ns v2 Workshop

Kannan Varadhan

USC/Information Sciences Institute

⟨kannan@catarina.usc.edu⟩

18 September, 1997

The work of K. Varadhan and the VINT project at USC/ISI is supported by the Defense Advanced Research Projects Agency (DARPA) under contract number ABT63-96-C-0054. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA.

Outline

1. Topology Generation, the nodes and the links

2. OTcl and C++: The Duality

3. Routing

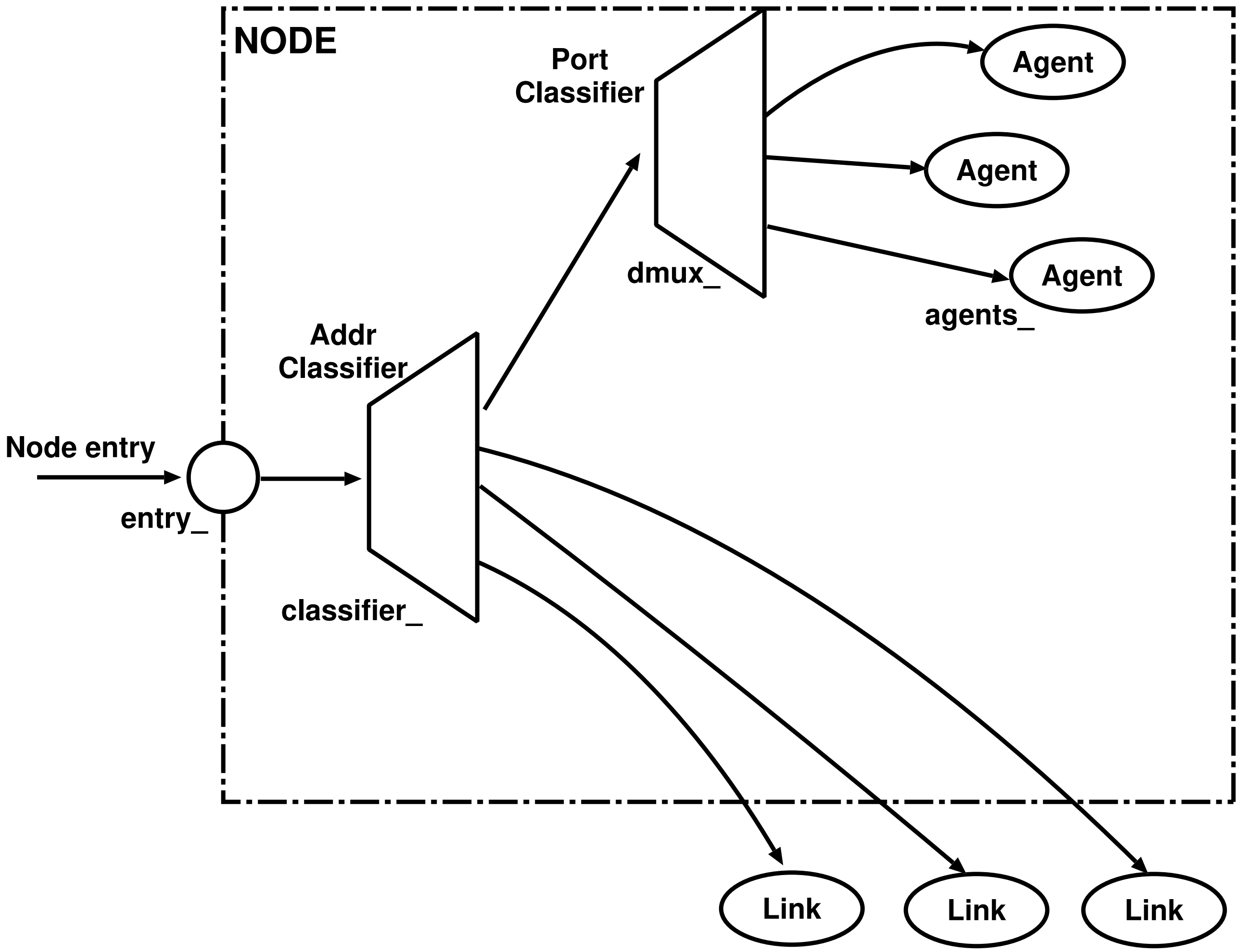
   - Unicast
   - Multicast
   - Network Dynamics

4. Multicast Transport

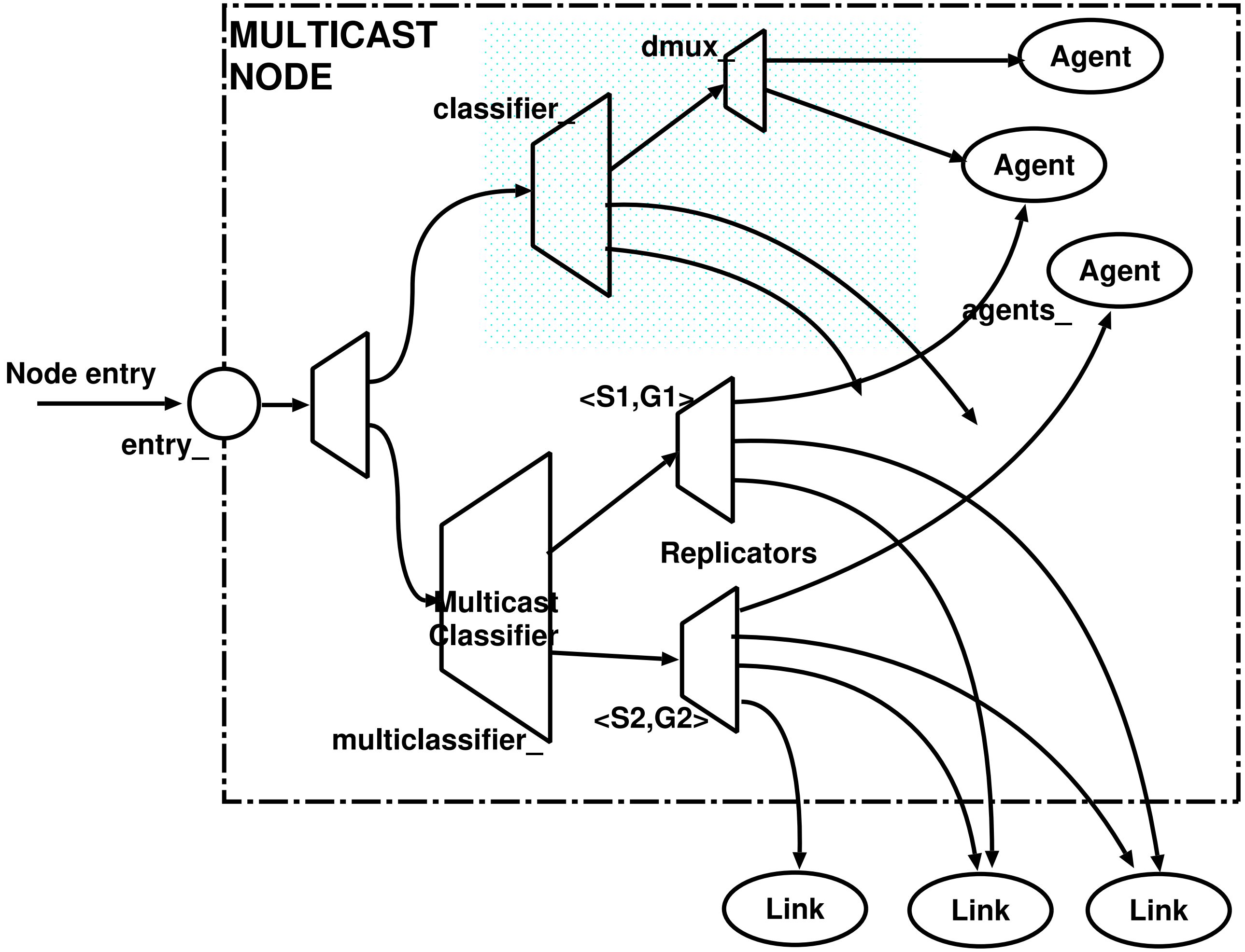
5. Issues of Scale

6. Programming and Simulation Debugging

Nodes


NODE
Port
Classifier
Agent
Agent
Agent
dmux_
agents_
Addr
Classifier
Node entry
entry_
classifier_
Link
Link
Link

Multicast Nodes



Classifiers

- Table of n slots
- Each slot can point to a TclObject
- When a packet is received

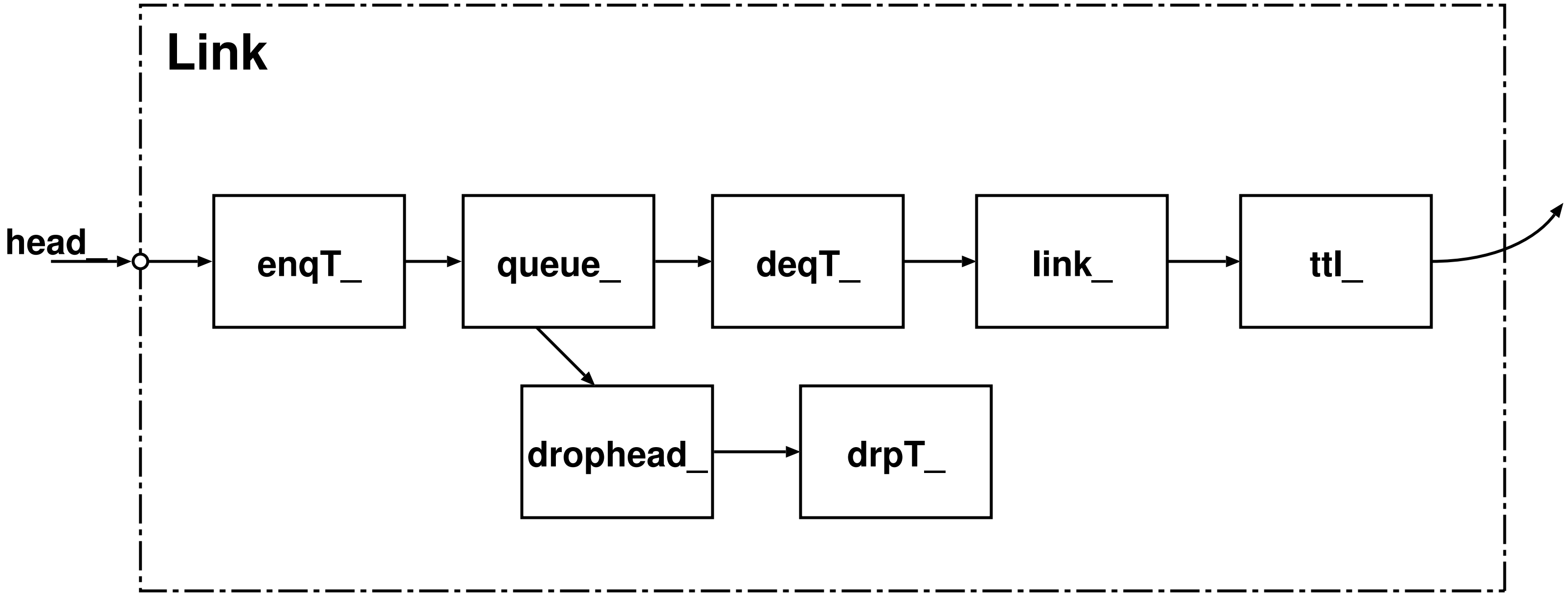
  — `classify`() identifies the slot to forward the packet to
- If slot is invalid, the classifier calls `no-slot`{}
- Many different types of classifiers

Address Classifiers	parse address in packet
MultiPath Classifier	returns next slot number to use
Replicator	uses classifier as a table

Classifier methods

- Install entries into classifier
  - `install{}`
  - `installNext{}`
- Query entries in classifier
  - `elements{}` returns current list of elements inserted
  - `slot{}` returns handle of object in the specified slot
- Delete entry in a particular slot
  - `clear{}`
- `classify()` internal method: receives a packet, and returns a slot number for that packet.

Links



Connectors

- Connectors receive incoming packets, and (usually) transmit them to their `target_`
- Many different types of connectors:

Queue	holds a certain number of packets. Packets exceeding their queue-size are sent to the queue's drop-target.
LinkDelay	models delay/bandwidth of the link for detailed simulations.
TTLChecker	decrements TTLs on each packet, drops the packet if the TTL becomes zero.
DynaLink	transmit packets if the link is up, drop packet otherwise
	Other tracing related objects

Connector methods

- Add tracing or monitoring:
  - `trace`
  - `attach-monitors`
  - `init-monitor`

Topology Generation Resources

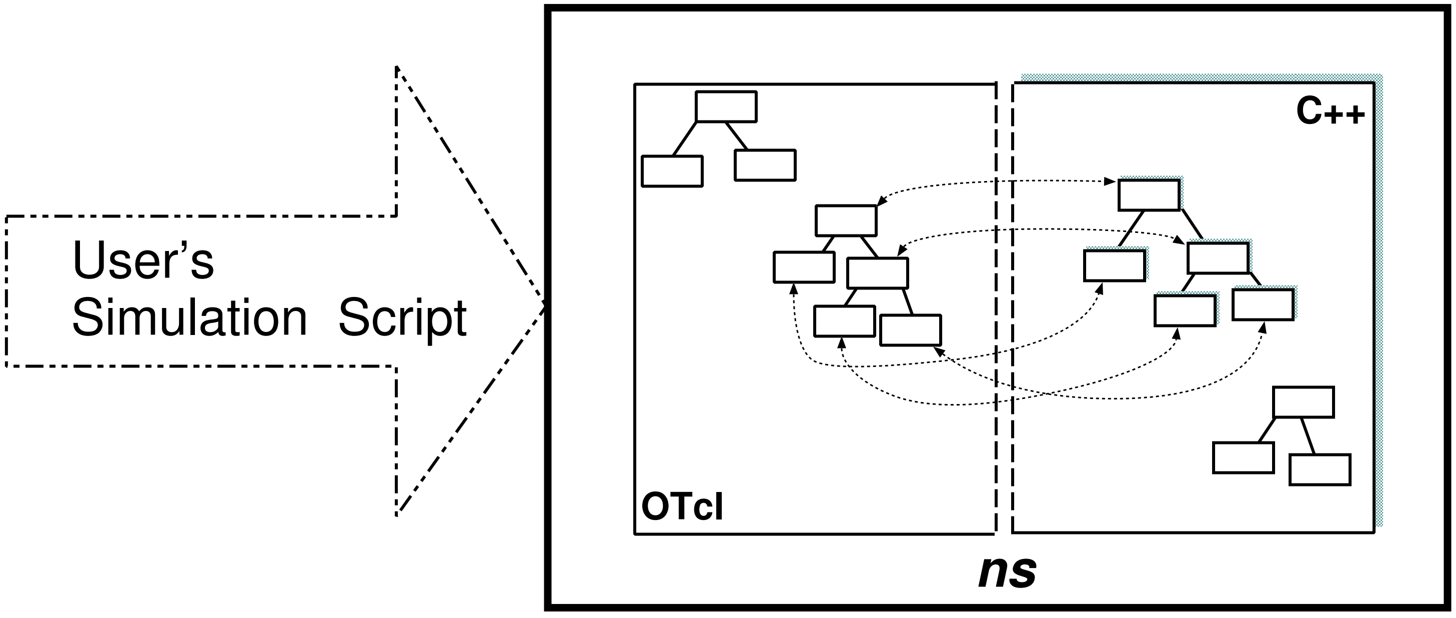
At `http://netweb.usc.edu/daniel/research/sims/`

- `ntg` by Steve Hotz ⟨hotz@isi.edu⟩
- `GT-ITM` by Ellen Zegura ⟨ewz@cc.gatech.edu⟩, Ken Calvert ⟨calvert@cc.gatech.edu⟩
- `TIERS` by Matthew Doar ⟨mdoar@nexen.com⟩
- `rtg` by Liming Wei ⟨lwei@cisco.com⟩, Lee Breslau ⟨breslau@parc.xerox.com⟩

Topology Generation

	Type of Graph	Edge Models
`ntg`	n-level hierarchy	user configurable probability distributions
`GT-ITM`	flat random, n-level hierarchies, transit-stub networks	many different edge models
`TIERS`	3-level hierarchy	Minimum spanning tree + random placement
`rtg`	flat random	waxman

OTcl and C++: The Duality



OTcl Linkage

C++ $\Longleftrightarrow$ OTcl

class Tcl	C++ methods to access the OTcl interpreter
class TclCommand	Basic script to provide limited global commands to the interpreter
class EmbeddedTcl	Container for Tcl scripts that are pre-loaded at startup
class TclObject	Root of basic object hierarchy in *ns*
class TclClass	C++ class to set up the TclObject hierarchy
class InstVar	internal class to bind C++ member variables to OTcl instance variables

C++ Methods to Access OTcl

The class Tcl

- Obtain a handle
- Invoke OTcl procedures
- Retrieve the result from the interpreter
- On invocation, pass a result string to the interpreter
- Return Success/Failure return codes

class Tcl: C++ Methods to access OTcl

```
Tcl& tcl = Tcl::instance();                          /* obtain a handle to the interpreter */
if (argc == 2) {                                     /* cmd: foo now */
  if (strcmp(argv[1], "now") == 0) {
    tcl.resultf("%g", clock());                      /* pass back the result */
    return TCL_OK;                                   /* return success code to interpreter */
  }
  tcl.result("command not understood");              /* alternate way of passing result */
  return TCL_ERROR;
} else if (argc == 3) {                              /* cmd: foo now ⟨callback⟩ */
  if (strcmp(argv[1], "now") != 0) {
    tcl.error("command not understood");             /* alternate way to return error */
  }
  char *callback = arv[2];
  tcl.eval(callback);                                /* invoke an OTcl procedure */
  tcl.evalc("puts hello, world");                    /* another variant */
  char* timestr = tcl.result();                      /* callback result from the interpreter */
  clock() = atof(timestr);
} else {
  Interp* ti = tcl.interp();                         /* access the interpreter directly */
  ...                                                /* ... to do whatever */
```

class TclCommand

Defines simple commands that execute in global interpreter context

For example, `ns-version`

class EmbeddedTcl: Adding new OTcl code into *ns*

- container for scripts pre-loaded at startup
  - ~*Tcl*/tcl-object.tcl
  - ~*ns*/tcl/lib/ns-lib.tcl
  - scripts recursively sourced by ~*ns*/tcl/lib/ns-lib.tcl

- ~*Tcl*/tcl-object.tcl activated by `Tcl::init`()

  ~*ns*/tcl/lib/ns-lib.tcl activated by `Tcl_AppInit`()

class TclObject

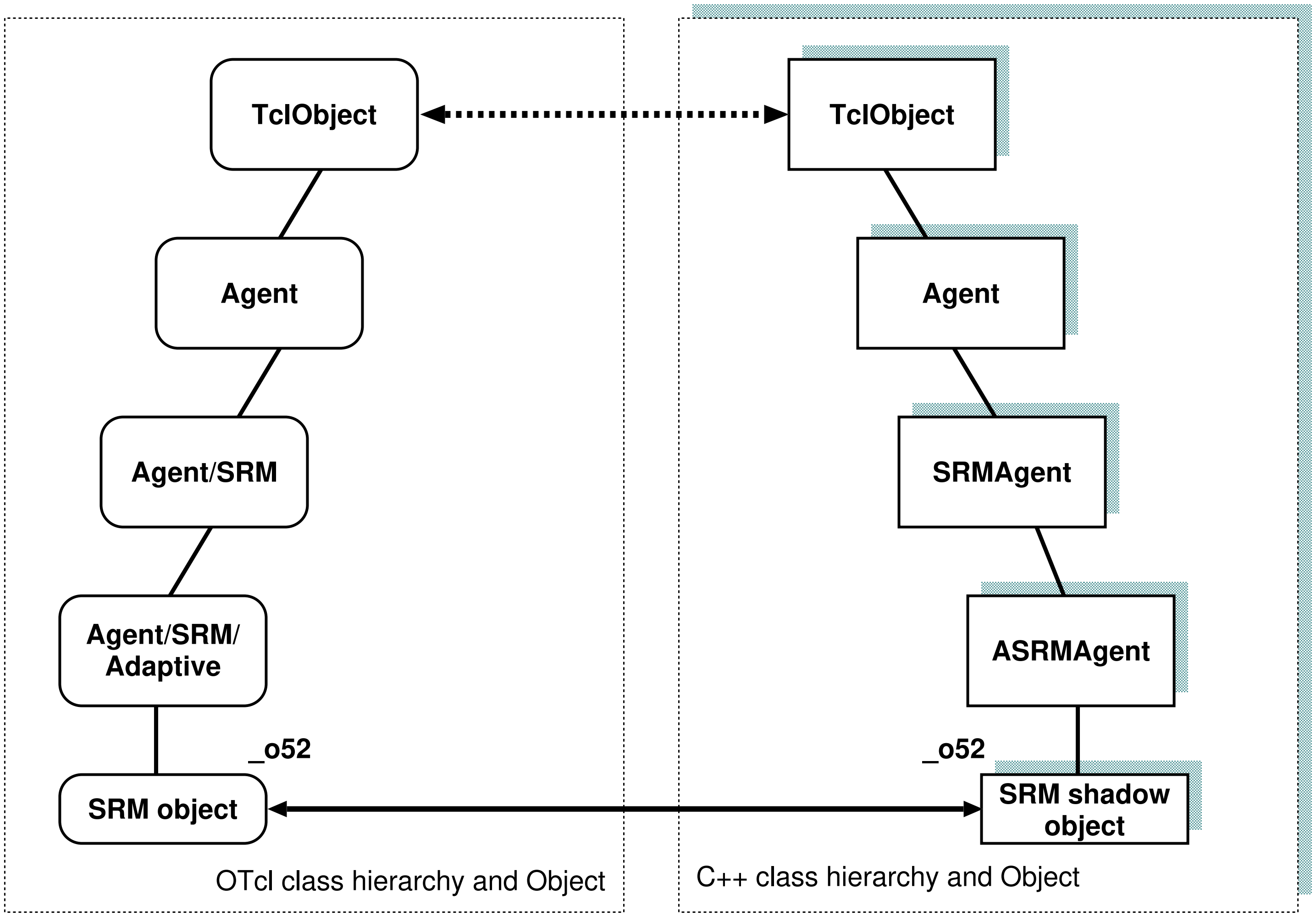
- Basic object hierarchy in *ns*

- Hierarchy mirrored in C++ and OTcl

- For example:

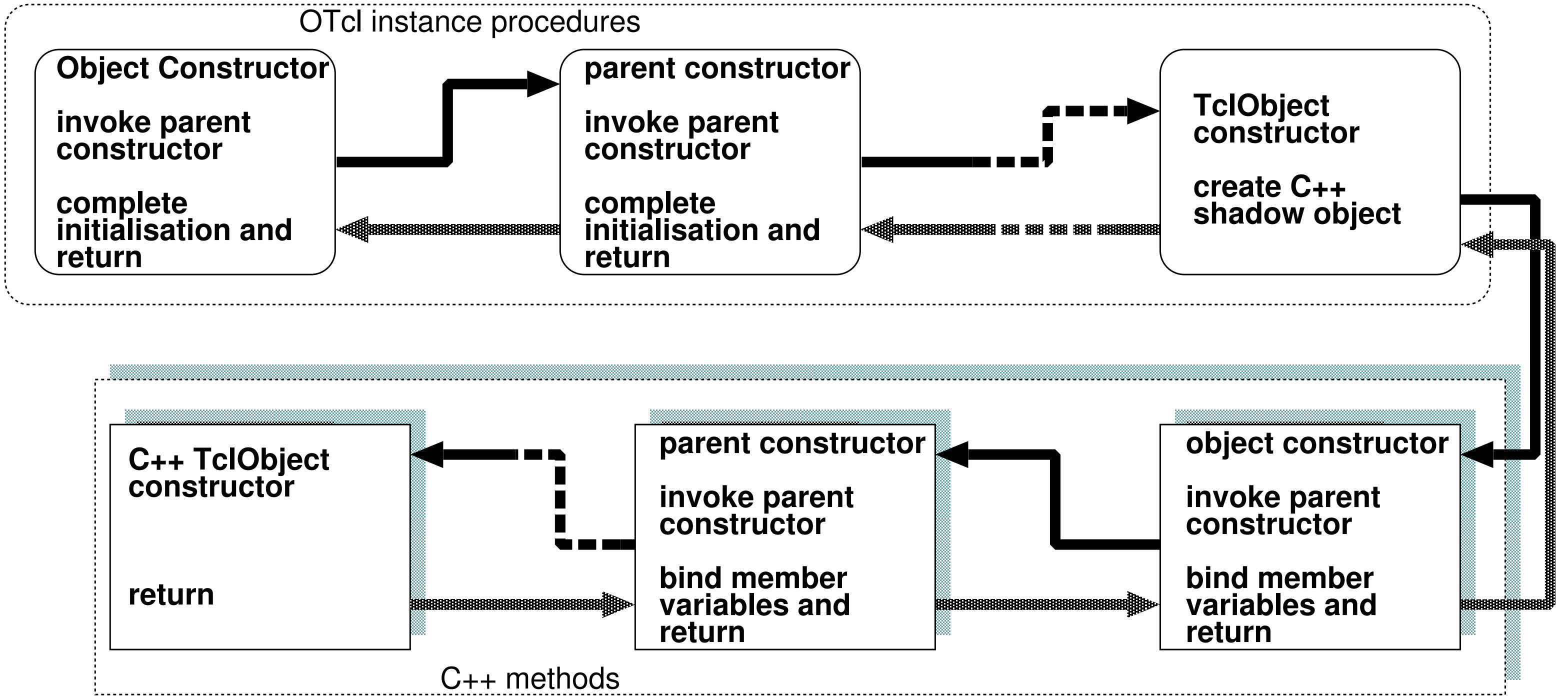
```
set srm [new Agent/SRM/Adaptive]
$srm set packetSize_ 1024
$srm traffic-source $s0
```

class TclObject: Hierarchy and Shadowing



class TclObject: Creation/Deletion Mechanisms

- global procedure `new{}`



- global procedure `delete{}`

class TclObject: Binding Variables

- makes identical C++ member variables to OTcl instance variables
- Syntax

```
ASRMAgent::ASRMAgent() {
  bind("pdistance_", &pdistance_);                    /* real variable */
  ...
}
```

- Initialisation through OTcl class variables

```
  Agent/SRM/Adaptive set pdistance_ 15.0
  Agent/SRM set pdistance_ 10.0
  ...
```

- Other methods: `bind()` (integers), `bind_time()` (time variables), `bind_bw()` (bandwidth variables), `bind_bool()` (boolean variables)

Examples of Specify Bound variables

```
$object set bwvar 1.5m
$object set bwvar 1.5mb
$object set bwvar 1500k
$object set bwvar 1500kb
$object set bwvar .1875MB
$object set bwvar 187.5kB
$object set bwvar 1.5e6

$object set timevar 1500m
$object set timevar 1.5
$object set timevar 1.5e9n
$object set timevar 1500e9p

$object set boolvar t                 ; # set to true
$object set boolvar true
$object set boolvar 1                 ; # or any non-zero value

$object set boolvar false             ; # set to false
$object set boolvar junk
$object set boolvar 0
```

class TclObject: `command()` methods

- shadow object is accessed by a `cmd{}` procedure, called "**instproc-like**"
- For example, `distance?{}` is an "instance procedure" of an Adaptive SRM agent

```
int ASRMAgent::command(int argc, const char*const*argv) (1) {
    Tcl& tcl = Tcl::instance();
    if (argc == 3) {
        if (strcmp(argv[1], "distance?") == 0) (2) {
            int sender = atoi(argv[2]);
            SRMinfo* sp = get_state(sender);
            tcl.tesultf("%f", sp->distance_);
            return TCL_OK; (3)
        }
    }
    return (SRMAgent::command(argc, argv)); (4)
}
```

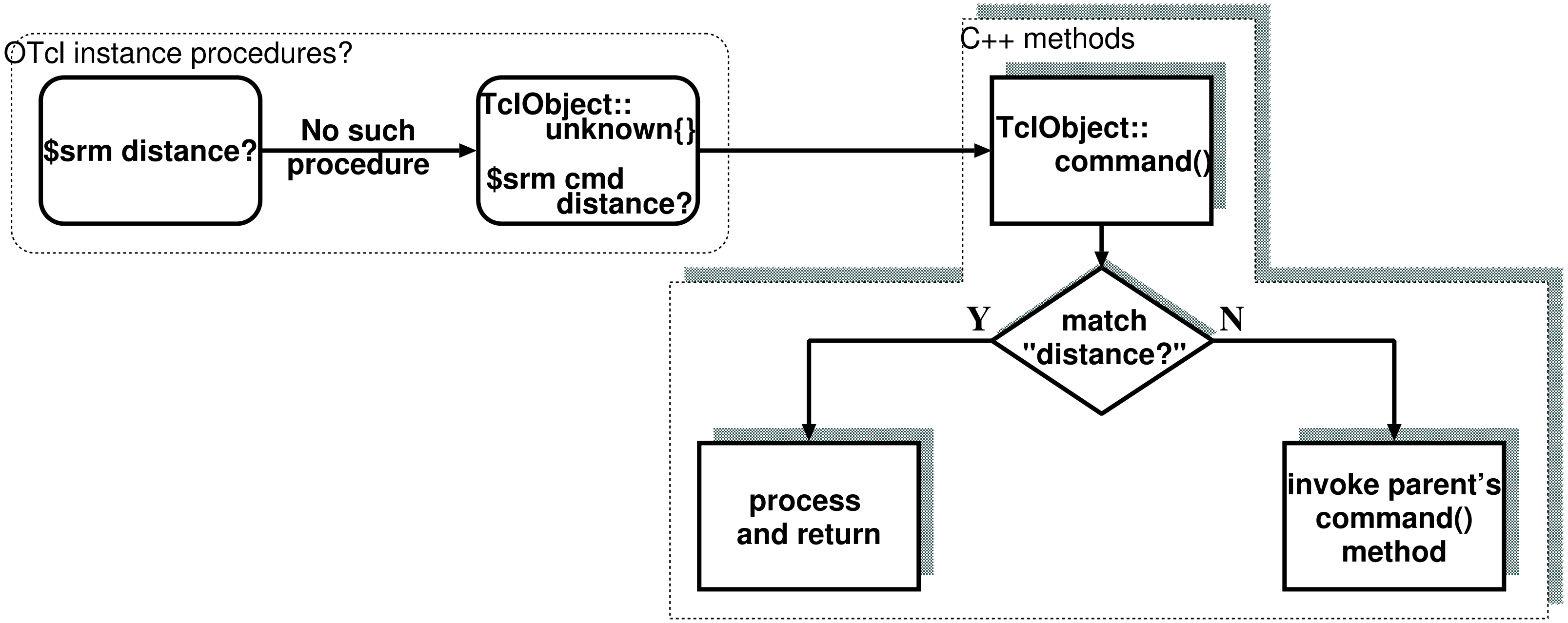
class TclObject: `command()` methods: call sequence

- Usage:

```
$srm distance?                ; # instproc-like usage
```

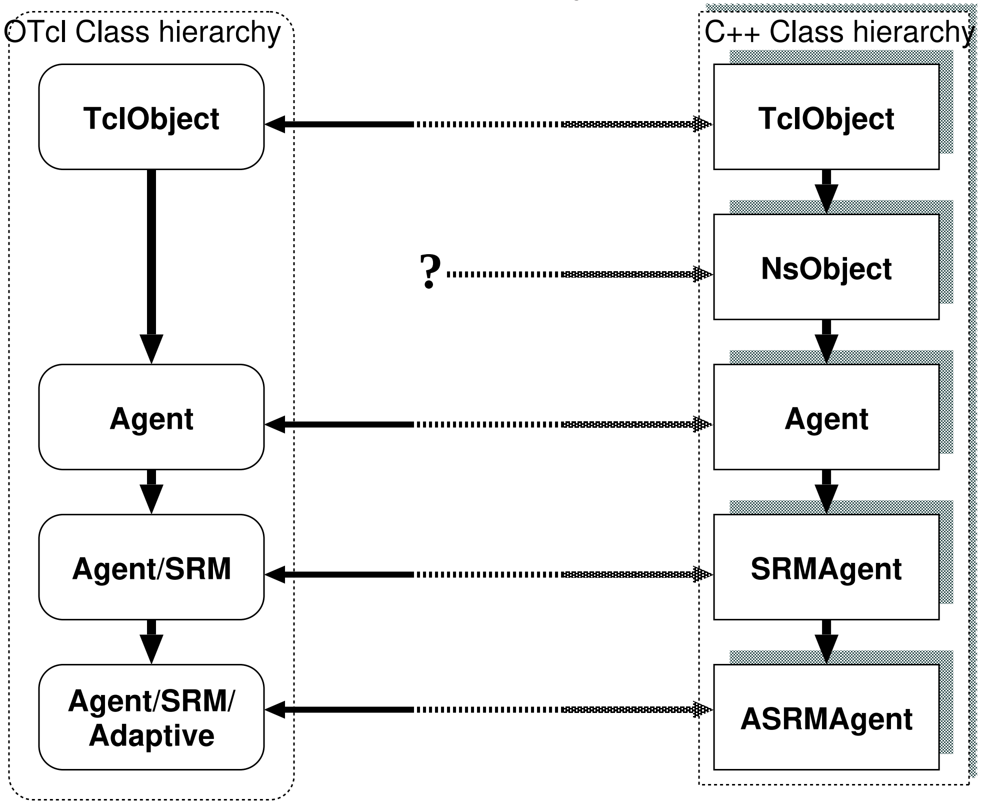
or

```
$srm cmd distance?            ; # explicit usage
```



class TclClass

Programmer defines C++ hierarchy that is mirrored in OTcl



not all classes are mirrored exactly

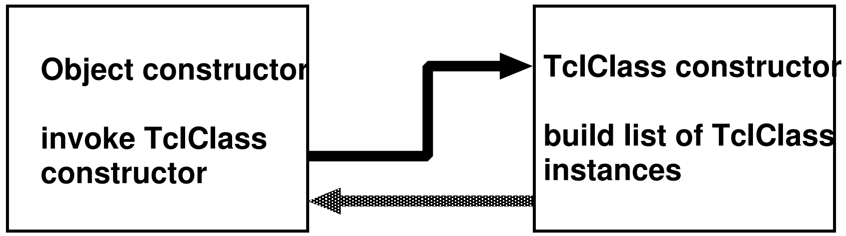
Class TclClass: Definition

- For example, Adaptive SRM agent class in C++ is mirrored into Agent/SRM/Adaptive

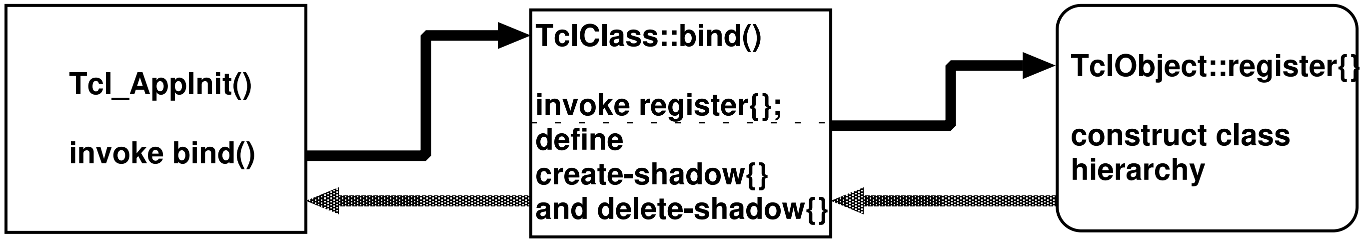
```
static class AdaptiveSRMAgentClass : public TclClass(1) {
public:
  AdaptiveSRMAgentClass() : TclClass("Agent/SRM/Adaptive")(2) {}
  TclObject* create(int /*argc*/, const char*const* /*argv*/)(3) {
    return (new AdaptiveSRMAgent())(4);
  }
} AdaptiveSRMAgentInstance(5);
```

Class TclClass: Mechanism

- static initialisation by compiler



- run time activation at startup



class Instvar

- One object per bound variable
- created by `TclObject::bind()` call
- Constructor activity
  1. point to C++ member variable
  2. create instance variable for interpreted object
  3. enable trap read/writes to instance variable using `Tcl_TraceVar()`

OTcl Linkage Summary

- Class Tcl
  - primitives to access the interpreter
- Class TclObject: root object for mirrored hierarchies
  - Unifies interpreted and compiled hierarchy
  - Provide seamless access to *ns* objects in compiled code and interpreted scripts
- Class TclClass: class that sets up the interpreted hierarchy
  - establish interpreted hierarchy
  - shadowing methods

Unicast Routing

Compute the forwarding tables at each node in topology

- specify

```
$ns rtproto ⟨protocol⟩ ⟨nodelist⟩
```

  - run protocol on nodes listed
  - ⟨protocol⟩: static, session, DV — default: static
  - ⟨nodelist⟩: — default: entire topology

- Tailor behaviour

```
$ns cost $n1 $n2 5
```

  - assign costs to links — default: cost = 1

Centralized Unicast Routing

Route computation is external to simulation execution

- Supported strategies
  - Static Routing: Routes precomputed prior to start of simulation
  - Session Routing: Static + Routes recomputed on topology change
- Dijkstra's All-pairs SPF algorithm

Detailed Unicast Routing

Route computation is part of simulation execution

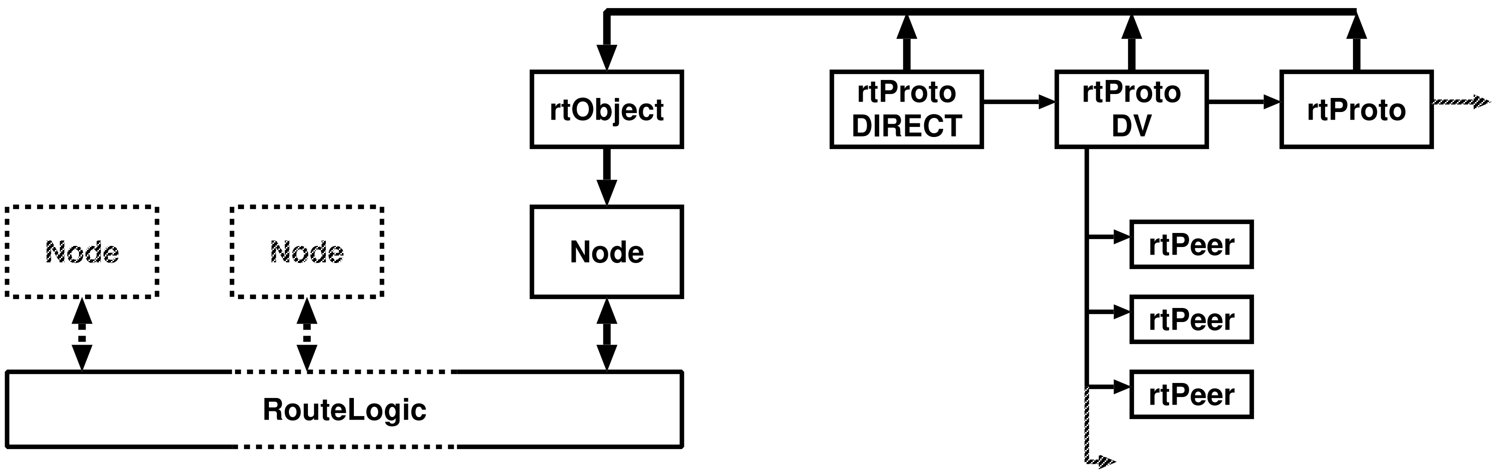
- Currently implemented protocols

  Distributed Bellman-Ford (**DV**) routing

  - `advertInterval` = $2s$. update interval
  - Split-horizon w/poison reverse advertisments
  - triggered updates on topology change, or new route entries

- Possible options such as equal cost multi-path routing

Class Architecture



class RouteLogic

- Route Configuration

register{}	configure{}	compute-routes{}
invoked by rtproto{}	invokes $rtproto init-all	centralised route compute algorithm

- Query Node n_1's nexthop to Node n_2

```
[$ns get-routelogic] lookup $n1 $n2
```

- Reconfiguration on Topology Changes

```
[$ns get-routelogic] notify
```

Dynamic Routing: class rtObject

Route Controller

`init-all{}`	Creates `rtObject` at each node in argument
`add-proto{}`	Adds ⟨protocol⟩ agent to ⟨node⟩
`lookup{}`	Returns nexthop for ⟨dest⟩ handle, -1 if none available
`compute-routes{}`	compute and install best route to destinations; invoke `send-updates{}`, `flag-multicast{}`
`intf-changed{}`	notify protocol agents; recompute-routes
`dump-routes{}`	to ⟨filehandle⟩ specified

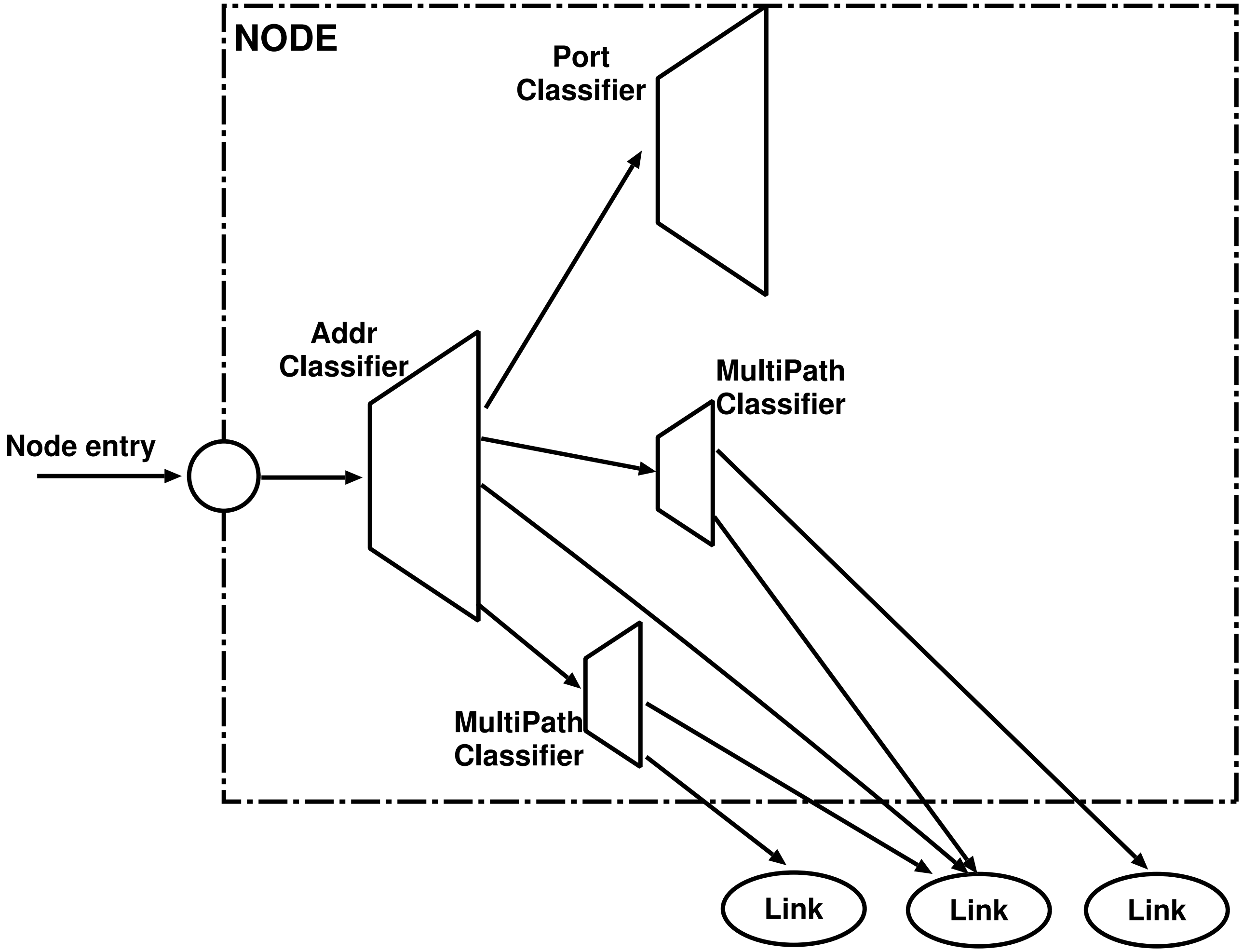
Dynamic Routing: class Agent/rtProto/DV

Route Protocol Agent

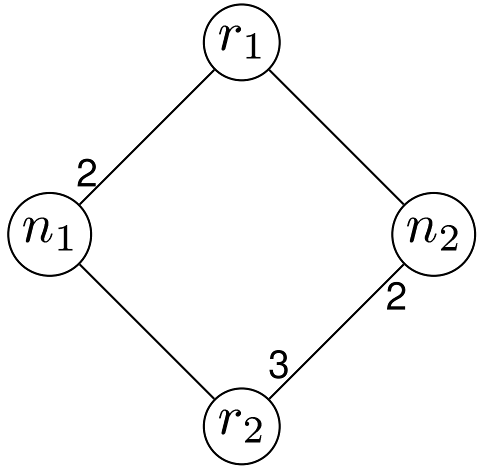
`init-all{}`	create protocol agent at each node in argument
`compute-routes{}`	invoked on startup and after topology change; compute best route in protocol; possibly set `rtsChanged_`
`intf-changed{}`	called after topology change on incident node
`send-updates{}`	whenever routes at node change

Equal Cost Multi-Path Routes

```
Node set multiPath_ 1
```



Asymetric Path Routing

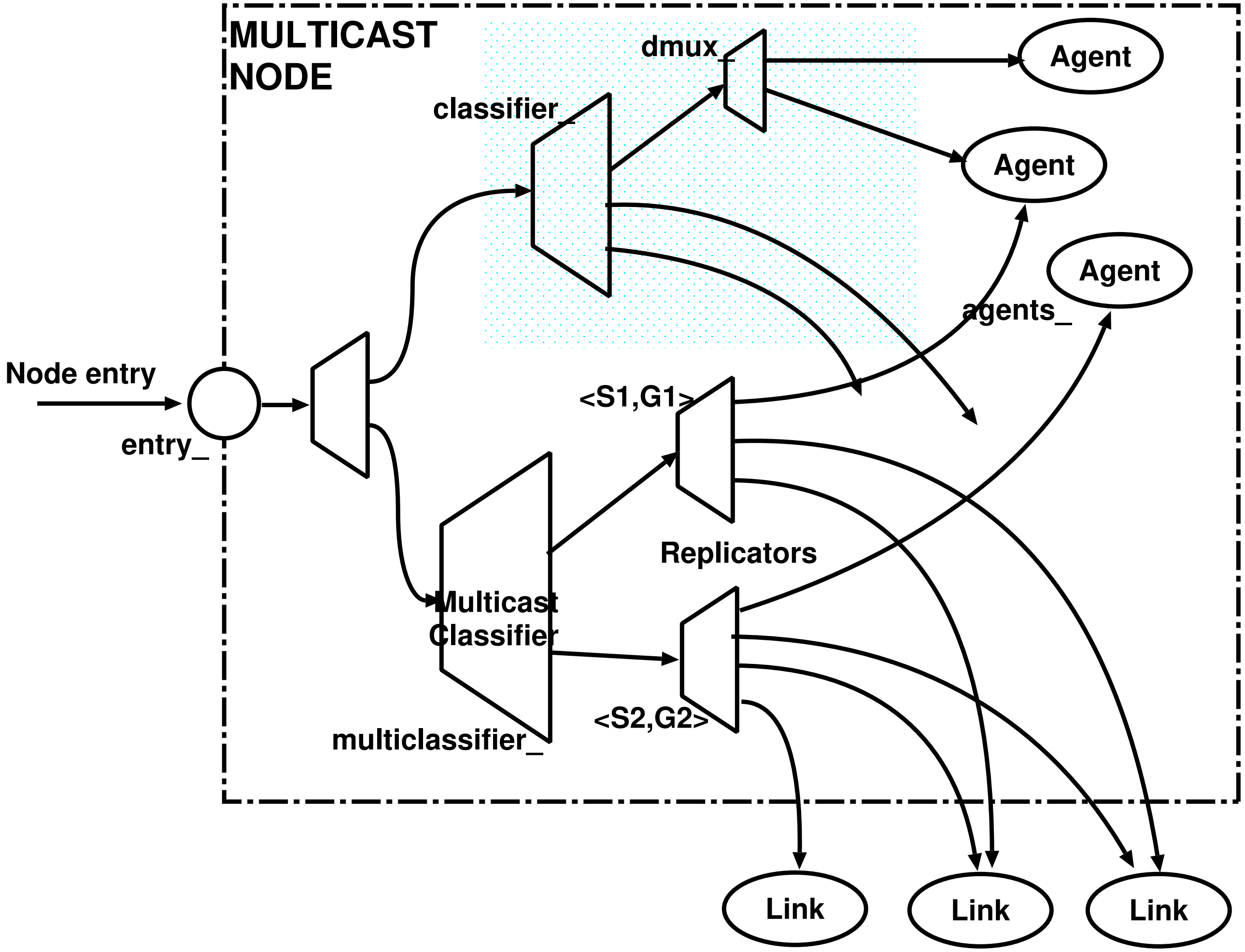



```
$ns cost $n1 $r1 2
$ns cost $n2 $r2 2
$ns cost $r2 $n2 3
```

Multicast Routing: Configuration

```
Simulator NumberInterfaces_ 1         ;# for some multicast routing protocols
Simulator EnableMcast_ 1
set ns [new Simulator]
Node expandaddr                       ;# if #nodes > 128
# allocate nodes
$ns mrtproto ⟨protocol⟩ ⟨nodelist⟩
set group [Node allocaddr]

$node join-group  $agent $group
$node leave-group $agent $group
```

Multicast Node Definition



Multicast Routing Protocols Implemented

Valid protocols currently implemented:

- Centralised protocols:

  1. `CtrMcast` Centralised Multicast (Sparse Mode Protocol)
  2. `DM` Dense Mode

- Detailed protocols:

  1. `dynamicDM` Dynamic Dense Mode
  2. `pimDM` PIM Dense Mode

Centralised Multicast Configuration

Sparse Mode implementation of multicast

```
$ctrmcastcomp compute-mroutes
$ctrmcastcomp switch-treetype $group
```

Defaults to creating a shared tree.

Centralised Dense Mode Configuration

- computes parent-child relationships prior to start of simulation
- Can study membership dynamics
- No response to topology changes
- Only one configuration parameter

```
DM set PruneTimeout ⟨newValue⟩                ;# default 0.5s.
```

Dynamic Dense Mode Configuration

- Extension of static dense mode
- As before, computes parent child relationships at start of simulation
- Also recomputes whenever topology changes, or unicast route updates are received
- Configuration parameters are:

```
dynamicDM set PruneTimout ⟨newValue⟩            ;# default 0.5s.
dynamicDM set ReportRouteTimeout ⟨newvalue⟩     ;# default 1s.
```

PIM Dense Mode Configuration

- Extension of static dense mode
- does not compute parent child relationships
- Configuration parameters are:

```
pimDM set PruneTimout ⟨newValue⟩
```

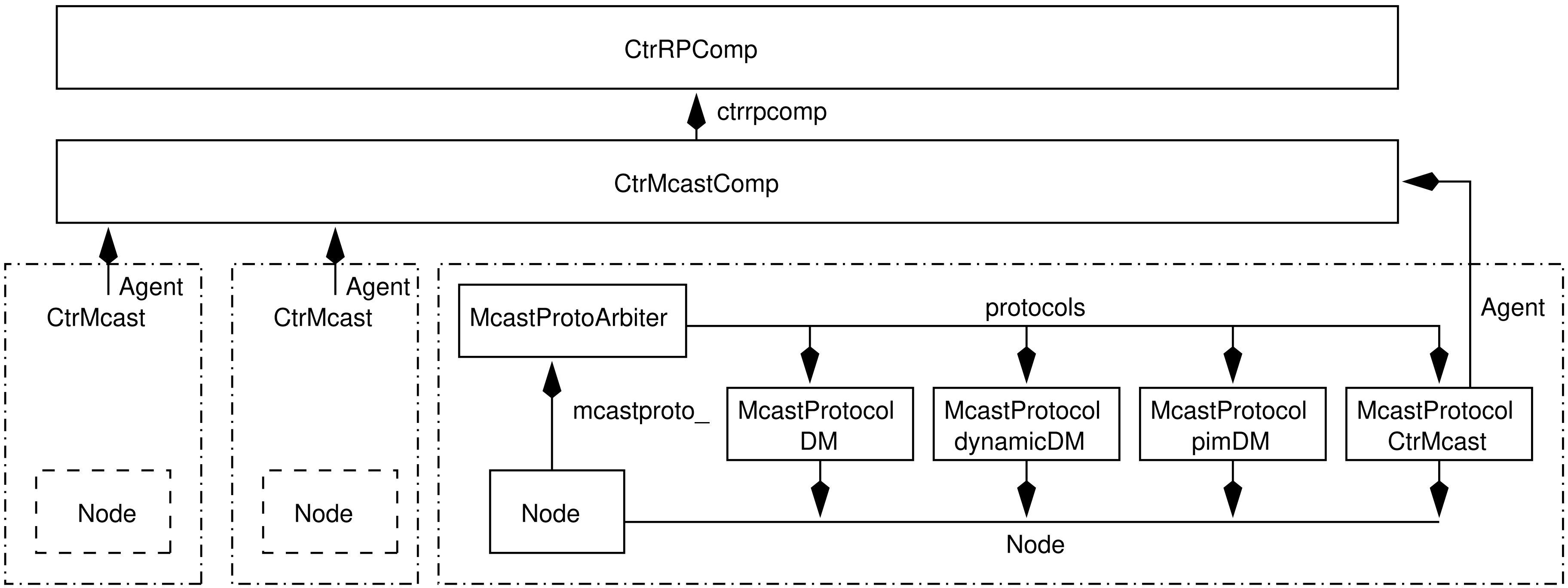
; # *default* $0.5s.$

Other protocol work in progress

Detailed PIM

Sparse mode and Dense mode

Multicast Classes



Network Dynamics

The ability to make elements of topology fail/recover.

- Model of failure and recovery

  — Exponential, Deterministic, Trace driven, Manual (or one-shot)

- Operates on single link

  — *i.e.* takes link “down” or “up”

- Operation on single node possible, translated into operation on collection of links incident on node

Specification

- Create an instance of ⟨model⟩

  `$ns` **`rtmodel`** `⟨model⟩ ⟨parameters⟩ ⟨node|link⟩`

  Command returns the handle of the model created

- Perform ⟨operation⟩ at ⟨time⟩ on ⟨node|link⟩;

  `$ns` **`rtmodel-at`** `⟨time⟩ ⟨operation⟩ ⟨node|link⟩`

  return handle to newly created instance of model.

- Delete route model specified by ⟨handle⟩

  `$ns` **`rtmodel-delete`** `⟨handle⟩`

Models and Configuration

1. Exponential
   - ⟨[startTime], upInterval, downInterval, [endTime]⟩
2. Deterministic
   - ⟨[startTime], upInterval, downInterval, [endTime]⟩
3. Trace (based)
   - ⟨fileName⟩
4. Manual (or one-shot)
   - ⟨operation, time⟩

Defining your Own Model

Derive model from the base class, `rtModel`

1. `set-parms{}` process model parameters
2. `set-first-event{}` specify the first event
3. `up{}`, `down{}` specify next action

— Use class rtQueue to schedule events.

— Instance of queue in rtModel class variable, `rtq_`

class rtQueue

Internal class to synchronise topology change activity

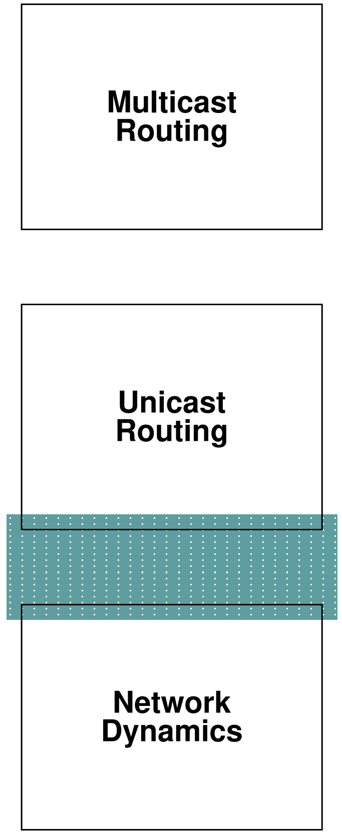
`insq{}` Given ⟨time⟩, ⟨object⟩, and an instance procedure for that object and arguments, will invoke instance procedure of object at specified time.

`insq-i{}` identical to `insq{}`, except ⟨time⟩ specifies increment when the procedure will be executed

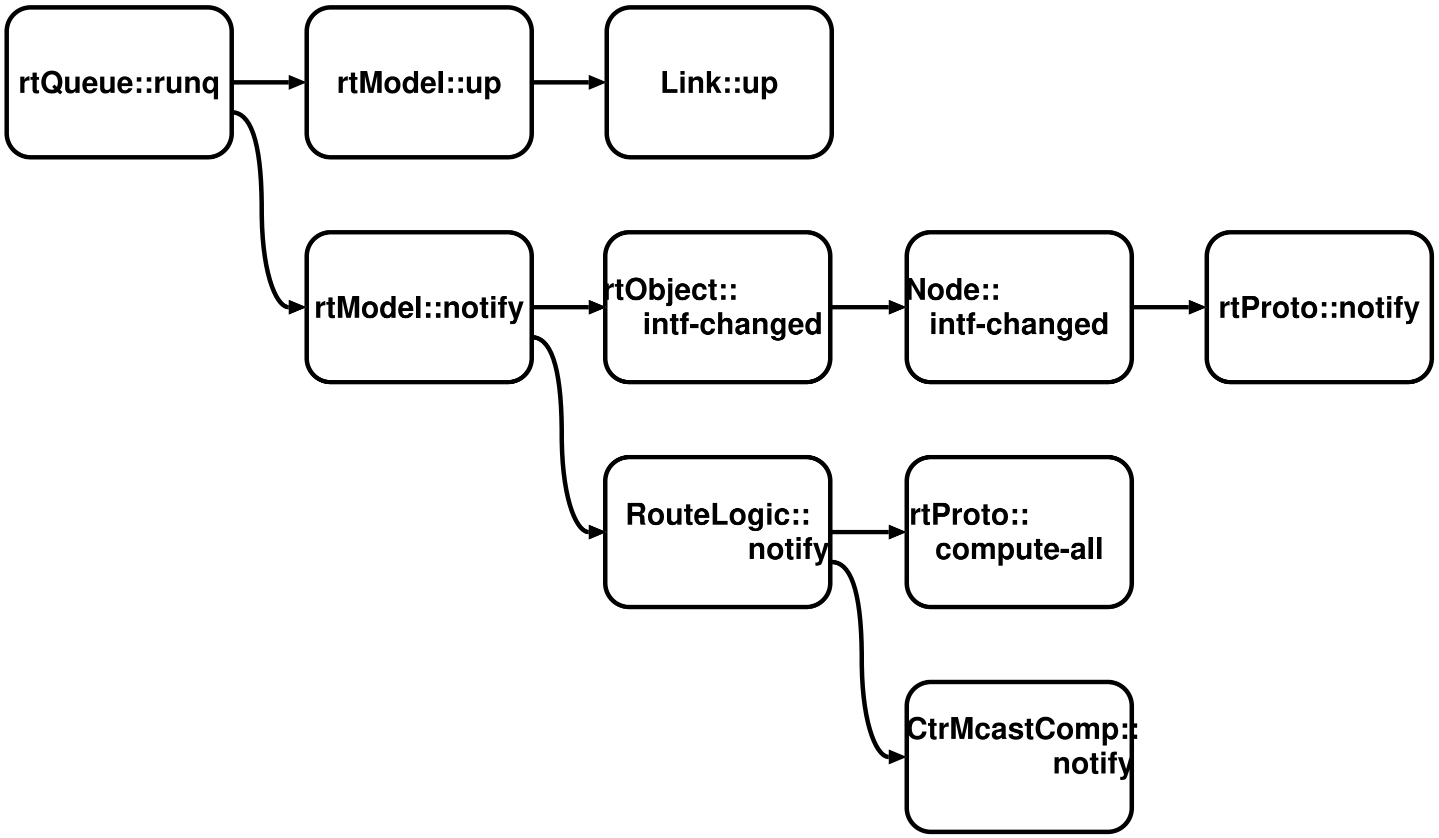
`runq{}` executed procedures for the specified time; finish by invoking `notify{}` for each object

`delq{}` remove event for object at specified time

Interface to Unicast Routing



Interface to Unicast Routing



Deficiencies in the Network Dynamics API

- Link centric
- Node failure not consistent with an operation model
- API cannot specify clusters/clouds

Multicast Transport

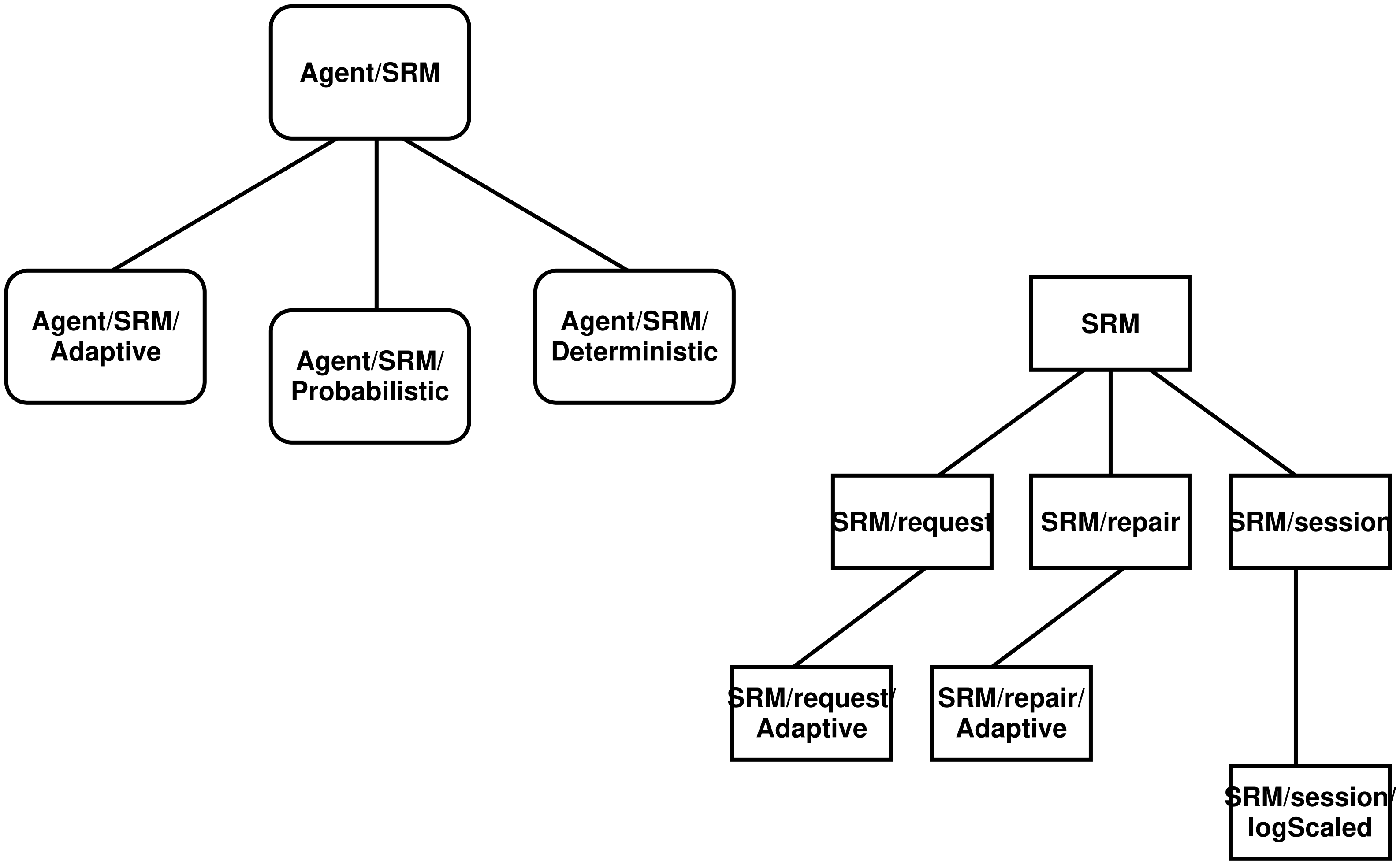
Currently, SRM and RTP implemented

- goals of implementation: programmability
- in particular, for SRM: programming timers, session message handling, loss recovery
- and for RTP, mostly programmed in OTcl, with minimalpacket related statistics counters in C++

Simple SRM configuration

```
set ns [new Simulator]                          ;# preamble initialization
$ns enableMcast
set node [$ns node]                             ;# agent to reside on this node
set group [$ns allocaddr]                       ;# multicast group for this agent

set srm [new Agent/SRM]
$srm  set dst_ $group                           ;# configure the SRM agent
$ns attach-agent $node $srm

$srm set fid_ 1                                 ;# optional configuration
$srm log [open srmStats.tr w]                   ;# log statistics in this file
$srm trace [open srmEvents.tr w]                ;# trace events for this agent

set packetSize 210
#  configure the traffic generator and traffic source
set s0 [new Agent/CBR/UDP]                      ;# attach traffic generator to application
$s0 attach-traffic $exp0
$srm(0) traffic-source $s0                      ;# attach application to SRM agent
$srm(0) set packetSize_ $packetSize             ;# to generate repair packets of appropriate size

$srm start
$srm start-source
```

Class Hierarchies



Loss Detection

Triggered by receipt of data or control messages

```
$agent request ⟨sender⟩ ⟨list of messages⟩
```

Actual loss recovery occurs in OTcl through special purpose loss recovery objects

Loss Recovery

- Each loss is handled by separate “SRM” object.
- Nodes schedule “request” or “repair” objects following a data loss
- Each loss recovery object handles its own scheduling and messaging activity

Session messages

- One session message handling object per agent
- Responsible for sending periodic session messages
- Session message interval controlled by class variable, `sessionDelay_`
- Two types of session functions implemented:
  1. SRM/Session uses default fixed interval session message sending
  2. SRM/Session/logScaled uses $\log(\mathrm{groupSize\_}) * \mathrm{sessionDelay\_}$ to send next advertisement.

Statistics

1. Agent—response to Data Loss: Detection and Recovery

2. Loss Object—Per Loss recovery Statistics

Tracing

- Per event tracing by loss recovery objects
- Sample trace file is:

```
3.5543 n 1 m <1:1> r 0 Q DETECT
3.5543 n 1 m <1:1> r 1 Q INTERVALS C1 2.0 C2 0.0 d 0.0105 i 1
3.5543 n 1 m <1:1> r 1 Q NTIMER at 3.57527
...
3.5753 n 1 m <1:1> r 2 Q NACK IGNORE-BACKOFF 3.59627
3.5828 n 3 m <1:1> r 0 Q DETECT
3.5828 n 3 m <1:1> r 1 Q NTIMER at 3.6468
3.5854 n 0 m <1:1> r 0 P NACK from 257
3.5854 n 0 m <1:1> r 1 P RTIMER at 3.59586
3.5886 n 2 m <1:1> r 2 Q NTIMER at 3.67262
3.5886 n 2 m <1:1> r 2 Q NACK IGNORE-BACKOFF 3.63062
3.5959 n 0 m <1:1> r 1 P SENDREP
...
```

RTP

- Session/RTP tracks incoming data, and sends periodic delivery reports.
- Agent/CBR/RTP sends data to the group at a specified rate.

Multicast Transport Starter Kit

- Code and Documentation: SRM and RTP
- Multiple flavours of multicast

  includes Dense mode, Sparse mode, centralised multicast, session level multicast
- Multiple flavours of unicast and network dynamics

  Ability to study the problem under dynamic topologies, including partition scenarios
- Multiple physical layers including broadcast and wireless LANs
- building a catalog of canonical topologies
- Examples in ~*ns*/tcl/ex
- test suite in development

Multicast Transport: Other types

- Structured hierarchy mechanisms
  - Exists Multicast and Network Dynamics
  - Exists Some of the possible timer mechanisms
  - Exists Different loss models
  - Requires building different modules for request/repair
  - Requires defining the hierarchy and mechanisms
  - Expect mostly work in OTcl
- Route Support
  - Pointers to files for various models of implementation
  - Exists Multicast and Dynamics
  - Exists Loss Models
  - Requires modifying some network layer code
  - Expect Some minor work in C++, Lots of “plumbing” in OTcl
- Multicast Congestion Control

Current NS Scaling

NS is currently *memory limited* for large simulations.

To scale, we're taking two approaches:

- tuning/monitoring detailed simulations (to reduce memory usage)
- abstract simulations (to remove "unimportant" details)

Detailed Simulations: Limits

Some large topology configurations are in the distribution:

~*ns*/tcl/ex/newmcast/cmcast-150.tcl

150 nodes, 2200 links	Centralised multicast simulation; Uses $\approx$ 53MB
2420 nodes, 2465 links	static unicast routing; path computation algorithms; Uses $\approx$ 800MB
500 nodes, 815 links	sparse multicast group of 80 members; studying scalable session messsages

Detailed Simulations: Object Sizes

Rough estimates of memory per object

Simulator	268KB
Node	2
Multicast Capable Node	6
duplex-link	9
duplex-link w. interfaces	14

Hints to reduce memory usage

- avoid trace-all
- use arrays [$a(1), $a(2)...] instead of vars [$a1, $a2]
- other hints at
  `http://www-mash.cs.berkeley.edu/ns/ns-debugging.html`

Further Memory Debugging

- purify
  - Use purify 3.2 if possible
  - purify 4.0.2 may cause *ns* to crash.
    Try using flags: -staticchecking=false -force-rebuild
  - Typical call syntax for purify is:

```
purify -staticchecking=false -force-reuild -collector=⟨ld⟩ g++ -o ns .
```

- Gray Watson ⟨gray@letters.com⟩'s dmalloc library

  At **`http://www.letters.com/dmalloc`**

  - Link into *ns*, and analyse memory usage

Sample Dmalloc Summary

Sample summary:

```
         size        count        gross function
                    172114      6358277 total
           84        16510      1386840 ra=0x8064846
     subtotal        42634      1000426 TclObject::bind(char const *, int *)
           18           12          216 "
           19          522         9918 "
    ...
           32        30263       968416 StringCreate
     subtotal        30158       742472 NewVar
           24        30077       721848 "
           27            1           27 "
    ...
```

Additional Debugging Hints

Instructions are at:

`http://www-mash.cs.berkeley.edu/ns/ns-debugging.html`

None of this impacts order of magnitude improvements

Beyond this, Session Level Simulations. . .

Abstract or Session-level Simulations

- Why Session-level Packet Distribution? — Scaling

```
before – 150  nodes, 2180 links, ~53 MB
after  – 2000 nodes, 8108 links, ~40 MB
```

- When to use Session-level Packet Distribution?
  - multicast simulations
  - low source rate (lower than the bottlenect link)
  - little cross-traffic
  - *e.g.*, SRM sessions

- ***Caveat: Work in Progress***

Tradeoff: Accuracy

- ignores queuing delay within the routers

Abstractions

- Nodes are compacted, only store node id and port id

  i.e., no classifiers, replicators, route tables, *etc*

- Links only store bw/delay attributes

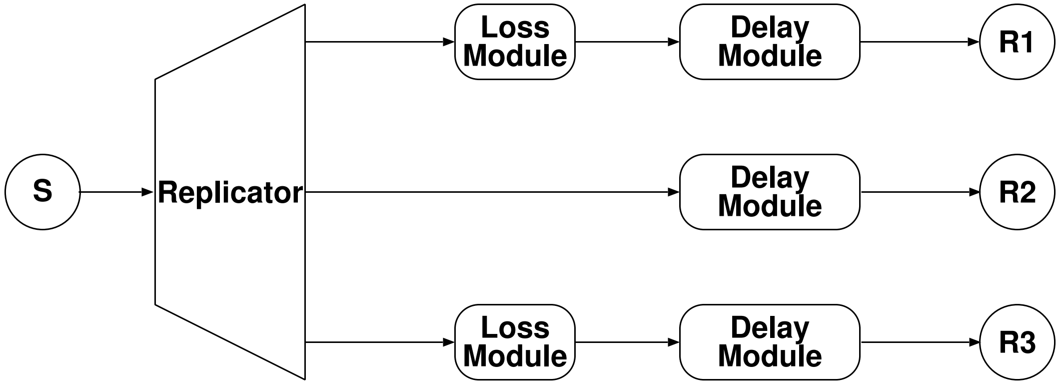
  i.e., no queues, delay elements, tracing, classifiers, *etc*

- Links get translated into a virtual end-to-end mesh. . . sorta

Configuration

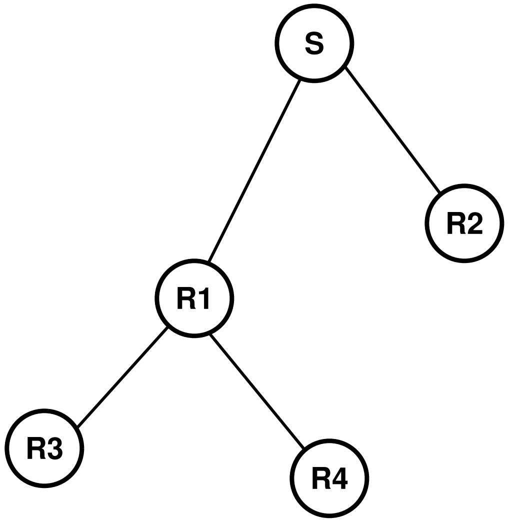
```
set ns [new SessionSim]                    ;# note difference in simulator

#  configuration of topology and multicast as usual

#  configuration source agent as usual
set srcAgent [new Agent/CBR]
$srcAgent set dst_ ⟨dst⟩
$ns attach-agent $node $srcAgent

set sessionHelper [$ns create-session $node $srcAgent]
```

Internal Realisation of a Session



Realising Loss

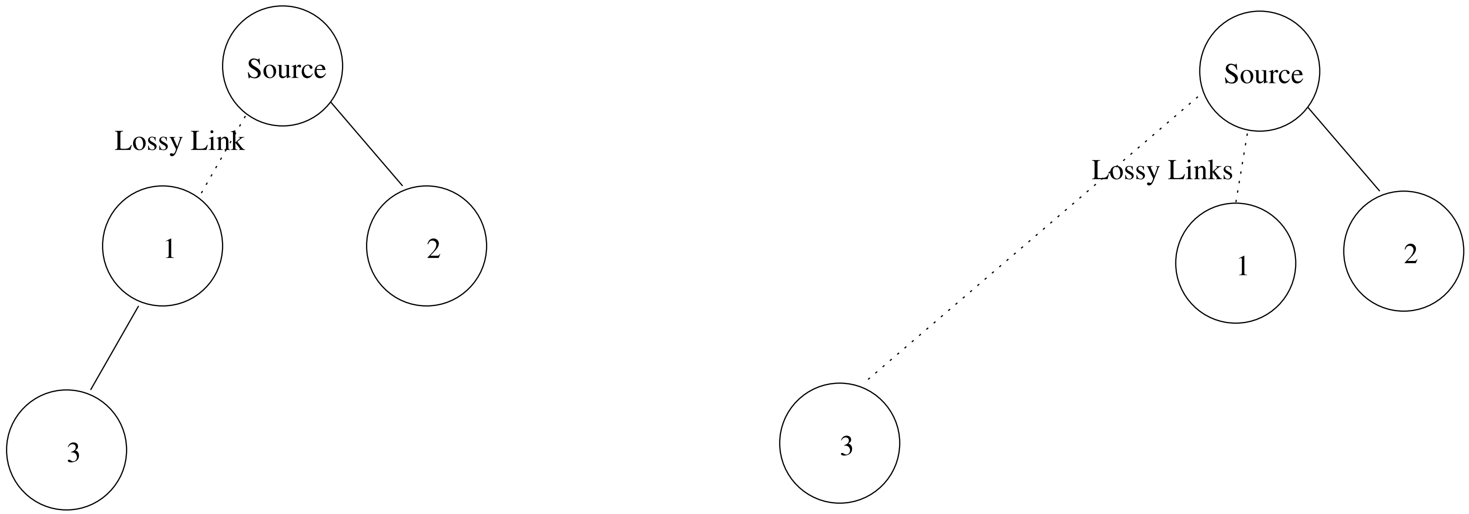
```
$node join-group $receiver $group

$sessionHelper insert-loss $lossModule $receiver
$sessionHelper insert-depended-loss $receiver $srcAgent $group
```



- Location of loss impacts translation into virtual mesh

Realisation and Comparisons

Comparison of Multicast trees



Abstract Simulation Summary

- Almost identical configuration
- Lots of Work still in progress
  - Loss Dependency
  - Queueing approximation models
  - Calibrating Error
  - Mixed simulations
- Completed but not yet checked in:
  - Early versions of Loss Dependency completed

Debugging in *ns*: Overview

- Correctness
  - tcldebugger
  - gdb
- Coexistence of gdb and tcldebugger
- Support in the Simulator

Tcl debugger

- Don Libes ⟨libes@nist.gov⟩'s Tcl debugger, written in Tcl
- At **`http://expect.nist.gov/tcl-debug/`**
- single stepping through lines of code
- supports break points based on procedures, or with arbitrary regular expressions

Tcl debugger

- Can also trap to debugger from the script, place `debug 1` at the appropriate location
- works with in user's simulation scripts
- works even through (or in) embeddedTcl code
- examine and set data or code using Tcl-ish commands

Co-existence Semi-seamless Debugging

```
(gdb) run
Starting program: /nfs/prot/kannan/PhD/simulators/ns/ns-2/ns
...
^C
Program received signal SIGINT, Interrupt.
0x102218 in write ()
(gdb) call Tcl::instance().eval("debug 1")
15: lappend auto_path $dbg_library
dbg15.3> w
*0: application
 15: lappend auto_path /usr/local/lib/dbg
dbg15.4> Simulator info instances
_o1
dbg15.5> _o1 now
0
dbg15.6> # and other fun stuff
dbg15.7> c
(gdb) where
#0  0x102218 in write ()
#1  0xda684 in FileOutputProc ()
...
(gdb) c
```

$ns gen-map output

```
% $ns gen-map
Node _o6(id 0)
                classifier__o7(Classifier/Addr)
                dmux_(NULL_OBJECT)

        Link _o11, fromNode_ _o6(id 0) -> toNode_ _o8(id 1)
        Components (in order) head first
                _o10    Queue/DropTail
                _o12    DelayLink
                _o14    TTLChecker
---
Node _o8(id 1)
                classifier__o9(Classifier/Addr)
                dmux_(NULL_OBJECT)

        Link _o16, fromNode_ _o8(id 1) -> toNode_ _o6(id 0)
        Components (in order) head first
                _o15    Queue/DropTail
                _o17    DelayLink
                _o19    TTLChecker
---
%
```