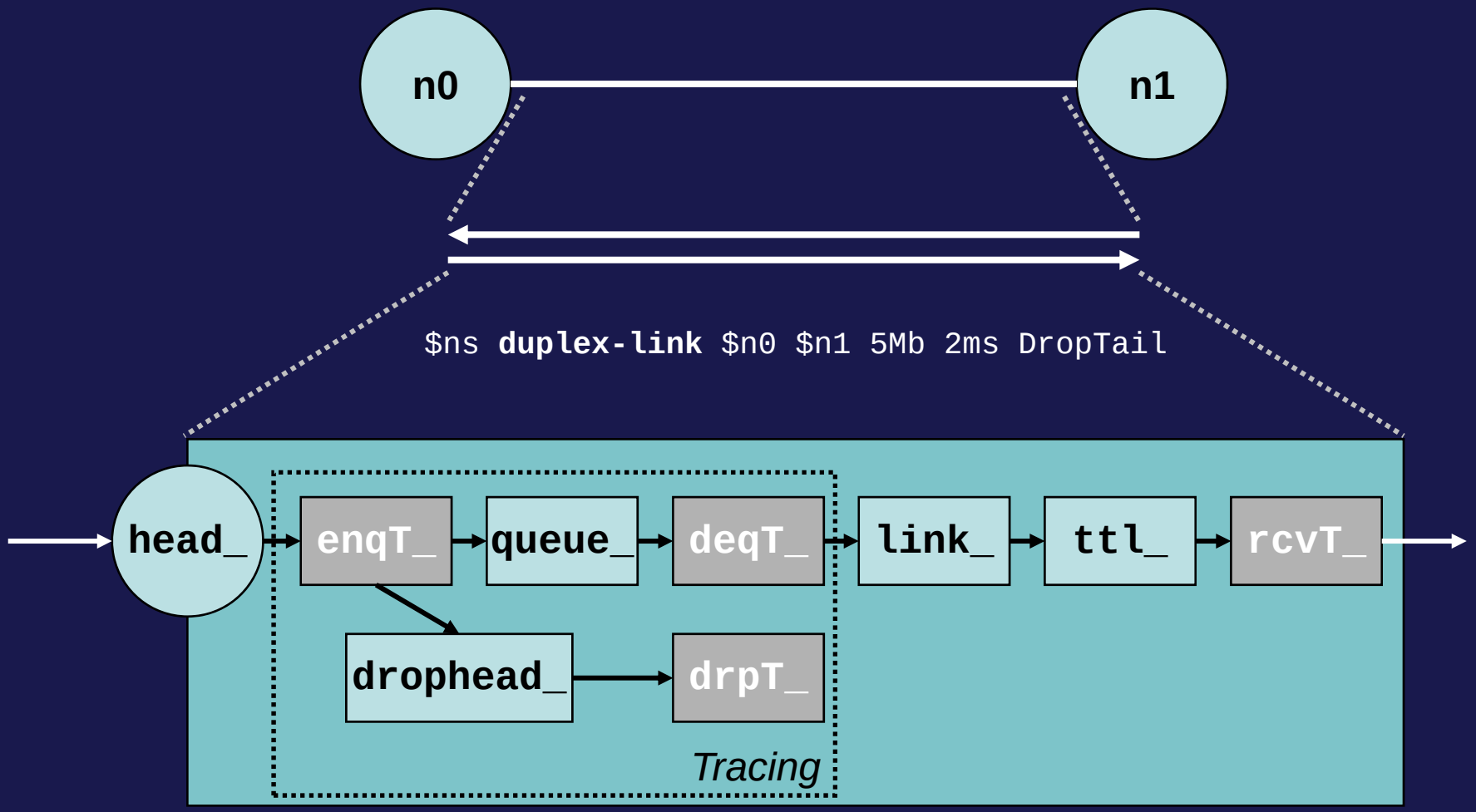


Links (Ch.6, p.62)

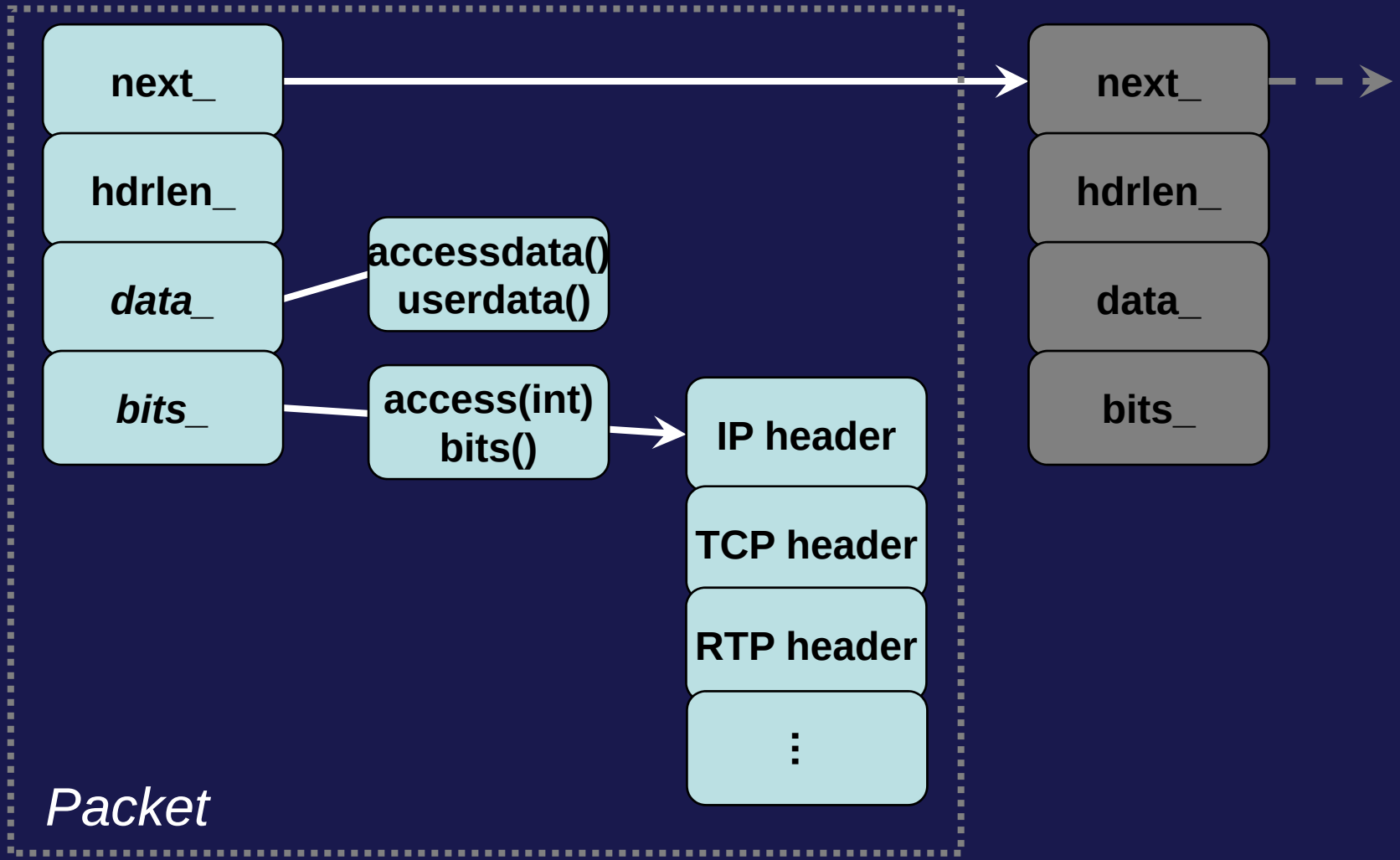
- An OTcl amalgam of C++ objects
  [ns-2.29/tcl/lib/ns-link.tcl]
  - `class **SimpleLink** –superclass Link`
  - `Simulator instproc **duplex-link** { n1 n2 bw delay type args }`
- More complex links may have complex link delay/bw characteristics…
  [ns-2.29/link/delay.h]
  - `class **LinkDelay** : public Connector`
- …and multiple queue_ elements
  [ns-2.29/queue/{queue.h,red.h,…}]
  - `class **Queue** : public Connector`
  - `class **RedQueue** : public Queue`



Links


n0
n1
$ns duplex-link $n0 $n1 5Mb 2ms DropTail
head_
enqT_
queue_
deqT_
link_
ttl_
rcvT_
drophead_
drpT_
Tracing

Packets


next_
hdrlen_
data_
bits_
accessdata()
userdata()
access(int)
bits()
IP header
TCP header
RTP header
⋮
Packet
next_
hdrlen_
data_
bits_

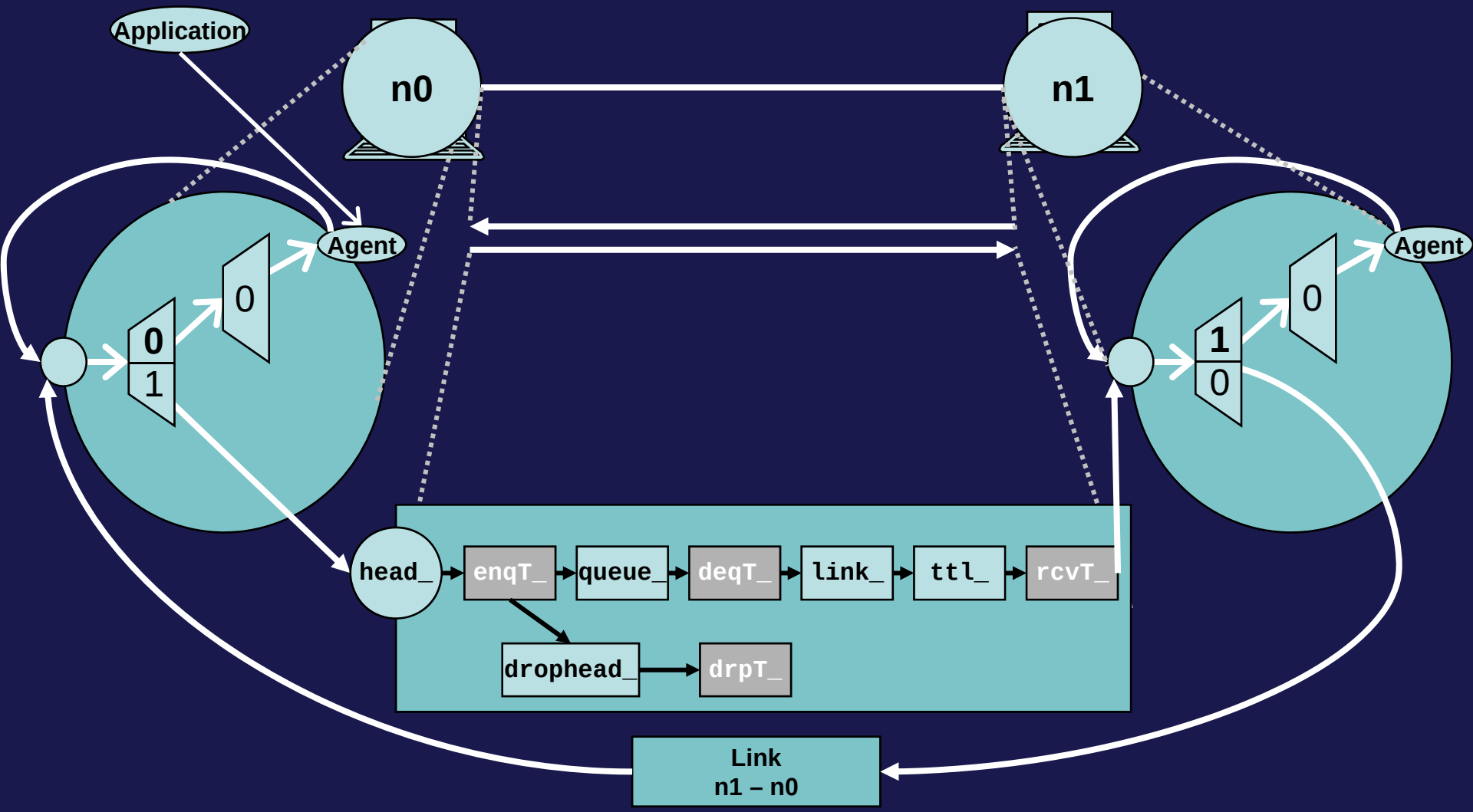
Packets

- Derived from base `class Event`
  - Other derived class is `at-event` (OTcl)
- Packets are set of headers plus data
  - » `foreach cl [ PacketHeader info subclass ] { puts "$cl [$cl set hdrlen_]" }`
  - 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)
      - » `foreach cl [PacketHeader info subclass] { puts $cl }`
      - `remove-packet-header AODV ARP …`, or
      - `remove-all-packet-headers`
        `add-packet-header IP TCP …`

```
%  foreach cl [ PacketHeader
    info subclass ] { puts "$cl
    [$cl set hdrlen_]" }
```

PacketHeader/LRWPAN 216
PacketHeader/XCP 64
PacketHeader/Lms 56
PacketHeader/PGM 16
PacketHeader/PGM_SPM 8
PacketHeader/PGM_NAK 16
PacketHeader/Pushback 4
PacketHeader/NV 8
PacketHeader/LDP 40
PacketHeader/MPLS 20
PacketHeader/rtProtoLS 8
PacketHeader/Ping 32
PacketHeader/TFRC 56
PacketHeader/TFRC_ACK 64
PacketHeader/Diffusion 192
PacketHeader/RAP 24
PacketHeader/AODV 808
PacketHeader/SR 720
PacketHeader/TORA 32
PacketHeader/IMEP 512
PacketHeader/ARP 32
PacketHeader/MIP 32
PacketHeader/IPinIP 4
PacketHeader/LL 32
PacketHeader/Mac 40
PacketHeader/Encap 4
PacketHeader/HttpInval 4
PacketHeader/MFTP 64
PacketHeader/SRMEXT 8
PacketHeader/SRM 16
PacketHeader/aSRM 8
PacketHeader/mcastCtrl 20
PacketHeader/CtrMcast 12
PacketHeader/rtProtoDV 4
PacketHeader/GAF 8
PacketHeader/Snoop 24
PacketHeader/SCTP 8
PacketHeader/TCPA 16
PacketHeader/TCP 80
PacketHeader/IVS 32
PacketHeader/RTP 12
PacketHeader/Message 64
PacketHeader/Resv 16
PacketHeader/TCP_QS 12
PacketHeader/UMP 16
PacketHeader/Src_rt 76
PacketHeader/IP 28
PacketHeader/Common 104
PacketHeader/Flags 9

A Simple Topology



Outline

- Background
- Usage
- Simulation
- Fundamentals
- Infrastructure
  - Addressing. Routing. Dynamics. Maths. Tracing.
- Debugging

Addressing

- Two modes: *default* and *hierarchical*
  [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:
      `$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

Routing (Ch.27, p.243; Ch.30, p.274)

- Links have assigned weights
  - Default to 1
    - `$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 *[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
  - MRG32k3a proposed in P. L'Ecuyer, "**Good parameters and implementations for combined multiple recursive random number generators**," *Operations Research, 47(1):159—164, 1999.*
    - » (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

op	ts	prevhop	nexthop	type	size	flags	flowid	src	dst	seqno	pktid
+	1.102	0	1	tcp	40	------	0	0.0	1.0	1	1

- 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 *dmalloc*

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`
    - Allocates trace objects on all links, 14kB/link
  - Remove unnecessary packet headers
  - 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
    - 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
  - Links. Purpose. History. Components. Status.
- Usage
  - Process. Split object model. Languages. Tcl/OTcl. Linkage.
- Simulation
  - Model. Class hierarchy.
- Fundamentals
  - Applications. Agents. Nodes. Links. Packets. A simple topology.
- Infrastructure
  - Addressing. Routing. Dynamics. Maths. Tracing.
- Debugging

Thank You