Part I: Fundamentals

Outline

- Overview
- ns Primer
  - Getting started
  - Wired world
  - Wireless world
- Emulator

ns Primer – Wired World

- Basic ns
  - Architecture
  - Basic Tcl, OTcl
  - Elements of ns
- A complete example
  - Multicast routing
- Visualization

ns Architecture

- Object-oriented (C++, OTcl)
- Scalability + Extensibility
  - Control/"data" separation
  - Split C++/OTcl object
- Modular approach
  - Fine-grained object composition

Object-Oriented

+ Reusability
+ Maintenance

– Performance (speed and memory)
– Careful planning of modularity

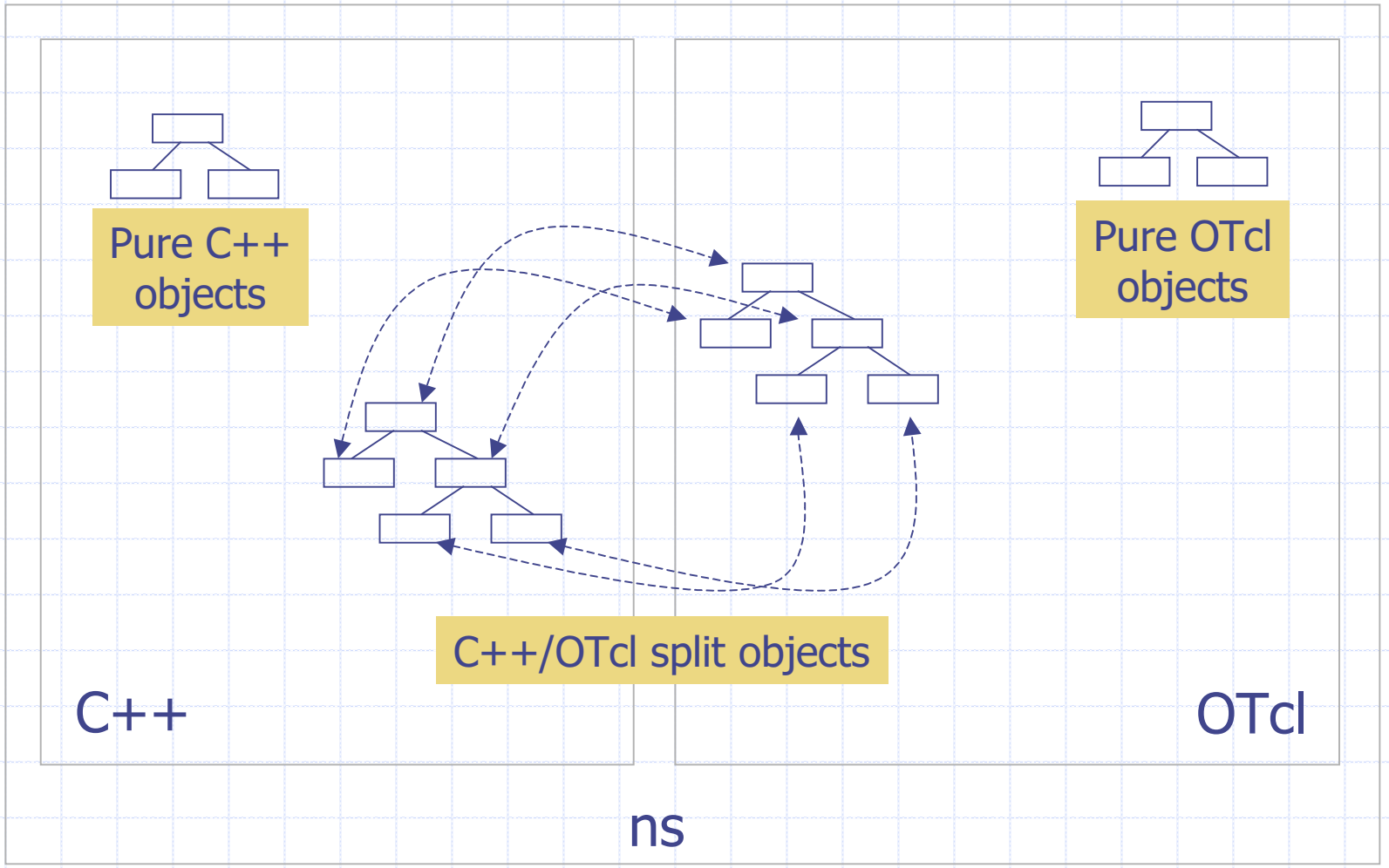
C++ and OTcl Separation

- C++ for "data"
  - Per packet action
- OTcl for control
  - Periodic or triggered action

+ Compromise between composibility and speed

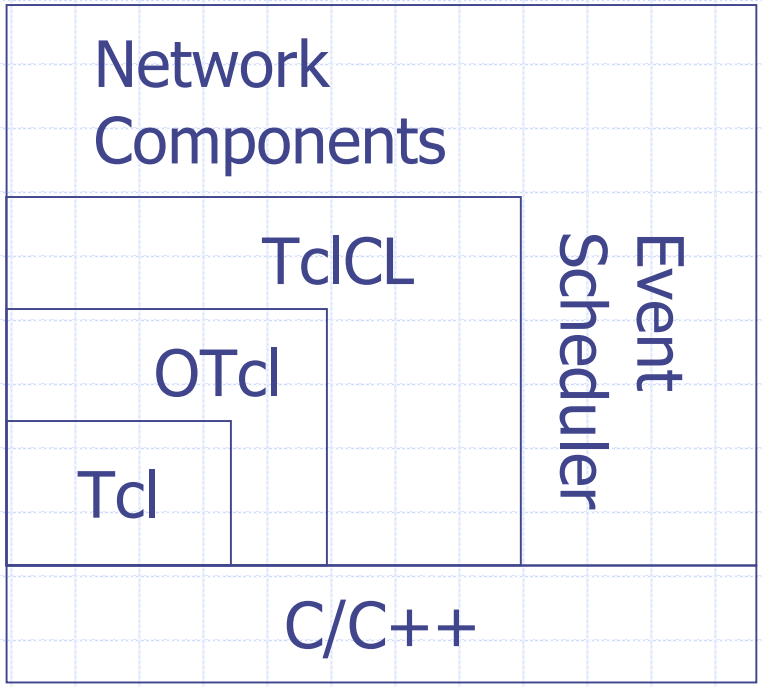
– Learning and debugging

OTcl and C++: The Duality



Extending Tcl Interpreter

- OTcl: object-oriented Tcl
- TclCL: C++ and OTcl linkage
- Discrete event scheduler
- Data network components
  - Link layer and up
  - Emulation support



ns-2

Hello World - Interactive Mode

```
swallow 71% ns
% set ns [new Simulator]
_o3
% $ns at 1 "puts \"Hello World!\""
1
% $ns at 1.5 "exit"
2
% $ns run
Hello World!
swallow 72%
```

Hello World - Batch Mode

simple.tcl

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

```
swallow 74% ns simple.tcl
Hello World!
swallow 75%
```

Basic tcl

```
set a 43
set b 27
proc test { a b } {
   set c [expr $a + $b]
   set d [expr [expr $a - $b] * $c]
   for {set k 0} {$k < 10} {incr k} {
     if {$k < 5} {
       puts "k < 5, pow = [expr pow($d, $k)]"
     } else {
       puts "k >= 5, mod = [expr $d % $k]"
     }
   }
}
test 43 27
```

Basic OTcl

```
Class Mom
Mom instproc greet {} {
   $self instvar age_
   puts "$age_ years old
   mom: How are you
   doing?"
}

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

```
set mom [new Mom]
$mom set age_ 45
set kid [new Kid]
$kid set age_ 15

$mom greet
$kid greet
```

Elements of ns-2

- Create the event scheduler
- [Turn on tracing]
- Create network
- Setup routing
- Insert errors
- Create transport connection
- Create traffic
- Transmit application-level data

Creating Event Scheduler

- Create event scheduler
  - set ns [new Simulator]
- Schedule events
  - $ns at <time> <event>
  - <event>: any legitimate ns/tcl commands
- Start scheduler
  - $ns run

Tracing

- Trace packets on all links
  - $ns trace-all [open test.out w]

```
<event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
    + 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    - 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    r 1.00234 0 2 cbr 210 ------- 0 0.0 3.1 0 0
```

- Trace packets on all links in nam-1 format
  - $ns namtrace-all [open test.nam w]
- Must appear immediately after creating scheduler

Tracing

- Turn on tracing on specific links
  - $ns trace-queue $n0 $n1
  - $ns namtrace-queue $n0 $n1

Creating Network

- Nodes
  - set n0 [$ns node]
  - set n1 [$ns node]
- Links and queuing
  - $ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>
  - <queue_type>: DropTail, RED, CBQ, FQ, SFQ, DRR

Creating Network: LAN

- LAN
  - $ns make-lan <node_list> <bandwidth> <delay> <ll_type> <ifq_type> <mac_type> <channel_type>
  - <ll_type>: LL
  - <ifq_type>: Queue/DropTail,
  - <mac_type>: MAC/802_3
  - <channel_type>: Channel

Inserting Errors

- Creating Error Module
  - set loss_module [new ErrorModel]
  - $loss_module set rate_ 0.01
  - $loss_module unit pkt
  - $loss_module ranvar [new RandomVariable/Uniform]
  - $loss_module drop-target [new Agent/Null]
- Inserting Error Module
  - $ns lossmodel $loss_module $n0 $n1

Network Dynamics

- Link failures
  - Hooks in routing module to reflect routing changes
- Four models

```
$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
```

- Parameter list

```
[<start>] <up_interval> <down_interval> [<finish>]
```

Setup Routing

- Unicast
  - $ns rtproto <type>
  - <type>: Static, Session, DV, cost, multi-path
- Multicast
  - $ns multicast (right after [new Simulator])
  - $ns mrtproto <type>
  - <type>: CtrMcast, DM, ST, BST

Creating Connection: 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

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]

Creating Connection: 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

Creating Traffic: On Top of TCP

- FTP
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp
- Telnet
  - set telnet [new Application/Telnet]
  - $telnet attach-agent $tcp

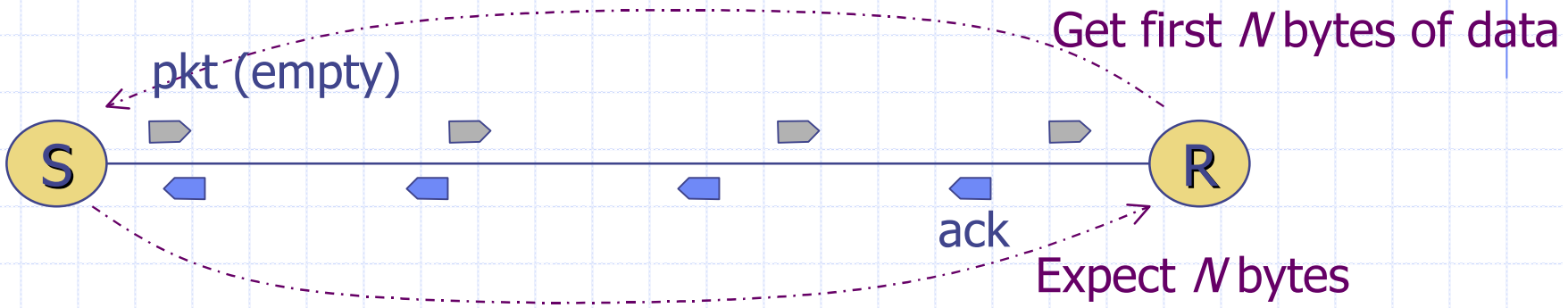
Creating Traffic: Trace Driven

- Trace driven
  - set tfile [new Tracefile]
  - $tfile filename <file>
  - set src [new Application/Traffic/Trace]
  - $src attach-tracefile $tfile
- <file>:
  - Binary format (native!)
  - inter-packet time (msec) and packet size (byte)

Application-Level Simulation

- Features
  - Build on top of existing transport protocol
  - Transmit user data, e.g., HTTP header
- Two different solutions
  - TCP: Application/TcpApp
  - UDP: Agent/Message

Application/TcpApp



- Abstraction: TCP as a FIFO pipe
- Before sending: S notifies about R data size
- After receiving: R gets data (arbitrary string) from S

} Out-of-band

Application/TcpApp

- Step 1: 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
```

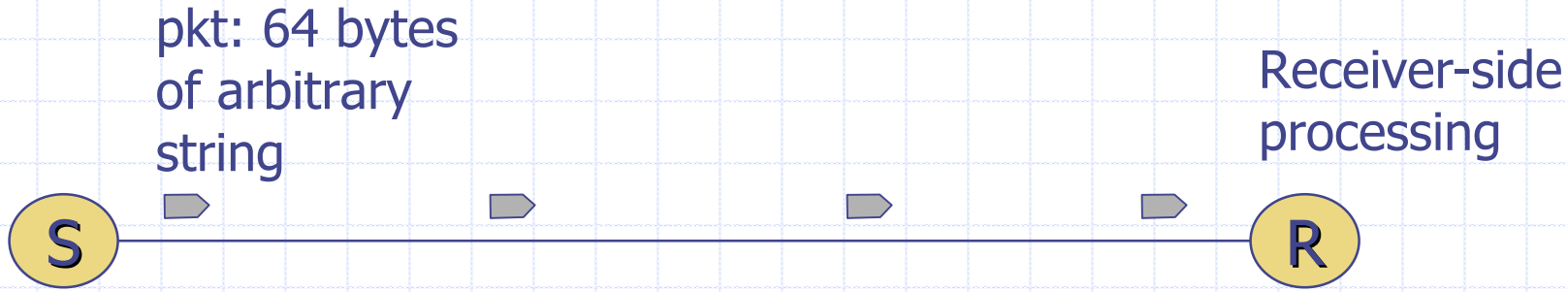
Application/TcpApp

- Step 2: reliable, in-order user data transfer

```
set app1 [new Application/TcpApp $tcp1]
set app2 [new Application/TcpApp $tcp2]
$app1 connect $app2

# <ns-2 command>: will be executed when
  received at the receiver TcpApp
$ns at 1.0 "$app1 send <data_byte>
  \"<ns-2 command>\""
```

Agent/Message



- A UDP agent (without UDP header)
- Up to 64 bytes user message
- Good for fast prototyping a simple idea
- Usage requires extending ns functionality
  - We'll give an example tomorrow

Summary: Generic Script Structure

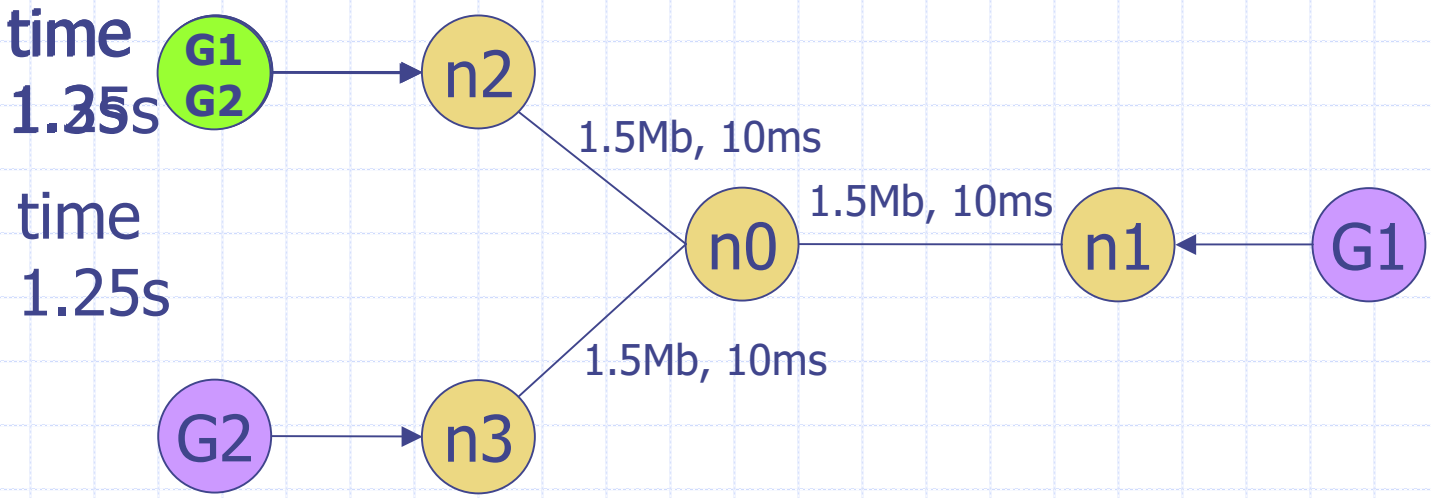
```
set ns [new Simulator]
# [Turn on tracing]
# Create topology
# Setup packet loss, link dynamics
# Create routing agents
# Create:
#    - multicast groups
#    - protocol agents
#    - application and/or setup traffic sources
# Post-processing procs
# Start simulation
```

ns Primer – Wired World

- Basic ns
- A complete example
  - Multicast routing
- Visualization

Example: Multicast Routing

- Dynamic group membership under Dense Mode



Multicast: Step 1

- Scheduler, tracing, and topology

```
# Create scheduler
set ns [new Simulator]

# Turn on multicast
$ns multicast

# Turn on Tracing
set fd [new "mcast.nam" w]
$ns namtrace-all $fd
```

Multicast: Step 2

- Topology

```
# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# Create links
$ns duplex-link $n0 $n1 1.5Mb 10ms DropTail
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n0 $n3 1.5Mb 10ms DropTail
```

Multicast: Step 3

- Routing and group setup

```
# Routing protocol: let's run distance vector
$ns mrtproto DM

# Allocate group addresses
set group1 [Node allocaddr]
set group2 [Node allocaddr]
```

Multicast: Step 4

- Sender 0

```
# Transport agent for the traffic source
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
$udp0 set dst_addr_ $group1
$udp0 set dst_port_ 0

# Constant Bit Rate source #0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
# Start at time 1.0 second
$ns at 1.0 "$cbr0 start"
```

Multicast: Step 5

- Sender 1

```
# Transport agent for the traffic source
set udp1 [new Agent/UDP]
$ns attach-agent $n3 $udp1
$udp1 set dst_addr_ $group2
$udp1 set dst_port_ 0

# Constant Bit Rate source #0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
# Start at time 1.1 second
$ns at 1.1 "$cbr1 start"
```

Multicast: Step 6

- Receiver with dynamic membership

```
# Can also be Agent/Null
set rcvr [new Agent/LossMonitor]

# Assign it to node $n2
$ns at 1.2 "$n2 join-group $rcvr $group2"
$ns at 1.25 "$n2 leave-group $rcvr $group2"
$ns at 1.3 "$n2 join-group $rcvr $group2"
$ns at 1.35 "$n2 join-group $rcvr $group1"
```

Multicast: Step 7

- End-of-simulation wrapper (as usual)

```
$ns at 2.0 "finish"
proc finish {} {
    global ns fd
    close $fd
    $ns flush-trace
    puts "running nam..."
    exec nam out.nam &
    exit 0
}
$ns run
```

Other Examples

- Available in the Lab this afternoon
- Web traffic model
- Multicast routing
- RED
- Queueing

ns Primer – Wired World

- Basic ns
- Two examples
  - TCP, multicast routing
- Visualization

Visualization Tools

- nam-1 (Network AniMator Version 1)
  - Packet-level animation
  - Well supported by ns
- xgraph
  - Conversion from ns trace to xgraph format

nam

- Basic visualization
  - Topology layout
  - Animation control
  - Synchronous replay
- Fine-tune layout
- TCP/SRM visualization
- Editor: generate ns simulation scripts

ns→nam Interface

- Color
- Node manipulation
- Link manipulation
- Topology layout
- Protocol state
- Misc

nam Interface: Color

- Color mapping

```
$ns color 40 red
$ns color 41 blue
$ns color 42 chocolate
```

- Color ↔ flow id association

```
$tcp0 set fid_ 40 ;# red packets
$tcp1 set fid_ 41 ;# blue packets
```

nam Interface: Nodes

- Color

```
$node color red
```

- Shape (can't be changed after sim starts)

```
$node shape box  ;# circle, box, hexagon
```

- Marks (concentric "shapes")

```
$ns at 1.0 "$n0 add-mark m0 blue box"
$ns at 2.0 "$n0 delete-mark m0"
```

- Label (single string)

```
$ns at 1.1 "$n0 label \"web cache 0\""
```

nam Interfaces: Links

- Color

```
$ns duplex-link-op $n0 $n1 color "green"
```

- Label

```
$ns duplex-link-op $n0 $n1 label "abced"
```

- Dynamics (automatically handled)

```
$ns rtmodel Deterministic {2.0 0.9 0.1} $n0 $n1
```

- Asymmetric links not allowed

nam Interface: Topo Layout

- "Manual" layout: specify everything

```
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(1) $n(2) orient right
$ns duplex-link-op $n(2) $n(3) orient right
$ns duplex-link-op $n(3) $n(4) orient 60deg
```

- If anything missing → automatic layout

nam Interface: Protocol State

- Monitor values of agent variables

```
$ns add-agent-trace $srm0 srm_agent0
$ns monitor-agent-trace $srm0
$srm0 tracevar C1_
$srm0 tracevar C2_
# … …
$ns delete-agent-trace $tcp1
```

nam Interface: Misc

- Annotation
  - Add textual explaination to your sim

```
$ns at 3.5 "$ns trace-annotate \"packet drop\""
```

- Set animation rate

```
$ns at 0.0 "$ns set-animation-rate 0.1ms"
```

Multicast Example: nam-Enhanced

- Packet coloring
- Node color
- Node label
- Link label
- Annotation
- Manual layout
- Queueing

Multicast: Step 1.1

- Define nam color

```
# Colors for packets from two mcast groups
$ns color 10 blue
$ns color 11 red

# Prune packets (predefined)
$ns color 30 purple
# Graft packets
$ns color 31 green
```

Multicast: Step 2.1

- Layout topology

```
# Manual layout: order of the link is significant!
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n0 $n3 orient right-down

# Show queue on simplex link n0->n1
$ns duplex-link-op $n0 $n1 queuePos 0.5
```

Multicast: Step 4.1, 5.1

- Source coloring

```
# Group 0
$udp0 set fid_ 10
$n1 color blue
$n1 label "Source for group 0"

# Group 1
$udp1 set fid_ 11
$n3 color red
$n3 label "Source for group 1"
```

Multicast: Step 6.1

- Receiver coloring

```
$n2 label “Receiver”
$ns at 1.2 "$n2 join-group $rcvr $group2; \
   $n2 add-mark m0 red"
$ns at 1.25 "$n2 leave-group $rcvr  $group2; \
   $n2 delete-mark m0"
$ns at 1.3 "$n2 join-group $rcvr \ $group2; \
   $n2 add-mark m1 red"
$ns at 1.35 "$n2 join-group $rcvr $group1; \
   $n2 add-mark m2 blue"
```

Multicast: Step 7.1

- One final tweak

```
# Animation was too fast...
$ns set-animation-rate 0.8ms
```