(An Incomplete)

Introduction to NS-2

Bahador Bakhshi

High Speed Networks Lab.

CE & IT Department

Amirkabir University of Technology

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

What We Will NOT Learn

- Multicast
- Satellite networks
- MPLS & DiffServ
- LANs
- In details TCP variations
- A few transport protocols
- HTTP & Worms
- Emulation
- And a lot of other things ;-)

Who are we?

- You
  - Your background
  - Your skill
  - Your objective

- Me
  - Who am I?
  - I am not NS-2 guru
  - Resources are available at

    ceit.aut.ac.ir/~bakhshis/NS-2/

What is the NS-2?

- NS-2: Network Simulator version 2.
- NS-2
  - Is a discrete event simulator for networking research
  - Simulates at packet level
  - Substantial support to simulate many protocols
  - Simulate wired and wireless network
  - Is primarily Unix based.
- NS-2 is the de facto experiment environment in research community

Source: [1]

What is not the NS-2?

- Easy to use friendly simulator
- Well documented easy to understand programming environment
- Well designed software
- Bug free software
- NS-2 is not for dummies

Advantages & Disadvantages

- Advantages
- Open source
  - Free (no money)
- Supported protocols
- Supported platforms
- Modularity
- Popular
- Documentation

- Disadvantage
- Complicated structure
- Bugs
  - Unreliable
  - Simulation validation
- Patching & Extending
- Unrealistic abstraction
- Speed & Memory

Source: [2,3]

Supported protocols

- Wired Networking
  - Routing: Unicast, Multicast, and Hierarchical Routing, etc.
  - Transportation: TCP, UDP, others;
  - Traffic sources: web, ftp, telnet, cbr, etc.
  - Queuing disciplines: drop-tail, RED, FQ, DRR, etc.
  - QoS: IntServ and Diffserv Wireless Networking
- Wireless
  - Ad hoc routing and mobile IP
  - Routing Protocol: AODV, DSDV, DSR, etc.
  - MAC layer Protocol: TDMA, CDMA(?), IEEE Mac 802.x
  - Physical layers: different channels, directional antenna
  - Sensor networks: diffusion
  - Satellite networks

Source: [4]

NS-2 and We

- NS-2 is packet level simulator
  - Details about protocol were implemented
    - TCP, UDP, Wireless MAC, …
- If we need the packet level simulation
  - We need the NS-2
- If we modify protocols
  - We need the NS-2
- Otherwise we have not use NS-2 for the particular research
- But, as a network research we need to learn it ;-)

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Installation

- The primary platform of NS-2: Linux
- It supports other platforms: Windows
- In this course
  - Linux: Fedora Core 13
    - It was already installed in VM and is ready to use
  - NS-2: 2.34
  - The “all-in-one’’ package is used
    - ns-allinone-2.34.tar.gz

Platform

- Fedora Core 13
  - All development packages were installed
- Tested in VMware 6.5.2
- root password: **********
- Networked
  - Linux (Guest) : 192.168.224.1/24
  - Windows (Host) : 192.168.224.2/24
- Samba server is enabled
- Firewall & SELinux are disabled

NS-2 installation

- The “all-in-one” package contains all components
  - In a .tar.gz file
  - Extract the file

```
tar -xzf ns-allinone-2.34.tar.gz
```

- All components are installed by a single command

```
./install
```

Post installation steps: 1

- Set a few environmental variables
  - In Bash, add the followings to the **`~/.bashrc`**
- Optional
  - Set $NS2 to the root of NS-2 package

    `NS2=/root/work/ns-allinone-2.34/`
  - Update $PATH

    `PATH=$PATH:$NS2/bin:$NS2/tcl8.4.18/unix:$NS2/tk8.4.18/unix`
- Required
  - Set $LD_LIBRARY_PATH

    `LD_LIBRARY_PATH=$NS2/otcl-1.13:$NS2/lib`
  - Set $TCL_LIBRARY

    `TCL_LIBRARY=$NS2/tcl8.4.18/library`

Post installation steps: 2

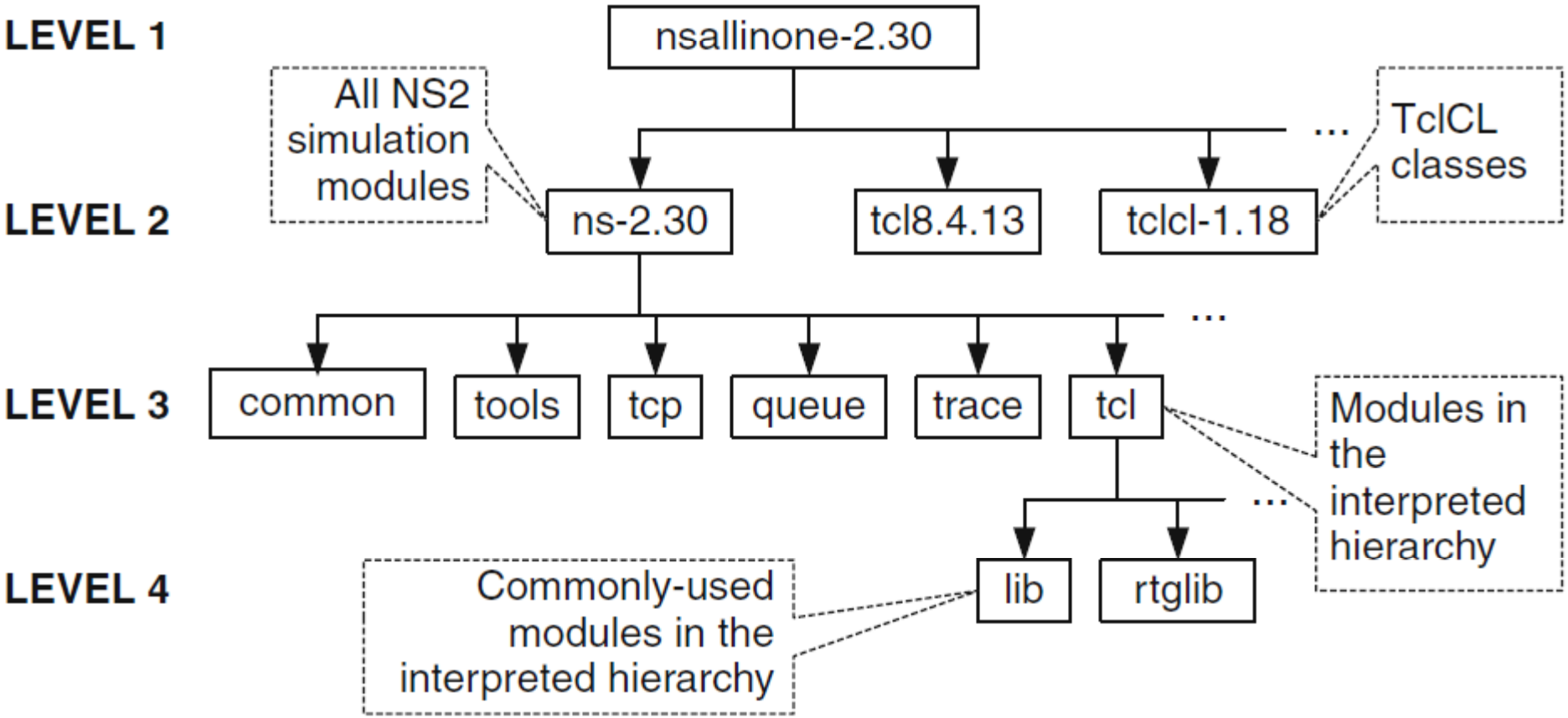
- Export the variables

```
export NS2
export PATH
export LD_LIBRARY_PATH
export TCL_LIBRARY
```

- Validate the installation

```
cd $NS2/ns-2.34
./validate
```

Directories



Source: [8]

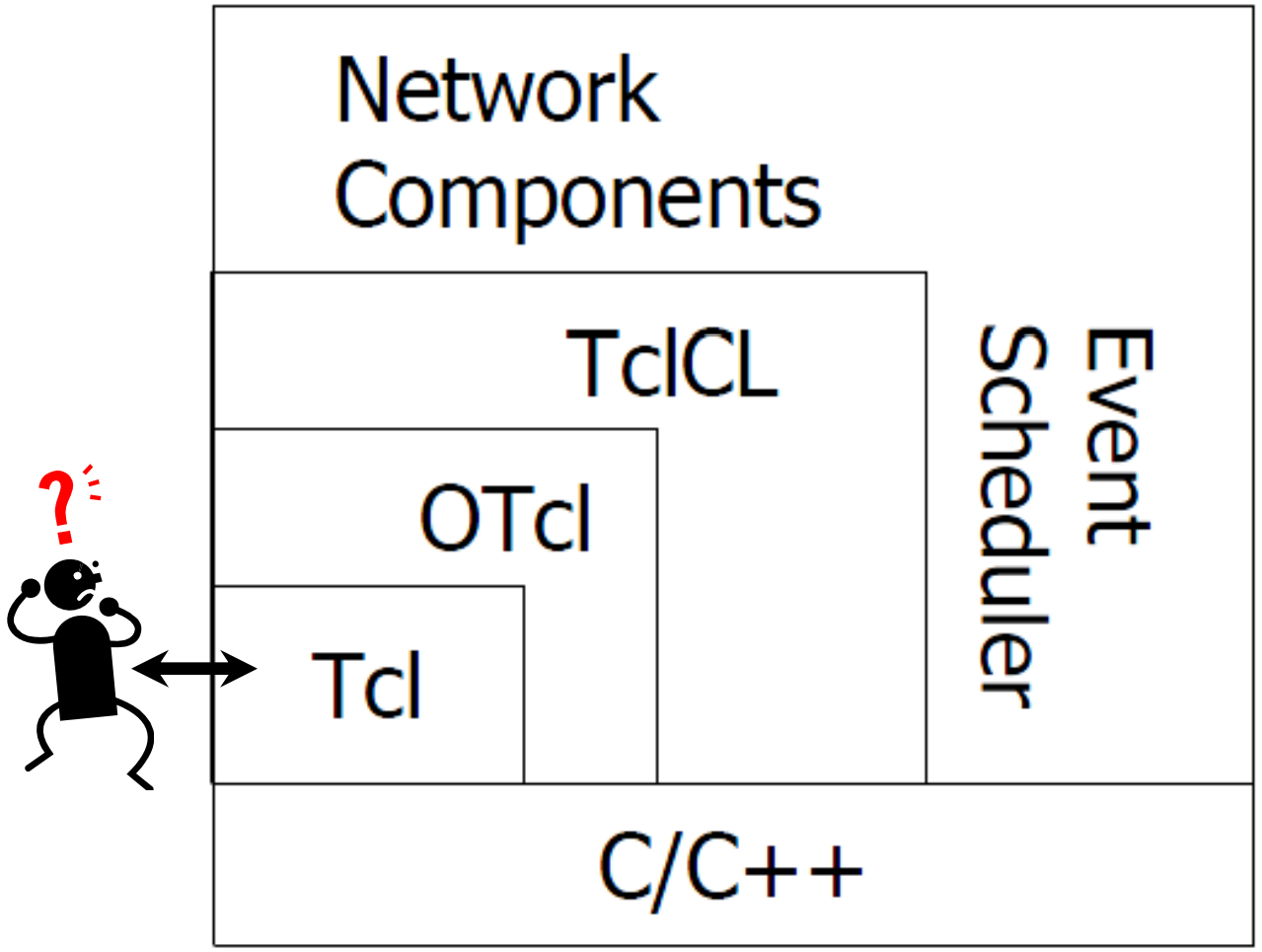
What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

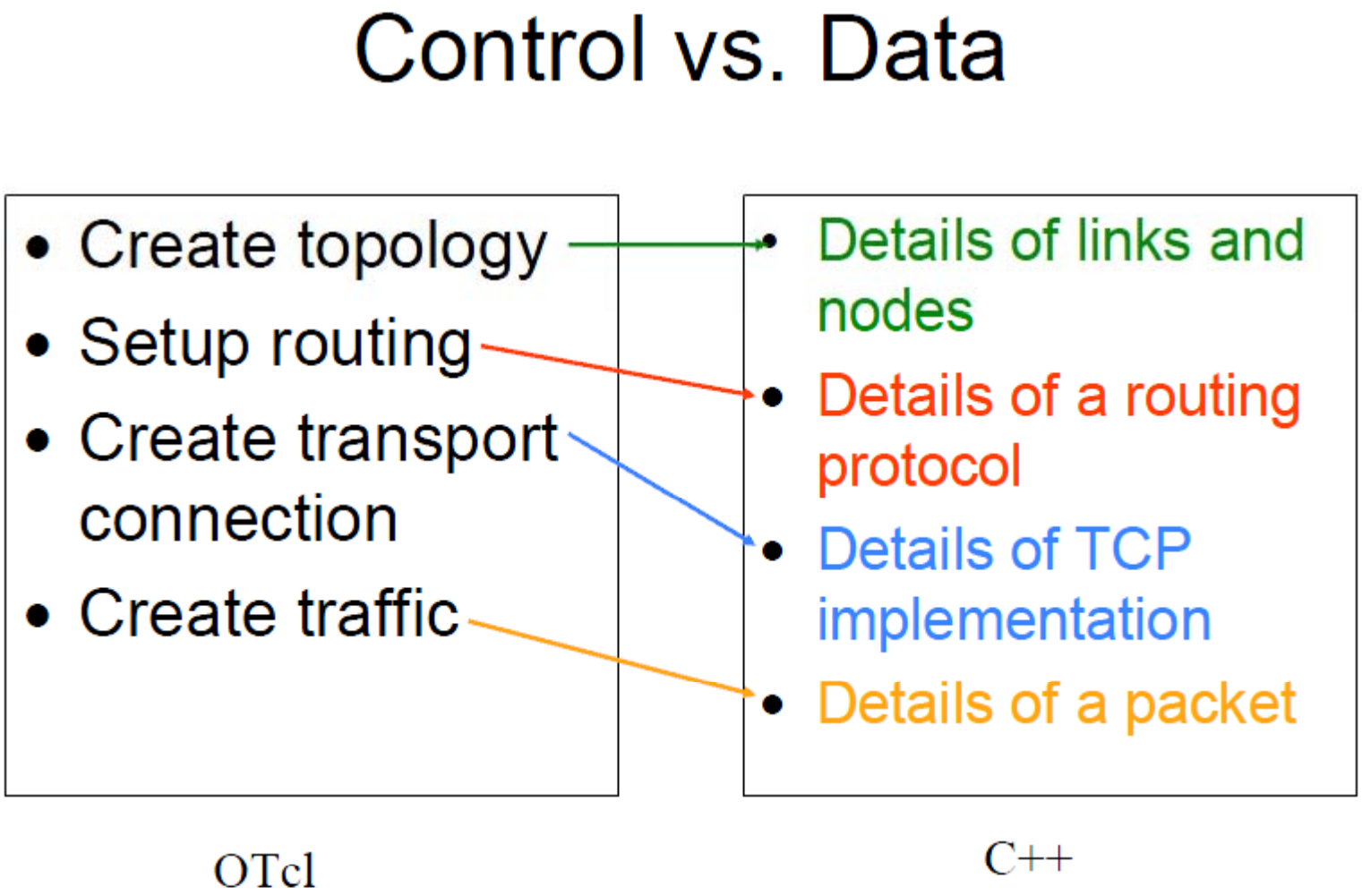
Languages

- NS-2 is implemented by two languages
  - C++
  - OTcl: OO + TCL
- Why?
  - C++ is powerful and fast language which is complied
  - OTcl is easy to use which is interpreted
- Data path, packet processing, simulation engine
  - C++
- Control path, configuration, and simulation scenarios
  - OTcl

Architecture

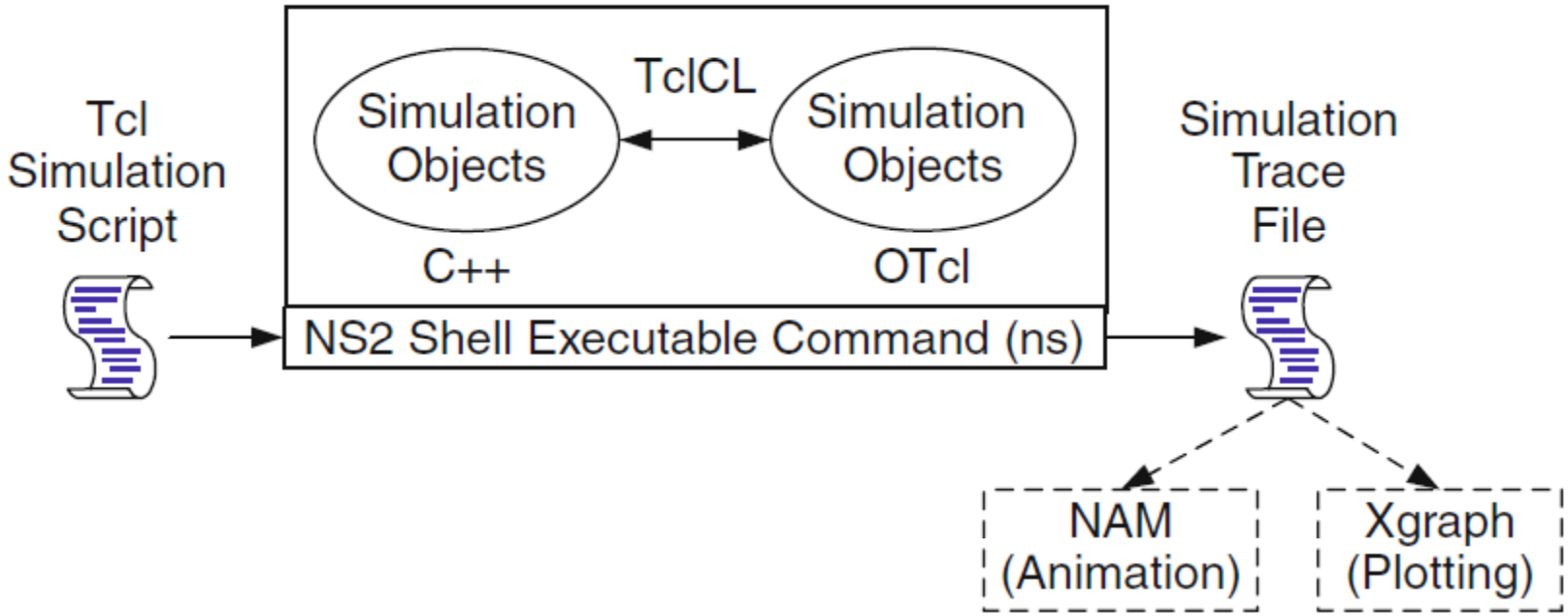


Architecture (cont'd)



Source: [6]

Architecture (cont'd)

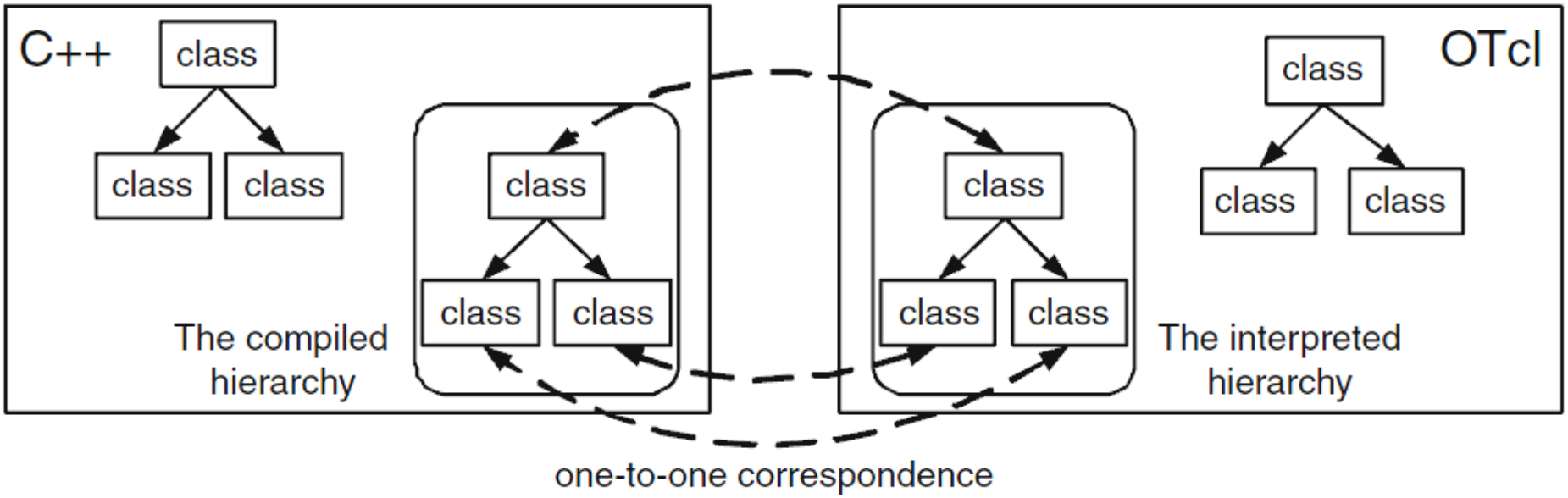


Source: [8]

Internal structure

- Two class hierarchies
  - One-to-one relation
  - C++ classes: Compiled hierarchy
  - OTcl classes: Interpreted hierarchy
- OTcl uses the complied C++ objects
  - We talk to the OTcl objects
- TclCL
  - The language for linkage between C++ and OTcl

Internal structure (cont'd)



Source: [8]

Working with NS

- NS
  - an OTcl interpreter with network libraries
- Invocation
  - Interactive mode

```
ns
%   (This is the ns prompt)
```

  - Batch mode

```
ns tclfile.tcl
```

“Hello World” Example (Interactive)

```
ns
% puts "Hello World!!!"
Hello World!!!
% puts "Hello NS newbie"
Hello NS newbie
%exit
```

“Hello World” Example (Batch)

- Hello.tcl

```
puts "Hello World!!!"
puts "Hello NS newbie"
```

- Invocation

```
ns hello.tcl
"Hello World!!!"
"Hello NS newbie"
```

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

TCL & OTcl

- Tcl stands “Toolkit Command Language”
- Tcl
  - Allows fast development
  - Provides graphical interface
  - Rich platform support
  - Flexible and easy to use
  - Free
- A Practical guide can be found in [15]
- OTcl is the object oriented extension of Tcl

Source: [5]

OTcl

- NS is an interpreter of OTcl
- We communicate with NS by OTcl
- We need to know how to use OTcl
  - Its syntax
  - Basic commands

OTcl Basics: 1

- **`set`**: Assign a value to a variable
- **`$x`**: The value of variable **`x`**
- **`[ … ]`**: Run command and return result
- **`[expr …]`**: Calculate the value of expression
- **`set x $a`**: Assign value of **`a`** to variable **`x`**
- **`puts …`** : print out
- **`proc`**: Define a procedure
- **`gets stdin x`**: Read variable **`x`** from **`stdin`**
- Be careful about spaces

OTcl: Example 1

```
puts "Enter 1th operand"
gets stdin a
puts "Enter 2th operand"
gets stdin b

set sum [expr $a + $b]
set diff [expr $a - $b]

puts "$a + $b = $sum"
puts "$a - $b = $diff"
```

OTcl: Example 2 (Fibonacci number)

```
proc next {f1 f2} {
   set res [expr $f1 + $f2]
   return $res
}
puts "Enter n >= 2"
gets stdin n
set x1 1
set x2 1
for {set i 2} {$i <= $n} {set i [expr $i + 1]} {
   set x3 [next $x1 $x2]
   set x1 $x2
   set x2 $x3
}
puts "n-th Fibonacci number = $x2"
```

Lab. Exercise

- Calculate the n!
  - Change the Fibonacci number example

- Do it yourself ;-)

OTcl Basics: 2

- `#`: Comments
- `incr x`: Increment (x = x + 1)
- `if {condition} {…}`
- `while {conditions} {…}`: While loop
- Extensive math library: `sin(x), asin(x), log(x), …`
- Array indexed can by numbers and strings
  - `set x(1) 10`
  - `set x(a) 20`

OTcl Basics: 3

- Scope variables
  - By default all variables are local
  - Global variables are specified by **`global`**

```
proc testScope {} {
   global x1
   set x1 100
   set x2 200
}
set x1 10
set x2 20
testScope
puts "x1 = $x1, x2 = $x2"
```

- Files

```
set file [open "nstrace.txt" w]
set line [gets $file]
puts $file "hello!"
close $file
```

OTcl Basics: 4 (OO programming)

C++	OTcl
constructor/destructor	init/destroy
`this`	**`$self`**
virtual	always “virtual”
`static` variable	class variable
Multiple inheritance	Multiple inheritance

Source: [6]

OTcl Basics: 4 (OO programming)

```
Class TestClass
TestClass instproc init { x } {
   puts "This is init ..."
   puts "x = $x"
}
TestClass instproc setVal {x} {
   $self instvar a
   set a $x
}
TestClass instproc getVal { } {
   $self instvar a
   return $a
}
set obj1 [new TestClass 10]
$obj1 setVal 20
set y [$obj1 getVal]
puts "y = $y"
```

OTcl Basics: 4 (OO programming)

```
Class TestClass
TestClass instproc init { x } {
   puts "This is init ..."
   puts "x = $x"
}
TestClass instproc setVal {x} {
   $self instvar a
   set a $x
}
TestClass instproc getVal { } {
   $self instvar a
   return $a
}
set obj1 [new TestClass 10]
$obj1 setVal 20
set y [$obj1 set a]   # This is one of the craziest feature of OTcl
puts "y = $y"
```

What We Will Learn

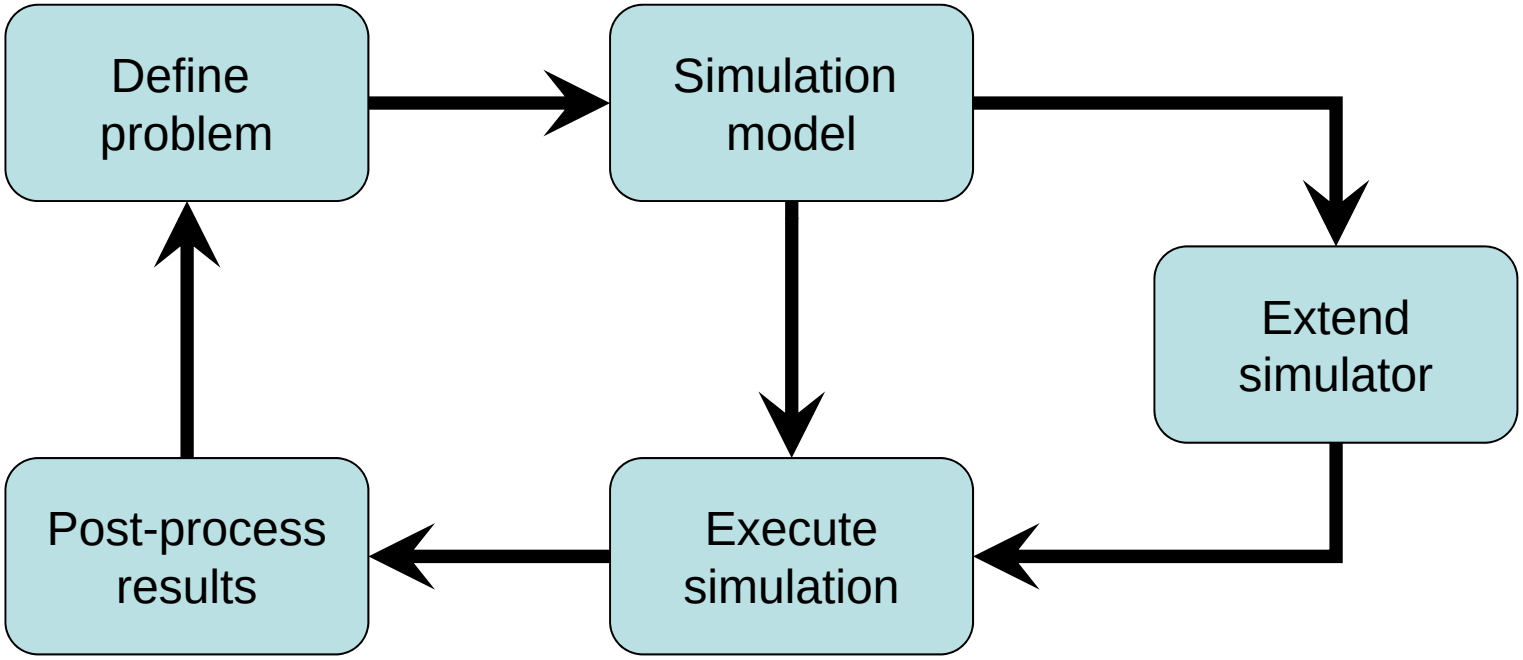
- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Simulation steps

- Simulation design
  - Simulation purposes
  - Type of expected results, …
- Configuring and Running simulation
  - Network configuration
    - Nodes, links, traffics, …
  - Simulation configuration
    - Clock, events, …
  - Run (multiple times)
- Post processing
  - Result, analyses, …

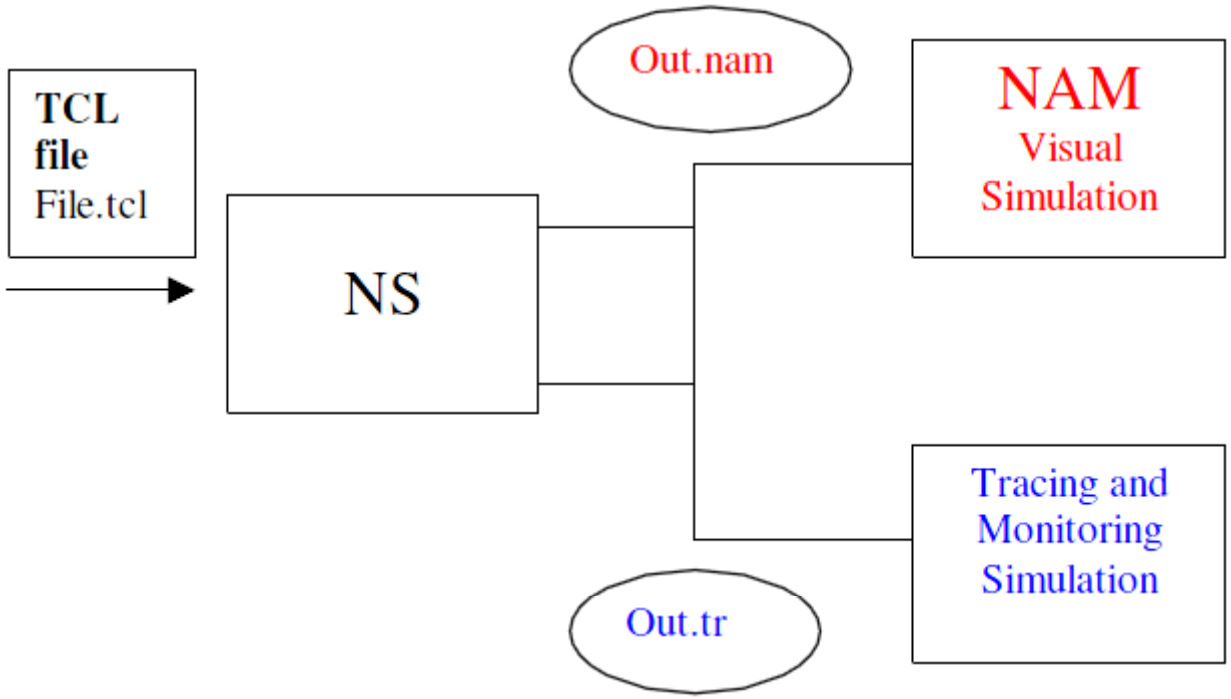
Source: [8]

Simulation steps (cont'd)



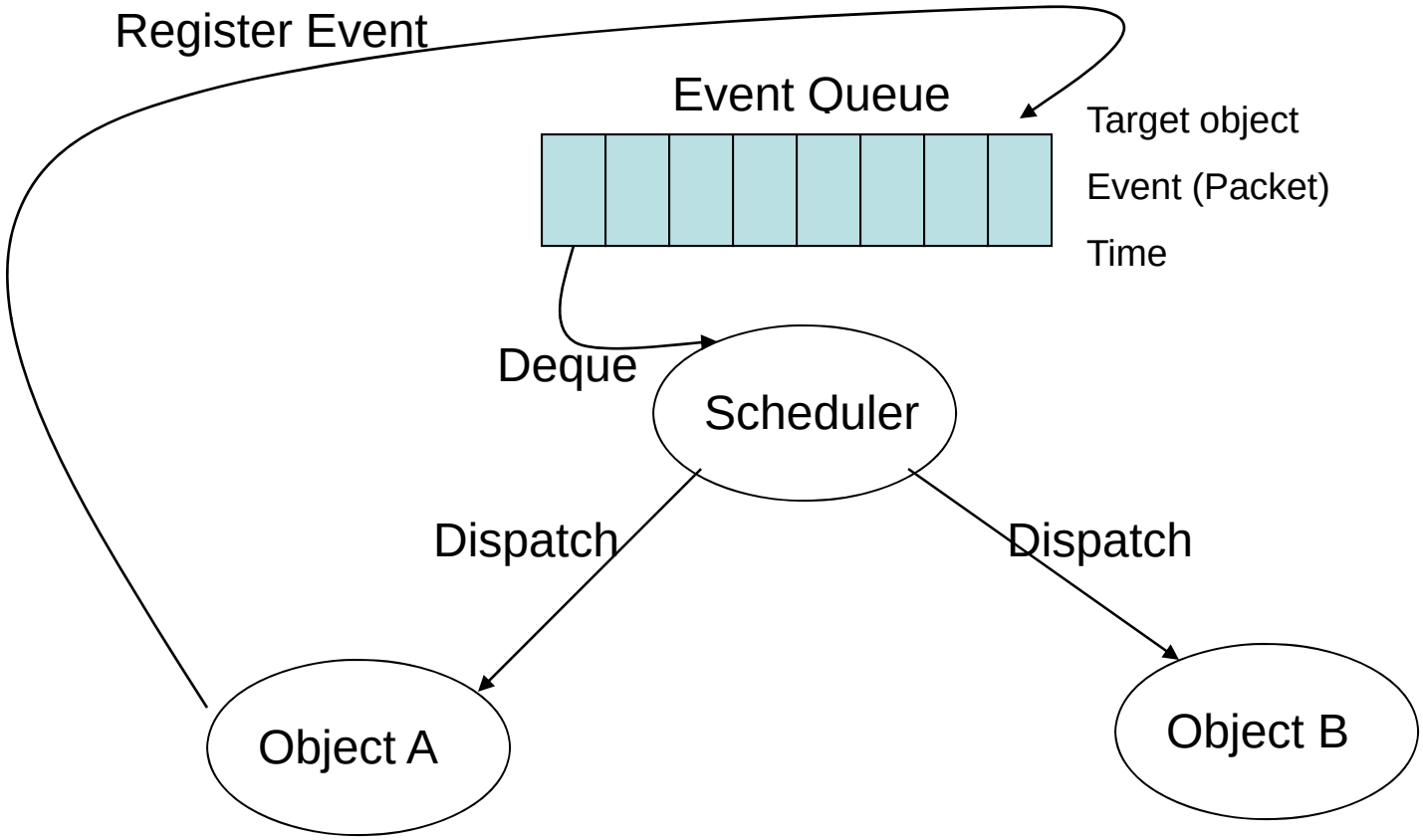
Source: [9]

Simulation steps (cont'd)



Source: [7]

How does NS work?

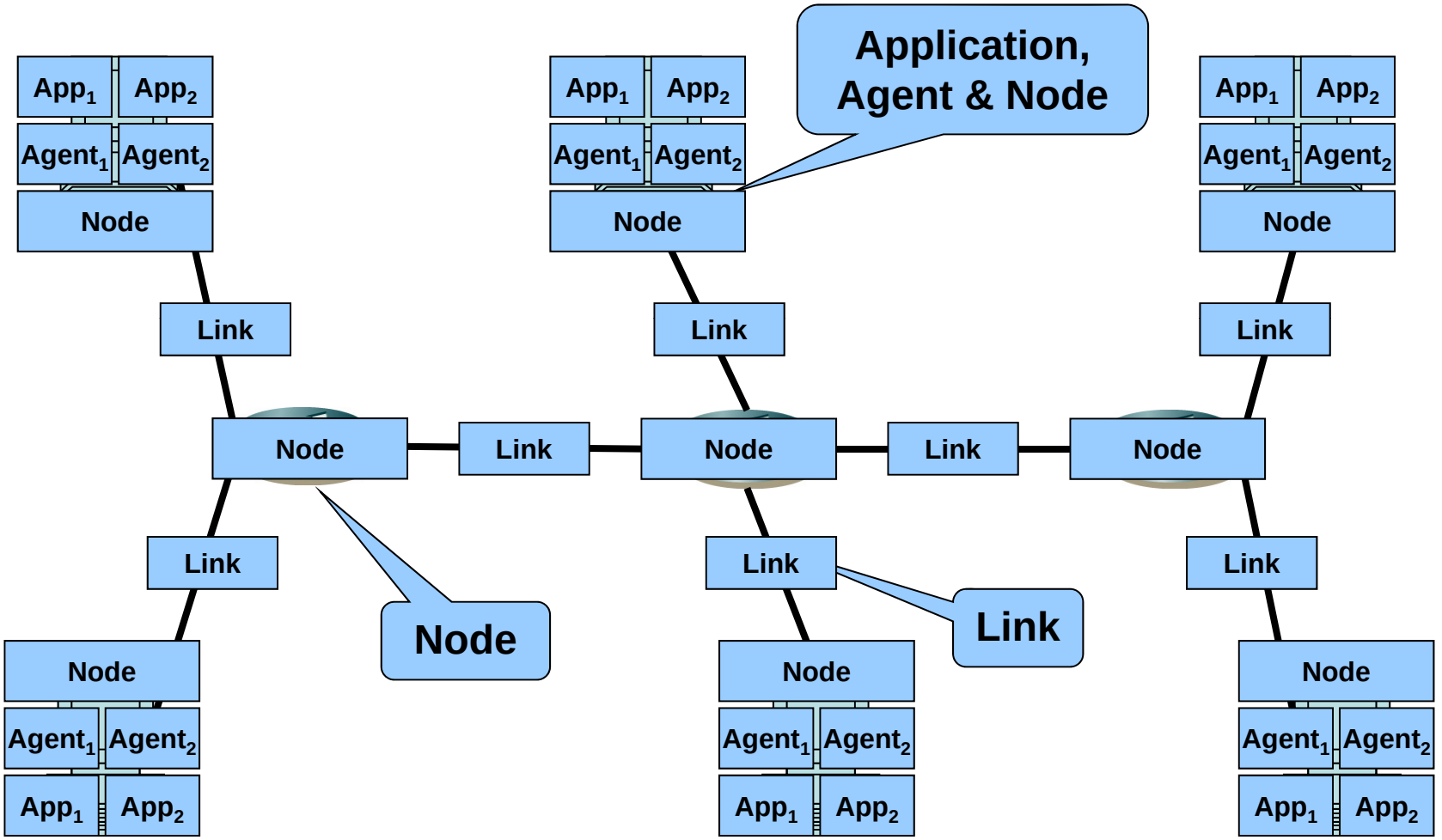


Source: [2]

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

The main players in network simulations

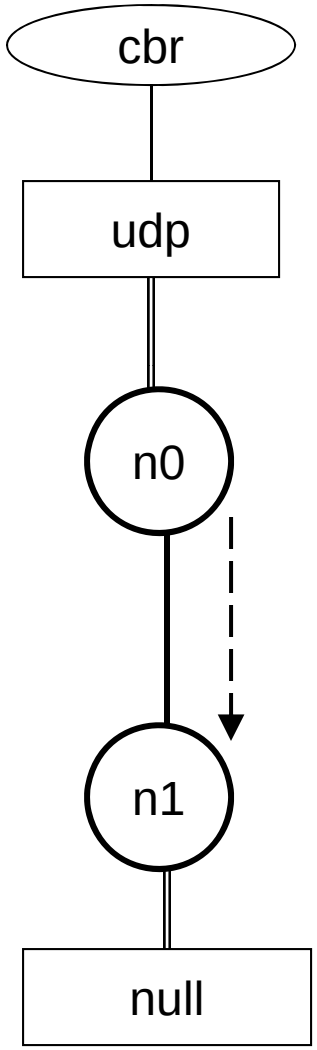


Source: [9]

The first example (wired network)

- Very simple topology
  - Two nodes are directly connected

Layer	Specification
Application	CBR traffic
Transport	UDP
Network	Direct connection (no routing)
MAC & PHY	1 Mb/s with delay 50ms



Source: [10]

The first example (cont'd)

- How to build the network
  - A bottom-up approach

1- Create Nodes

2- Connect them by a link

3- Attach transporters to nodes

4- Create traffic and attach it to transporters

The first example (cont'd)

- All we need
  - Objects
    - Nodes, links, traffic, …
  - Attaching and connecting objects
- Are ready in NS
- Some of them are available through the method of the "Simulator object"
- Some of them are directly accessible

The first example: Step 0

- Creating the “Simulator” object

```
set ns [new Simulator]
```

The first example: Step 1

- Creating nodes

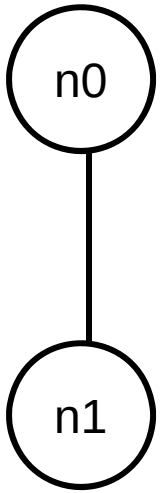
```
set n0 [$ns node]
set n1 [$ns node]
```

The first example: Step 2

- Connecting nodes
  - By a link

```
$ns duplex-link $n0 $n1 1Mb 50ms DropTail
```

  - 1Mb is the bandwidth
  - 50ms is the delay
  - DropTail is the queue type



The first example: Step 3

- Transport layer communication
- Create objects of transport layer

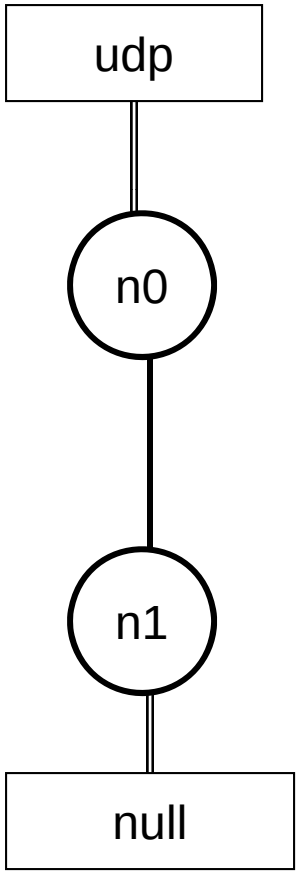
```
set udp [new Agent/UDP]
set null [new Agent/Null]
```

- Attach transporters to nodes

```
$ns attach-agent $n0 $udp
$ns attach-agent $n1 $null
```

- Connect transporters

```
$ns connect $udp $null
```



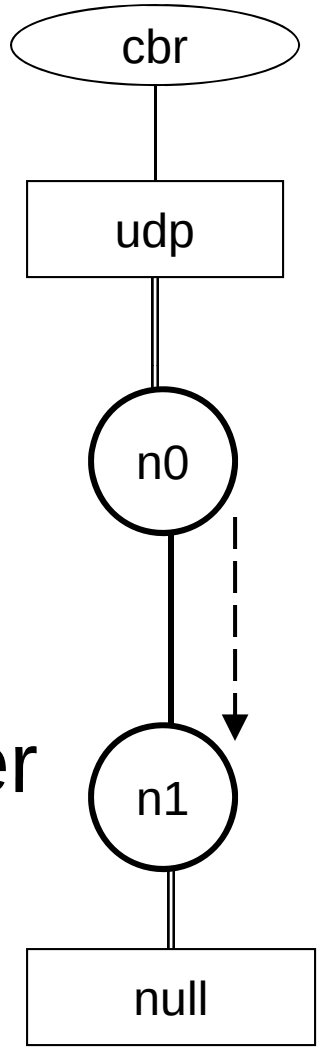
The first example: Step 4

- Traffic
- Create traffic generator

```
set cbr [new
        Application/Traffic/CBR]
```

- Attach traffic generator to transporter

```
$cbr attach-agent $udp
```



The first example: Step 5

- We need to monitor the simulation event
  - This is the output of simulation
- In the NS terminology
  - We should trace the events
  - Insert immediately after scheduler!

```
set trFile [open out.tr w]
$ns trace-all $trFile

set namFile [open out.nam w]
$ns namtrace-all $namFile
```

The first example: Step 5

- Schedule events, start and stop the simulation

```
proc finish {} {
    global ns trFile namFile
    $ns flush-trace
    close $namFile
    close $trFile
 }
$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop"
$ns at 5.0 "finish"
$ns run
```

```
proc finish {} {
   global ns trFile namFile
   $ns flush-trace
   close $namFile
   close $trFile
}
set ns [new Simulator]


set trFile [open out.tr w]
$ns trace-all $trFile
set namFile [open out.nam w]
$ns namtrace-all $namFile


set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1  1Mb
   50ms DropTail
```

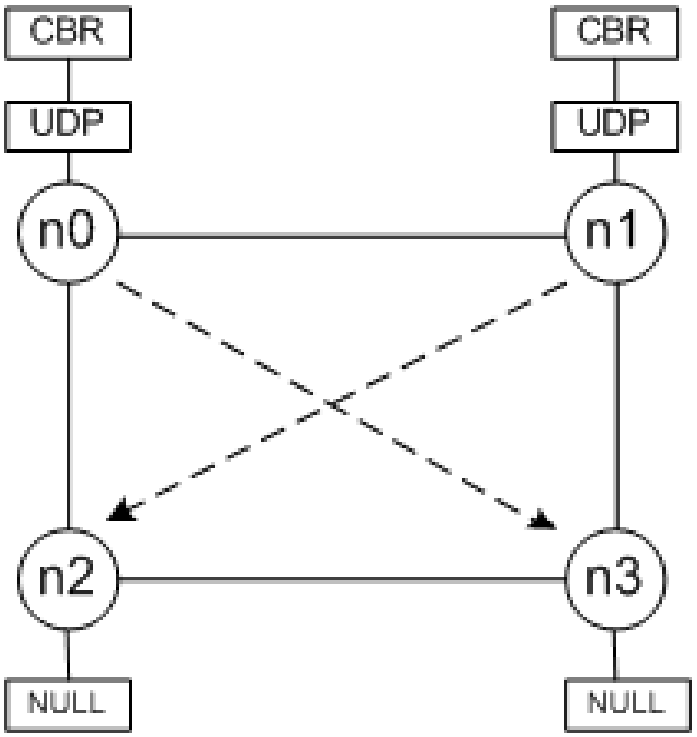
```
set udp [new Agent/UDP]
set null [new Agent/Null]


$ns attach-agent $n0 $udp
$ns attach-agent $n1 $null
$ns connect $udp $null


set cbr [new
   Application/Traffic/CBR]
$cbr attach-agent $udp


$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop"
$ns at 5 "finish"


$ns run
```

Simulation script structure

- [procedures]
- Create the simulation object
- Enable traces
- Create nodes
- Create links
- Create traffic (transporters and traffic src)
- Schedule events
- Fire the simulation

Lab. Exercise

- Extend the example to implement the following topology



What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Simulation results

- Two main types of simulation results
- NAM files
  - A text file for the "**nam**" software
  - The visual output of simulation
  - Is useful for demonstration
- Trace files
  - A text file with specific format
  - The raw data of simulation
    - Event records

Visual simulation results

- nam: Network AniMator
  - Packet level animation

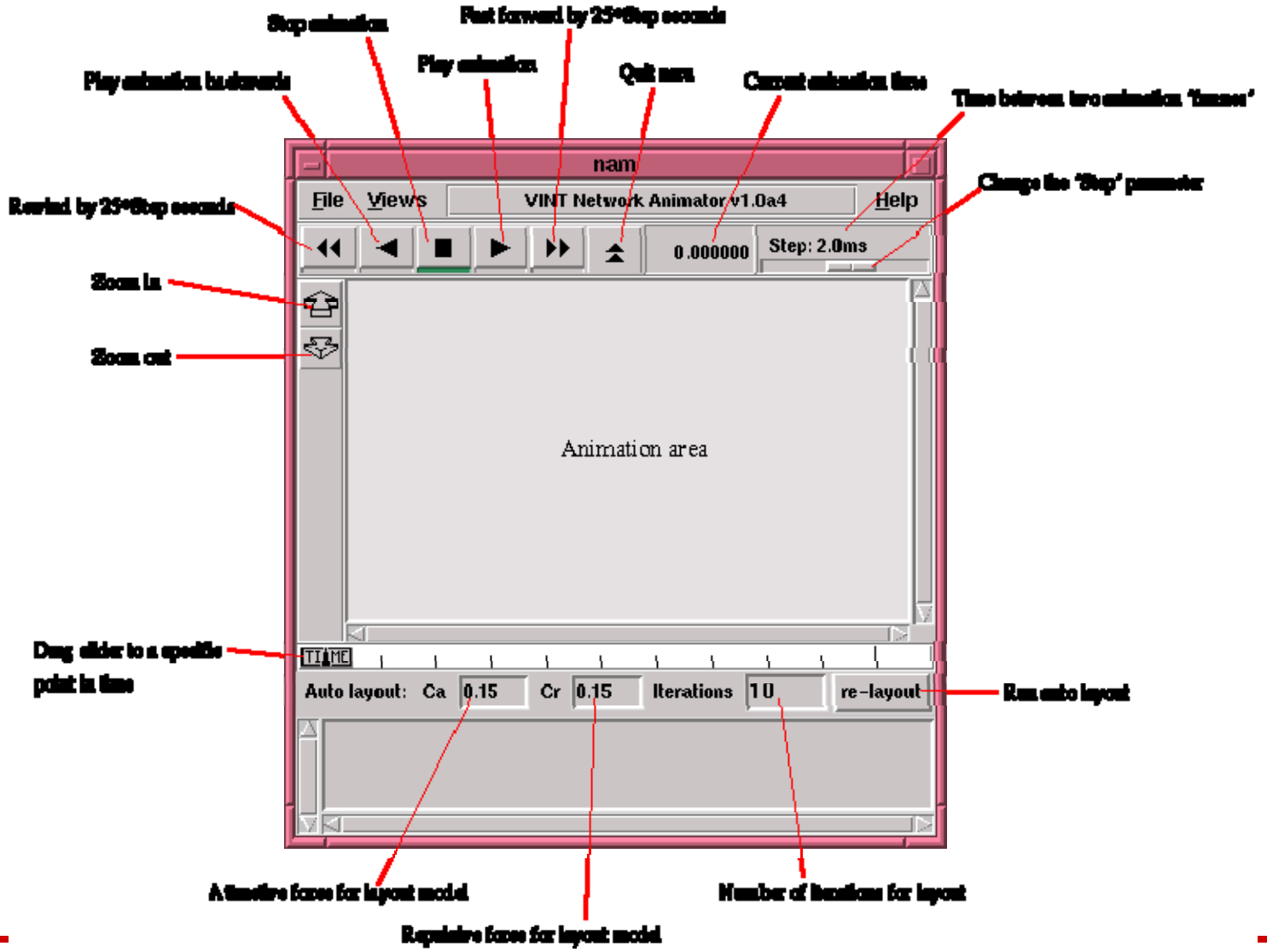
- NS records all needed info for nam

```
$ns namtrace-all $namFile
```

- **nam** reads the file and visualizes it

```
nam namfile.nam
```

nam



Source: [11]

Customize the visualization

- Node settings
  - Color, shape, …
- Link settings
  - Color, label, …
- Topology
  - Topology layout, orientation, …
- Misc
  - Protocol state, …

Source: [12]

Customize the visualization: Nodes

- Color

```
$node color red
```

- Shape

```
$node shape box
```

  - circle, box, hexagon

- Label (single string)

```
$n0 label "web cache"
```

Customize the visualization: Links

- Color

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

- Label

```
$ns duplex-link-op $n0 $n1 label "link01"
```

Customize the visualization: Others

- Topology

```
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns duplex-link-op $n(2) $n(3) orient down
```

- Animation rate

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

- Traffic color

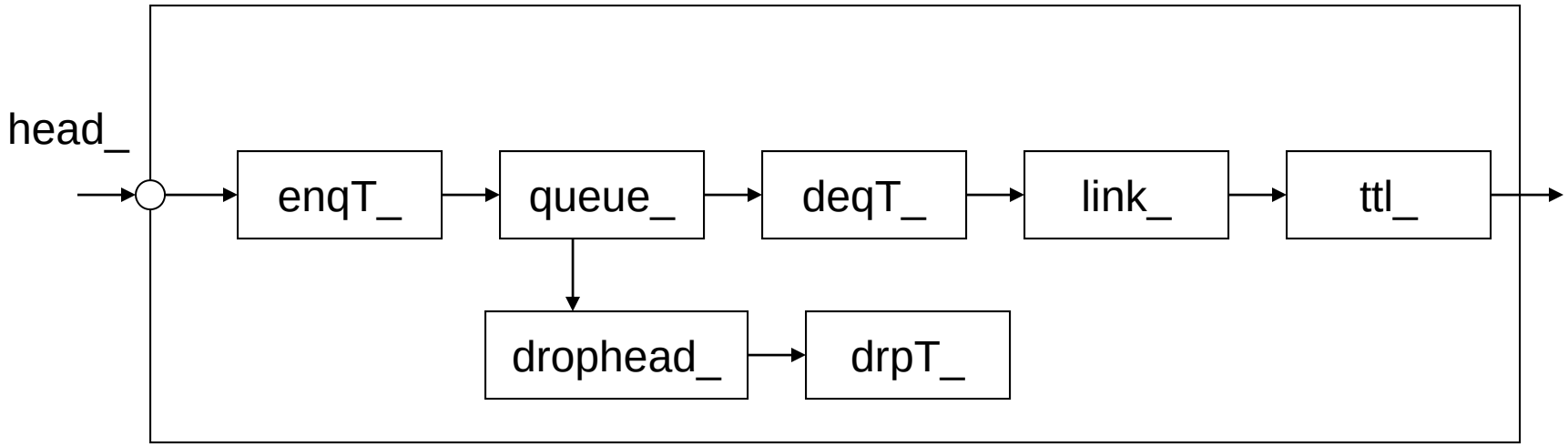
```
$ns color <fid> <color name>
```

Lab. Exercise

- Make your old topology fancy as much as you can ;-)

Tracing

- The implementation of a unidirectional link in NS-2



Trace file format

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop (at queue)

src_addr : node.port (3.0)
dst_addr : node.port (0.0)

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Source: [10]

How to work with trace files

- The excellent tools
  - **`awk`** and **`grep`**

- **`awk`** is in fact a programming language
  - An interpreted language
  - **`gawk`** is the GNU version

- **`grep`** is a pattern matching program

awk basics: 1

- The basic entity is a line
  - **`$0`** is the line
- A line is split into multiple fields
  - Default field separator is ‘ ‘
  - **`$1`** is the 1th field, **`$2`** is the 2th field, …
  - **`NF`** is the number of fields
- Syntax
  - Very similar to C (thanks to Prof. Kernighan)
    - if-else, while, ++, ==, !=, …
  - No variable declaration (similar to tcl)

awk basics: 2

- Three main building blocks
- The begin block
  - **`BEGIN{ }`**
  - Runs only one time at the beginning
- The main block
  - **`{ }`**
  - Runs per line
- The end block
  - **`END{ }`**
  - Runs only one time at the end

awk basics: 3 (invocation)

- Using script file
  - We don’t discuss here

- Through piping
  - **`cat out.tr | gawk 'BEGIN{} {} END{}'`**
  - The begin block runs at the beginning
  - The main block runs for each line of the file
  - The end block runs at the end

awk by example

- Print all the events of packet receptions

```
cat out.tr | gawk '{if($1 == "r") print $0}'
```

- Print all events related to packet 0

```
cat out.tr | gawk '{if($NF == "0") print $0}'
```

- Print total transmitted traffic on the link between n0 and n1

```
cat out.tr | gawk '{if(($1=="-") && ($3=="0")
  && ($4=="1")) sum+=$6} END{print sum}'
```

awk by example (cont'd)

- Compute the throughput between n0 and n1
  - It is identified by flow id = 0

```
cat out.tr | gawk '{if(($1 == "r") &&
($8 == "0")) print $0}' | gawk
'{if(s==0){s=1; start=$2}; bytes+=$6;
time=$2} END{print bytes*8/(time-
start)}'
```

awk by example (cont'd)

- Script file to compute the throughput of given flow

```
BEGIN{  print "Flow ID = " fid;  }
{
   if(($1 == "r") && ($8 == fid)){
      if(s==0){
          s=1;  start=$2
      }
      bytes+=$6; time=$2
    }
}
END{ print "Throughput of flow " fid " = " bytes*8/(time-
   start) }
```

awk by example (cont'd)

- How to use the script file

```
cat out.tr | gawk -v fid=0 -f
  throughput.awk
```

- **`-v var=val`**: define variable **`var`** and assign value **`val`** to it which is passed to the script
- **`-f filename`**: specify the script file name

Advanced tracing

- Per link tracing

```
$ns trace-queue $n0 $n1 $fileID
```

  - Trace only the events at the link between n0 & n1
  - Should be after creating the link

- Filtered trace file

```
set cbrFile [open "| grep \"cbr\" out.tr" w]
$ns trace-all $cbrFile
```

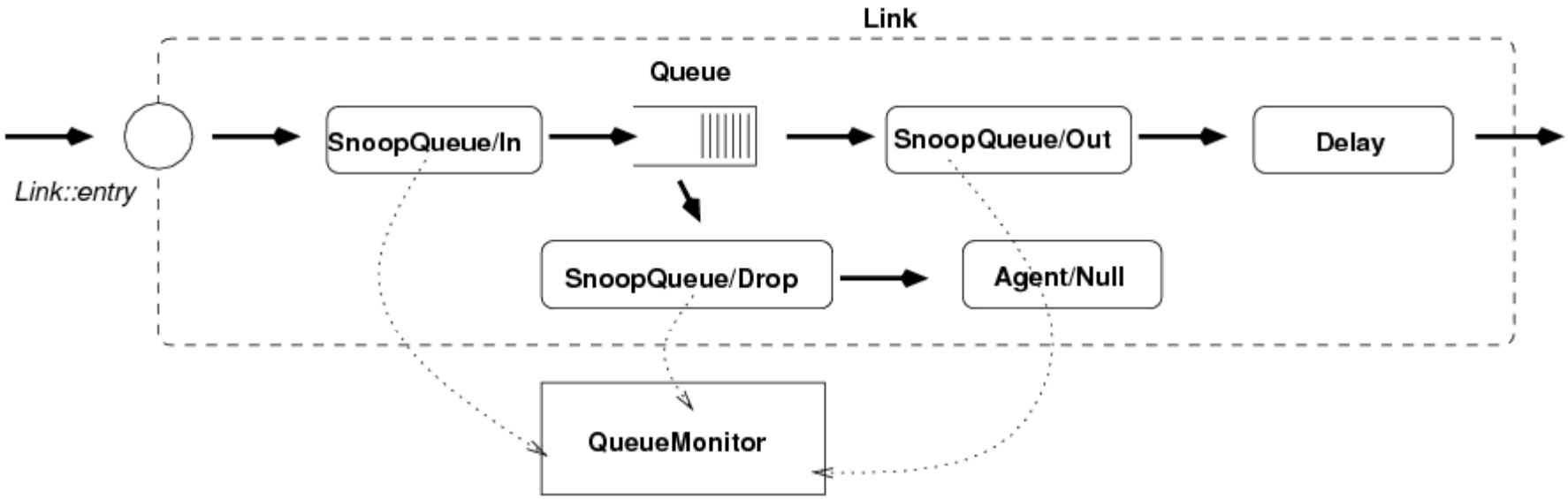
  - Trace only the events of cbr traffic

Monitors

- Useful tools to find detail information about queues
  - Average length, current length, …
- Can be
  - Link based
    - Aggregate traffic on the link is monitored
  - Flow based
    - A specific flow is monitored

Monitors: Link based monitoring

- In fact it is Queue monitor



Source: [17]

Monitors: Link based monitoring

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$ns duplex-link $n0 $n1 1Mb   1ms DropTail
$ns duplex-link $n1 $n2 100Kb 1ms DropTail
set qmon [$ns monitor-queue $n1 $n2 "" 0.01]

set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n0 $udp
$ns attach-agent $n2 $null
$ns connect $udp $null
```

Monitors: Link based monitoring

```
set cbr [new  Application/Traffic/CBR]
$cbr attach-agent $udp
proc sample { } {
   global ns qmon
   set t [$ns now]
   puts "$t [$qmon set size_] [$qmon set pkts_] [$qmon set
   parrivals_] [$qmon set barrivals_]"
   $ns at [expr $t + 1] "sample"
}
$ns at 0.5 "$cbr start"
$ns at 0 "sample"
$ns at 40.5 "$cbr stop"
$ns at 41 "exit"
$ns run
```

Monitors: Per flow monitoring

- Step1: Create monitor and attach to the link

```
set link01 [$ns link $n0 $n1]
set fmonitor [$ns makeflowmon Fid]
$ns attach-fmon $link01 $fmonitor
```

- Step 2: get statistics (e.g. in a sampling proc)

```
set fcla [$fmonitor classifier]
set flow [ $fcla lookup auto 0 0 1 ]
if { $flow != "" }
    set bytes [ $flow set barrivals_ ]
```

Monitors: Per flow monitoring

- An example of sampling proc

```
proc record {} {
        global f1 fmonitor ns
        set time 0.25
        set now [$ns now]
        set fcla [$fmonitor classifier]
        set flow [ $fcla lookup auto 1 0 3 ]
        if { $flow != "" } {
             set bytes [ $flow set barrivals_ ]
             puts $f1 "$now [expr $bytes/$time * 8]"
             $flow set barrivals_ 0
        }
         $ns at [expr $now+$time] "record"
}
```

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Parameters & State variables

- All the simulation objects
  - Nodes, links, traffic, …
- Have many parameters and state variables
  - Parameters: To control the object
    - Customize the simulation
  - State variables: To monitor the simulation
    - Find out specific information

Node parameters

- There are many parameters about node
  - More than 20 parameters
- Most of them are related to wireless networks
  - We will discuss later

Link parameters

`$ns simplex-link <node1> <node2> <bw> <delay> <qtype> <args>`

- Unidirectional link

`$ns duplex-link <node1> <node2> <bw> <delay> <qtype> <args>`

- Bidirectional link

`$ns lossmodel <lossobj> <from> <to>`

- Insert a loss module in regular links

`$ns queue-limit <node1> <node2> <size>`

- Specify the queue length

Agent parameters

- Agents are the end-points of traffic flow
  - Producer or consumer of packets
- Transport protocols are implements by agents
- Implemented agents in NS
  - TCP: TCP, Reno, Newreno, FullTcp, …
  - UDP
  - RTP
  - Message
  - Null
  - …

TCP agents

- `set tcp [new Agent/TCP]  #Sender`
- `set tcp [new Agent/TCPSink] #Reciver`
- `$tcp set window_  <window>`
- `$tcp set windowInit_  <window init>`
- `$tcp set packetSize_  <packet size>`
- `$tcp set ssthresh_  <slow start thr>`
- `$tcp set ack_`
- `$tcp set cwnd_`
- `$tcp set awnd_`

UDP agent

- This is a one-way traffic source
- Since there is not any ACK in UDP
  - The sink of the UDP traffic is the NULL agent

- **`set udp0 [new Agent/UDP]`**
- **`$udp set packetSize_ <pktsize>`**

Applications: Exponential

- `set exp [new Application/Traffic/Exponential]`
- Variables
  - **`packetSize_`**
    - constant size of packets generated.
  - **`burst_time_`**
    - average on time for generator.
  - **`idle_time_`**
    - average off time for generator.
  - **`rate_`**
    - sending rate during on time.

Applications: CBR

- **`set cbr [new Application/Traffic/CBR]`**
- **`$cbr start / stop`**
- Variables
  - **`packetSize_`**
    - constant size of packets generated.
  - **`maxpkts_`**
    - maximum number of packets to send
  - **`random_`**
    - whether or not noise, default is off.
  - **`rate_`**
    - sending rate during on time.

Applications: FTP

- **`set ftp [new Application/FTP]`**
- **`$ftp start/stop`**
- **`$ftp produce`**

- Variables
  - **`maxpkts_`**
    - maximum number of packets to send

Random numbers

- Pseudo-Random number generation
  - Is essential in computer simulations
  - Is (completely) supported in NS
- How to
  - Step 1
    - Create the Generator object and initialize seed
  - Step 2
    - Get random numbers from the generator

  or
    - Create a random variable with a distribution
      - We do not discuss here

Random numbers (cont'd)

- Creating the generator

```
set nsRNG [new RNG]
```

- Initializing the seed

```
$nsRNG seed 2

$nsRNG seed 0   # Random sequence per
                  invocation
```

Random numbers (cont'd)

```
$nsRNG normal mean std

$nsRNG lognormal mean std

$nsRNG exponential mean

$nsRNG uniform min max

$nsRNG integer max
```

Routing

- In wired networks, by default
  - The minimum hop is the routing path
  - The path is static

```
$ns rtproto Static
```

- Distance vector routing is also supported

```
$ns rtproto DV
```

- Link-state routing is also supported

```
$ns rtproto LS
```

Network dynamics

- To simulate link/node failure
  - Specify up/down times
- Manual

```
$ns rtmodel-at 20.0 down $n1 $n2
$ns rtmodel-at 50.0 up $n1 $n2
```

  - Or

```
$ns rtmodel-at 20.0 down $n2
$ns rtmodel-at 50.0 up $n2
```

Network dynamics

- Fixed intervals

```
$ns rtmodel Deterministic {10 5 3 50} $n0
```

  - 10: Start Time
  - 5: Up interval
  - 3: Down interval
  - 50: Stop time

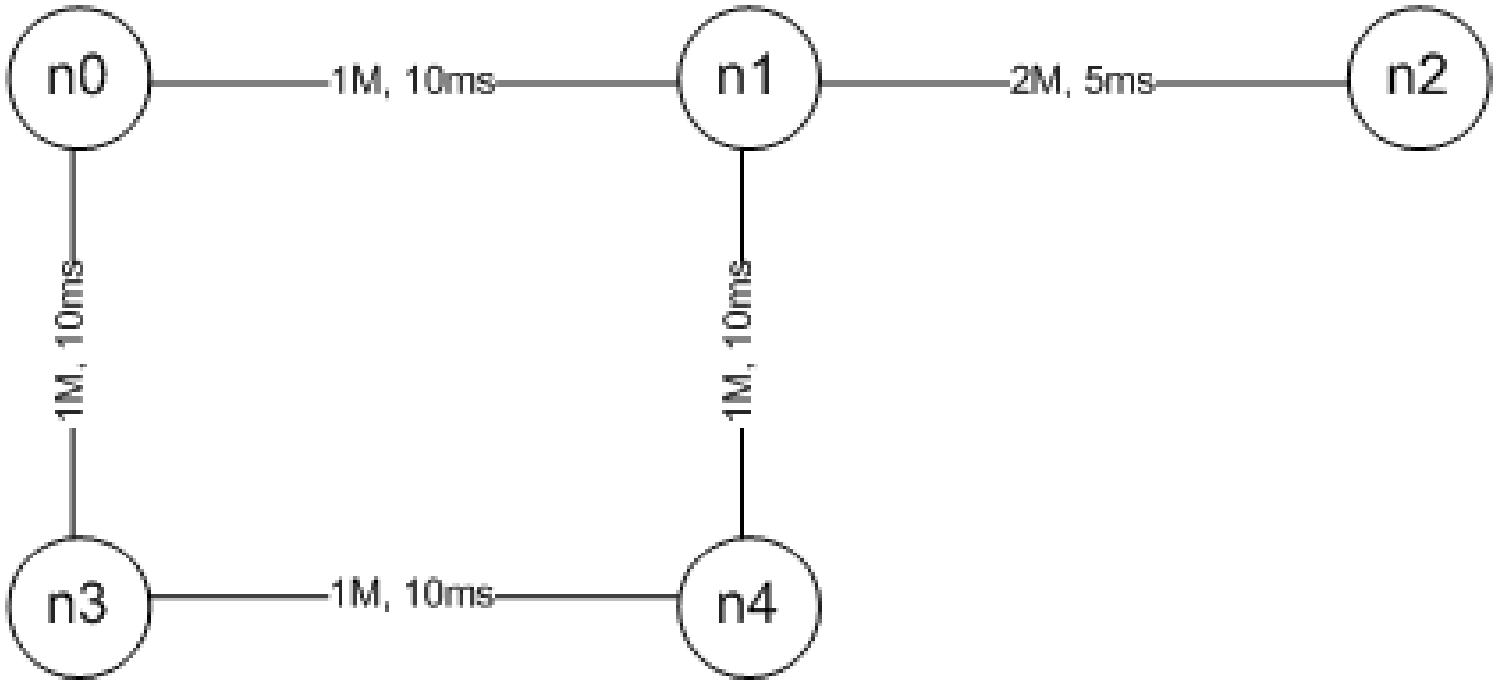
- Random intervals (exponential distribution)

```
$ns rtmodel Exponential {10 5 3 50} $n0
```

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Example: Simulation topology



- Two traffic stream
  - CBR over UDP: n0 → n2, 700kb/s, 1500
  - FTP over TCP: n4 → n2

```
set ns [new Simulator]

set traceAll [open out.tr w]
$ns trace-all $traceAll
set namFile [open out.nam w]
$ns namtrace-all $namFile

for {set i 0} { $i < 5} {incr i} {
   set n($i) [$ns node]
}
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 2Mb 5ms DropTail
$ns duplex-link $n(1) $n(4) 1Mb 10ms DropTail
$ns duplex-link $n(3) $n(4) 1Mb 10ms DropTail
```

```
set udp [new Agent/UDP]
$ns attach-agent $n(0) $udp
set nullsink [new Agent/Null]
$ns attach-agent $n(2) $nullsink
$ns connect $udp $nullsink
$udp set fid_ 100
$ns color 100 Red

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packet_size_ 500
$cbr set rate_ 700Kb
```

```
set tcp [new Agent/TCP]
$ns attach-agent $n(4) $tcp
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n(2) $tcpsink
$ns connect $tcp $tcpsink
$tcp set fid_ 200
$ns color 200 Blue

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

Example: Monitors

- Monitor bandwidth on n0-n1
- Monitor dropped packets on n4-n1
- Monitor both streams at n1-n2

```
proc BWmonitor {qmonitor window output} {
  global ns
  set t [$ns now]
  set bytes [$qmonitor set barrivals_ ]
  $qmonitor set barrivals_ 0
  puts $output "$t [expr $bytes * 8 / $window]"
  $ns at [expr $t + $window] "BWmonitor $qmonitor
  $window $output"
}
set BWout [open n0-n1-bw.tr w]
set bmon [$ns monitor-queue $n(0) $n(1) "" 0.05]
set window1 1
```

```
proc DropMonitor {qmonitor window output} {
  global ns
  set t [$ns now]
  set bytes [$qmonitor set pdrops_]
  puts $output "$t $bytes"
  $ns at [expr $t + $window] "DropMonitor
  $qmonitor $window $output"
}

set DropOut [open n4-n1-drop.tr w]

set dmon [$ns monitor-queue $n(4) $n(1) "" 0.05]
set window2 1
```

```
proc flowBW {flowMonitor sport dport fid window output} {
   global ns
   set t [$ns now]
   set fcla [$flowMonitor classifier]
   set flow [ $fcla lookup auto $sport $dport $fid ]
   if { $flow != "" } {
        set bytes [ $flow set barrivals_ ]
        puts $output "$t [expr $bytes * 8 / $window]“
        $flow set barrivals_ 0
   }
   $ns at [expr $t + $window] "flowBW $flowMonitor $sport $dport $fid
   $window $output"
}
set udpBW [open udp-bw.tr w]
set tcpBW [open tcp-bw.tr w]
set nlink [$ns link $n(1) $n(2)]
set fmonitor [$ns makeflowmon Fid]
$ns attach-fmon $nlink $fmonitor
set window3 1
```

Example: Finish and startup

- A create procedure to closed files
- At the startup
  - Schedule start of traffic sources
  - Schedule monitoring procedures
- At the end
  - Stop traffic generators

```
proc finish {} {
   global traceAll namFile BWout DropOut udpBW tcpBW
   close $traceAll
   close $namFile
   close $BWout
   close $DropOut
   close $udpBW
   close $tcpBW
   exit 0
}
```

```
$ns at 0.0 "BWmonitor $bmon $window1 $BWout"

$ns at 0.0 "DropMonitor $dmon $window2 $DropOut"

$ns at 0.0 "flowBW $fmonitor [$udp set agent_port_]
  [$udp set dst_port_] [$udp set fid_] $window3 $udpBW"

$ns at 0.0 "flowBW $fmonitor [$tcp set agent_port_]
  [$tcp set dst_port_] [$tcp set fid_] $window3 $tcpBW"

$ns at 5.0 "$cbr start"

$ns at 5.0 "$ftp start"

$ns at 55.0 "$cbr stop"

$ns at 55.0 "$ftp stop"

$ns at 60.0 "finish"

$ns run
```

Example: Results

- Run the simulation
- Check the nam output
- Plot the monitored bandwidths and packet
  - MS Excel is easy to use

Example: Network dynamics

- Turn off link n0→n1 from 20s to 30s

```
$ns rtmodel-at 20 down $n(0) $n(1)
$ns rtmodel-at 30 up $n(0) $n(1)
```

- Run and check results
  - Nam and plots

Example: Routing

- Enable routing and turn off/on the link

```
$ns rtproto LS
```

- Check the results
  - Both nam and plots

Example: Extending

- Try your own modifications, e.g.,
- Different link bandwidth
- Other traffic streams
- Random link failures
- …

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Wireless simulation

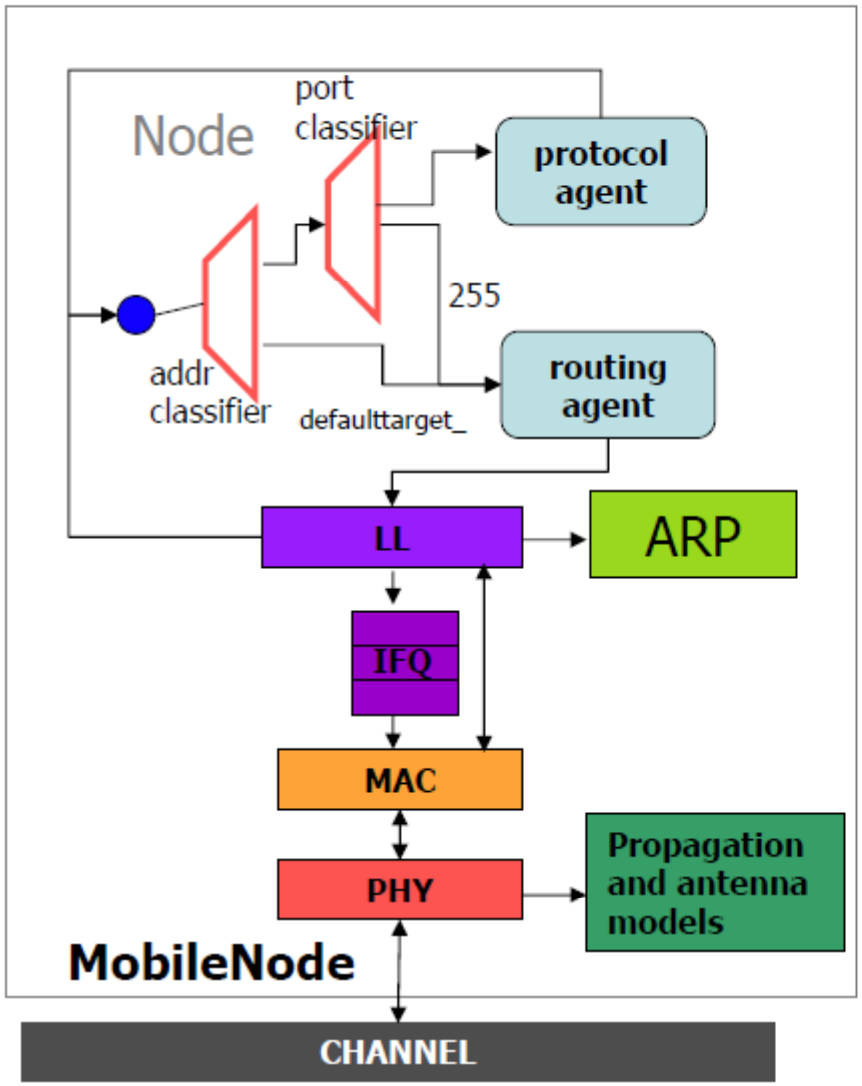
- Fundamental differences between wired and wireless simulations
- There is no link
  - (flow & link monitor ???)
- Nodes can be mobile
  - Very complicated routing
- Nodes may battery-powered

Wireless simulation (cont'd)

- Many parameters should be customized
  - MAC layer parameters
  - PHY layer parameters
  - Antenna parameters
- Node locations need to be specified
- A different trace format
- Most researches on the MAC layer

Wireless simulation (cont'd)

- Some similarities between wired and wireless simulations
  - Agents
    - TCP, UDP, …
  - Applications
    - CBR, FTP, …
  - The trace commands
    - Wireless is also supported by nam
  - Startup initialization and finish procedures



Classifier: Forwarding

Agent: Protocol Entity

Node Entry

LL — **LL**: Link layer object

IFQ — **IFQ**: Interface queue

MAC — **MAC**: Mac object

PHY — **PHY**: Net interface

Prop/ant — Radio propagation/ antenna models

Source: [6]

GOD

- Who is the GOD?
  - General Operations Director
- An omniscient observer
- Stores smallest number of hops from one node to another
- Optimal case to compare routing protocol performance
- Generated by scenario file

Source: [6]

Wireless simulation: The first example

- Network topology

The first example: Parameters

```
set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/Simple
set val(ifq)            Queue/DropTail/PriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(ifqlen)         50
set val(nn)             2
set val(rp)             DumbAgent
set val(x)              250
set val(y)              250
```

The first example: Initialization

```
set ns [new Simulator]

$ns use-newtrace

set traceFile [open out.tr w]

$ns trace-all $traceFile

set namFile [open out.nam w]

$ns namtrace-all-wireless $namFile
  $val(x) $val(y)
```

The first example: Wireless topology

- Set up topography object

```
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
```

- Create God

```
create-god $val(nn);    # :P
```

- Create channel

```
set chan_ [new $val(chan)]
```

The first example: Nodes

```
$ns node-config -adhocRouting $val(rp) -llType $val(ll) \
      -macType $val(mac)  -ifqType $val(ifq)  \
      -ifqLen $val(ifqlen) -antType $val(ant) \
      -propType $val(prop) -phyType $val(netif) \
      -topoInstance $topo a-agentTrace ON \
      -routerTrace ON -macTrace ON \
      -movementTrace ON -channel $chan_

for {set i 0} {$i < $val(nn)} {incr i} {
    set n($i) [$ns node]
    $n($i) random-motion 0
}
```

The first example: Node positions

```
$n(0) set X_ 20.0
$n(0) set Y_ 20.0
$n(0) set Z_ 0.0
$n(1) set X_ 120.0
$n(1) set Y_ 20.0
$n(1) set Z_ 0.0
# These are needed by nam
$ns at 0.0 "$n(0) setdest [$n(0) set X_] [$n(0)
  set Y_] 0.0"
$ns at 0.0 "$n(1) setdest [$n(1) set X_] [$n(1)
  set Y_] 0.0"
```

The first example: Traffics

```
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns attach-agent $n(0) $tcp
$ns attach-agent $n(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

The first example: Scheduling & Fire

```
proc stop {} {
    global ns traceFile namFile
    $ns flush-trace
    close $traceFile; close $nameFile
    exit 0
}

$ns at 0.5 "$ftp start"
$ns at 5.0 "$ftp stop"
$ns at 6.0 "stop"
$ns run
```

The first example: Run & Result

- The script is ready
- Run it
- View the simulation result in nam

Wireless simulation (cont'd)

- Trace file format
  - Two formats
    - Old
    - New
- We use the new one
- Enable the new tracing format

```
$ns use-newtrace
```

The new wireless trace file format

- General Information
- Type
  - **s**: Send, **r**: Receive, **d**: Drop, **f**: Forward
- **-t** time
- **-Hs**: id for this node
- **-Hd**: id for next hop towards the destination
  - -1 means that the packet is a broadcast packet
  - -2 means that the destination node has not been set
    - packets that are passed between the agent (-Nl AGT) and routing (-Nl RTR) levels

The new wireless trace file format (cont'd)

- Node information
- **-Ni**: node id
- **-Nx**: node's x-coordinate
- **-Ny**: node's y-coordinate
- **-Nz**: node's z-coordinate
- **-Ne**: node energy level
- **-Nl**: trace level, such as AGT, RTR, MAC
- **-Nw**: reason for the event (ref. NS manual)

The new wireless trace file format (cont'd)

- Mac layer information
- **`-Ma`**: duration
- **`-Md`**: dst's ethernet address
- **`-Ms`**: src's ethernet address
- **`-Mt`**: ethernet type

The new wireless trace file format (cont'd)

- IP level information
- **`-Is`**: source address.source port number
- **`-Id`**: destination address.dest port number
- **`-It`**: packet type
- **`-Il`**: packet size
- **`-If`**: flow id
- **`-Ii`**: unique id
- **`-Iv`**: ttl value

The new wireless trace file format (cont'd)

- Address Resolution Protocol information
- **`-P arp`**
- **`-Po`**: ARP Request/Reply
- **`-Pm`**: src mac address
- **`-Ps`**: src address
- **`-Pa`**: dst mac address
- **`-Pd`**: dst address

The new wireless trace file format (cont'd)

- CBR traffic information
- **`-P cbr`**
- **`-Pi`**: sequence number
- **`-Pf`**: how many times this pkt was forwarded
- **`-Po`**: optimal number of forwards

The new wireless trace file format (cont'd)

- TCP traffic information
- **`-P tcp`**
- **`-Ps`**: seq number
- **`-Pa`**: ack number
- **`-Pf`**: how many times this pkt was forwarded
- **`-Po`**: optimal number of forwards

Working with trace files

- A simple & (maybe) useful awk script

```
{
  if(($1 == type) &&
     ($9 == node) &&
     ($19 == level) &&
     (index($0, "-It "traffic) > 0)){
     for(i=1; (i <= NF) && ($i != "-"field); i++) ;
     print $3 " "$(i+1);
  }
}
```

Working with trace files (cont'd)

- The size of TCP packets sent by MAC node 0

```
cat out.tr | gawk -v type=s -v node=0 -v
  traffic=tcp -v level=MAC -v field=Il -f

  extract-wireless.awk
```

- The size of TCP packets received by the TCP agent in node 1

```
cat out.tr | gawk -v type=r -v node=1 -v
  traffic=tcp -v level=AGT -v field=Il -f
  extract-wireless.awk
```

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

More details

- Similar to wired simulation objects
- The wireless components have many configurable parameters
- What we will see
  - Node configuration
  - MAC protocols
  - Routing
  - Node mobility

Node config

wiredRouting	ON, OFF	OFF
llType	LL, LL/Sat	""
macType	Mac/802_11, Mac/Csma/Ca, Mac/Sat, Mac/Sat/UnslottedAloha, Mac/Tdma	""
ifqType	Queue/DropTail, Queue/DropTail/PriQueue	""
phyType	Phy/WirelessPhy, Phy/Sat	""
wireless-oriented		
adhocRouting	DIFFUSION/RATE, DIFFUSION/PROB, DSDV, DSR, FLOODING, OMNIMCAST, AODV, TORA, PUMA	""
propType	Propagation/TwoRayGround, Propagation/Shadowing	""
propInstance	Propagation/TwoRayGround, Propagation/Shadowing	""
antType	Antenna/OmniAntenna	""
channel	Channel/WirelessChannel, Channel/Sat	""
topoInstance	<topology file>	""
mobileIP	ON, OFF	OFF
energyModel	EnergyModel	""
initialEnergy	<value in Joules>	""
rxPower	<value in W>	""
txPower	<value in W>	""
idlePower	<value in W>	""
agentTrace	ON, OFF	OFF
routerTrace	ON, OFF	OFF
macTrace	ON, OFF	OFF
movementTrace	ON, OFF	OFF
errProc	UniformErrorProc	""
FECProc	?	?
toraDebug	ON, OFF	OFF

MAC protocols

- 802.11 variations
  - 802.11 DCF mode: **`Mac/802_11`**
  - 802.11 PCF mode (see **`$NS/ns-2.34/tcl/ex/802.11/infra.tcl`**)
  - More accurate version of 802.11: **`Mac/802_11Ext`** **`Phy/WirelessPhyExt`**
  - Multi-rate support: **`Mac/802_11/Multirate`**
- TDMA based
  - We do not discuss here

Routing protocols

- Some of the widely used routing algorithms
  - DSDV
  - DSR
  - TORA
  - AODV
  - …
- Routing protocol is specified by
  - **`-adhocRouting`** in **`node-config`**

802.11 parameters

- Many parameters (specified by the standard)
  - Contention window
  - SIF
  - …
- Default values for 802.11a is in

```
$NS/ns-2.34/tcl/ex/802.11/IEEE802.11a.tcl
```

PHY parameters

- Many parameters
  - Specified by the standard
  - Depends on wireless NIC
  - E.g.,
    - Carrier sensing threshold
    - Frequency
    - …
- Default values for 802.11a is in

`$NS/ns-2.34/tcl/ex/802.11/IEEE802.11a.tcl`

Node mobility

- Random motion
  - Set the random motion enable

```
$n($i) random-motion 1
```

  - Command the node to start movement

```
$n($i) start
```

- Manual
  - Set the position (in fact the movement direction)
  - Set the speed in this direction

```
$ns at 0.0 "$n(0) setdest 10 20 10”
$ns at 30.0 "$n(0) setdest 100 120 50"
```

Node mobility (cont'd)

- If node movements are complicated
  - There would be a mess in simulation scripts
- Put the movements in an scenario file
- Include the file in the simulation script

```
source movements-file.tcl
```

What We Will Learn

- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Example 1: Hidden node

- A simple topology to illustrate the hidden node problem
  - Using the IEEE802.11a setting
- 4 node
  - n0 → n1: A CBR traffic at rate 700kb
  - n2 → n3: A CBR traffic at rate 3Mb
- n1 is in the carrier sense range of n2
  - But n0 is not

Example 1 (cont'd)

- Very small carrier sensing rang
  - In IEEE802-11a.tcl

```
Phy/WirelessPhyExt set CSThresh_ 6.31e-1
```

- This is an unrealistic setting
  - But is useful for our purpose

```
source IEEE802-11a-smallCSR.tcl

set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhyExt
set val(mac)            Mac/802_11Ext
set val(ifq)            Queue/DropTail/PriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(ifqlen)         50
set val(nn)             4
set val(rp)             DumbAgent
set val(x)              1000
set val(y)              1000
```

```
set ns [new Simulator]
$ns use-newtrace
set traceFile [open out.tr w]
$ns trace-all $traceFile
set namFile [open out.nam w]
$ns namtrace-all-wireless $namFile $val(x) $val(y)

set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn);
set chan_ [new $val(chan)]

$ns node-config -adhocRouting $val(rp) -llType $val(ll) \
        -macType $val(mac)  -ifqType $val(ifq)  \
        -ifqLen $val(ifqlen) -antType $val(ant) \
        -propType $val(prop) -phyType $val(netif) \
        -topoInstance $topo a-agentTrace ON -routerTrace OFF \
        -macTrace ON -movementTrace ON -channel $chan_
```

```
for {set i 0} {$i < $val(nn)} {incr i} {
    set n($i) [$ns node]
    $n($i) random-motion 0
}

$n(0) set X_ 20.0
$n(0) set Y_ 20.0
$n(0) set Z_ 0.0

$n(1) set X_ 135.0
$n(1) set Y_ 135.0
$n(1) set Z_ 0.0

$n(2) set X_ 880.0
$n(2) set Y_ 880.0
$n(2) set Z_ 0.0
```

```
$n(3) set X_ 980.0
$n(3) set Y_ 980.0
$n(3) set Z_ 0.0

$ns at 0.0 "$n(0) setdest [$n(0) set X_] [$n(0) set Y_] 0.0"
$ns at 0.0 "$n(1) setdest [$n(1) set X_] [$n(1) set Y_] 0.0"
$ns at 0.0 "$n(2) setdest [$n(2) set X_] [$n(2) set Y_] 0.0"
$ns at 0.0 "$n(3) setdest [$n(3) set X_] [$n(3) set Y_] 0.0"
```

```
set udp01 [new Agent/UDP]
set sink01 [new Agent/Null]
$ns attach-agent $n(0) $udp01
$ns attach-agent $n(1) $sink01
$ns connect $udp01 $sink01
set cbr01 [new Application/Traffic/CBR]
$cbr01 set rate_ 700Kb
$cbr01 set packetSize_ 150b
$cbr01 attach-agent $udp01
```

```
set udp23 [new Agent/UDP]
set sink23 [new Agent/Null]
$ns attach-agent $n(2) $udp23
$ns attach-agent $n(3) $sink23
$ns connect $udp23 $sink23
set cbr23 [new Application/Traffic/CBR]
$cbr23 set rate_ 3Mb
$cbr23 attach-agent $udp23
```

```
proc stop {} {
    global ns traceFile namFile
    $ns flush-trace
    close $traceFile; close $namFile
    exit 0
}

$ns at 5 "$cbr01 start"
$ns at 25 "$cbr01 stop"
$ns at 5 "$cbr23 start"
$ns at 25 "$cbr23 stop"
$ns at 30.0 "stop"
$ns run
```

Example 1 (cont'd)

- Run the script
- Check the nam output
- Extract the packets sent by CBR traffic source in n0
- Extract the packets received by the Null sink in n1
- Use the "wireless-thr.awk" and plot the bandwidth of the extracted data

Example 1 (cont'd)

- Let n2 and n3 to move toward to n1
- The n2 node will become a hidden node if it is near the n1 node
- Command n2 and n3 to move away

Example 1 (cont'd): Movements

```
$ns at 0.0 "$n(2) setdest 400 400 70.0"
$ns at 0.0 "$n(3) setdest 500 500 70.0"

$ns at 20.0 "$n(2) setdest 850 850 100.0"
$ns at 20.0 "$n(3) setdest 950 950 100.0"
```

Example 1 (cont'd)

- Run the script
- Check the nam output
- Extract the packets sent by CBR traffic source in n0
- Extract the packets received by the Null sink in n1
- Use the "wireless-thr.awk" and plot the bandwidth of the extracted data

Hidden node example

- Extract dropped packets
- Two kinds of drop
  - MAC
  - IFQ
  - Merge the results

Lab Exercise

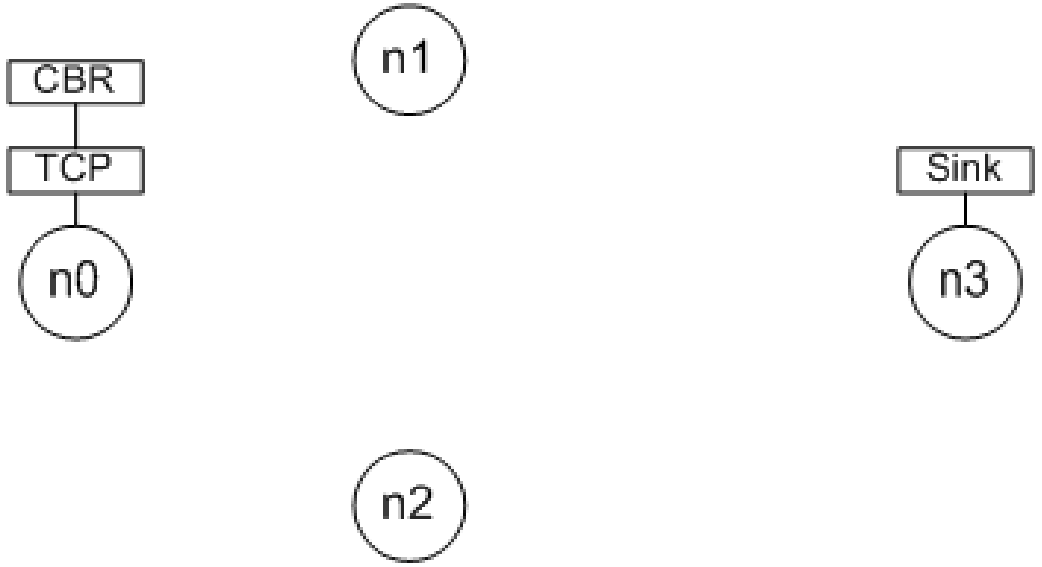
- Try to illustrate the exposed node problem

Example 2

- Simulate a multi-hop network
- The topology is dynamic
  - Even may be disconnected
- There is TCP stream
  - Analyze the behavior of TCP in this network

Example 2

- Topology



- We increase transmission power a bit

```
Phy/WirelessPhyExt set Pt_  0.002
```

```
source IEEE802-11a-highP.tcl

set val(chan)           Channel/WirelessChannel
...
set val(ifqlen)          50
set val(nn)             4
set val(rp)             AODV
set val(x)              400
set val(y)              1000
```

```
set ns [new Simulator]
$ns use-newtrace
...
...
for {set i 0} {$i < $val(nn)} {incr i} {
    set n($i) [$ns node]
    $n($i) random-motion 0
}
```

```
$n(0) set X_ 50.0
$n(0) set Y_ 500.0
$n(0) set Z_ 0.0

$n(1) set X_ 150.0
$n(1) set Y_ 550.0
$n(1) set Z_ 0.0

$n(2) set X_ 150.0
$n(2) set Y_ 450.0
$n(2) set Z_ 0.0

$n(3) set X_ 350.0
$n(3) set Y_ 500.0
$n(3) set Z_ 0.0
```

```
set tcp03 [new Agent/TCP]
$tcp03 set packetSize_ 500b
set sink03 [new Agent/TCPSink]
$ns attach-agent $n(0) $tcp03
$ns attach-agent $n(3) $sink03
$ns connect $tcp03 $sink03

set cbr03 [new Application/Traffic/CBR]
$cbr03 set rate_ 600kb
$cbr03 set packetSize_ 500b
$cbr03 attach-agent $tcp03
```

Example 2 (cont'd)

- Run the script
- Check the nam output
- Extract the packets sent by TCP traffic source in n0
- Extract the packets received by the TCP sink in n3
- Use the "wireless-thr.awk" and plot the bandwidth of the extracted data

Example 2 (cont'd)

- Add movements

```
$ns at 7.0  "$n(1) setdest [$n(1) set X_] 900 20"
$ns at 10.0 "$n(2) setdest [$n(2) set X_] 100 30"
$ns at 15.0 "$n(1) setdest [$n(1) set X_] 550 20”
```

- Rerun the simulation
- Check the new results
  - Is there any surprise ;-) ?
    - Check the sender traffic

Example 2 (cont'd)

- Reduce the traffic rate

```
$cbr03 set rate_ 100kb
```

- Rerun the simulation
- Check the new results
  - Can you explain the surprise?
  - More details (specially about the window size of TCP) can be found in Section 9 in [5]

What We Will Learn

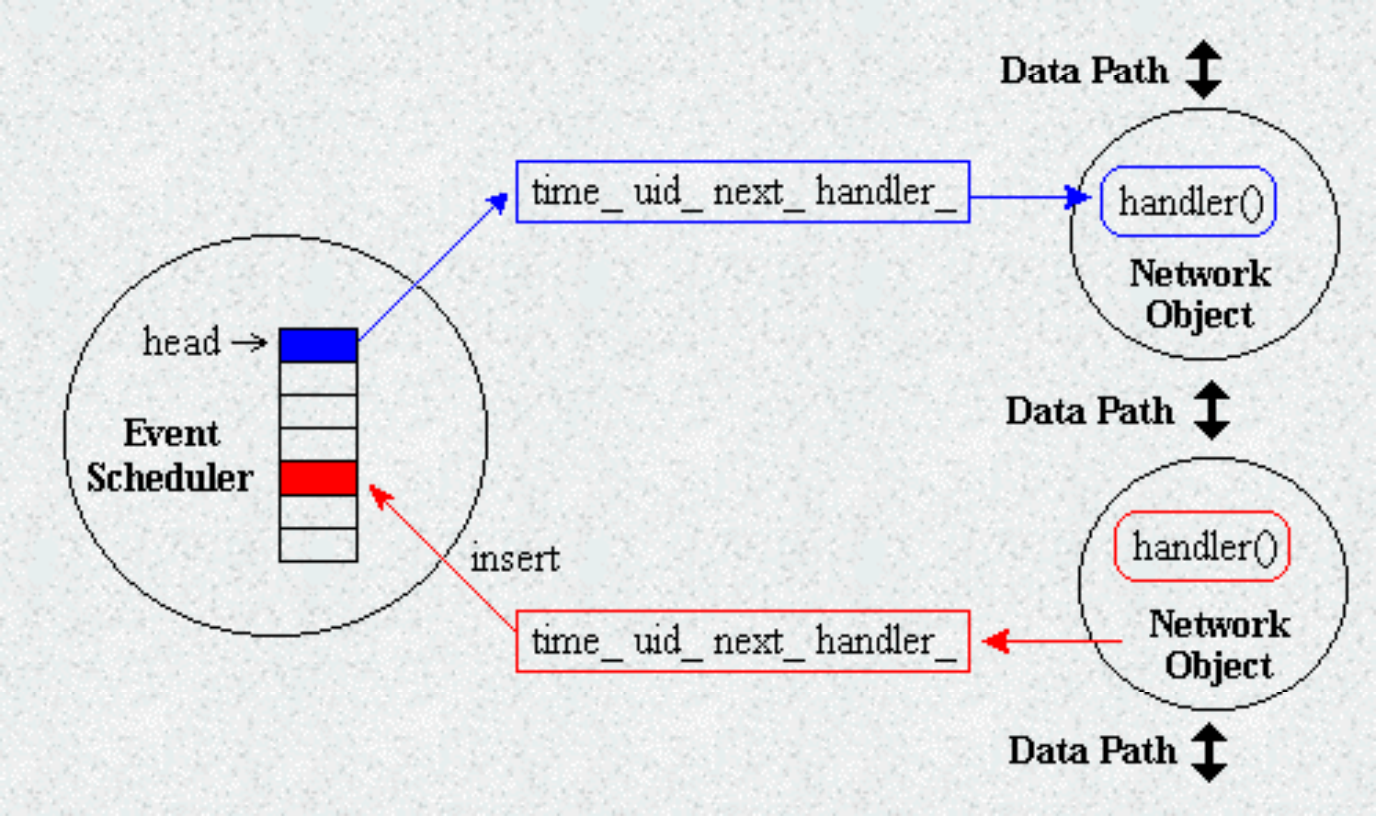
- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

NS-2 Internal implementation

- Discrete event scheduler
- Network topology
- Routing
- Transport
- Application
- Packet Flow
- Packet Format

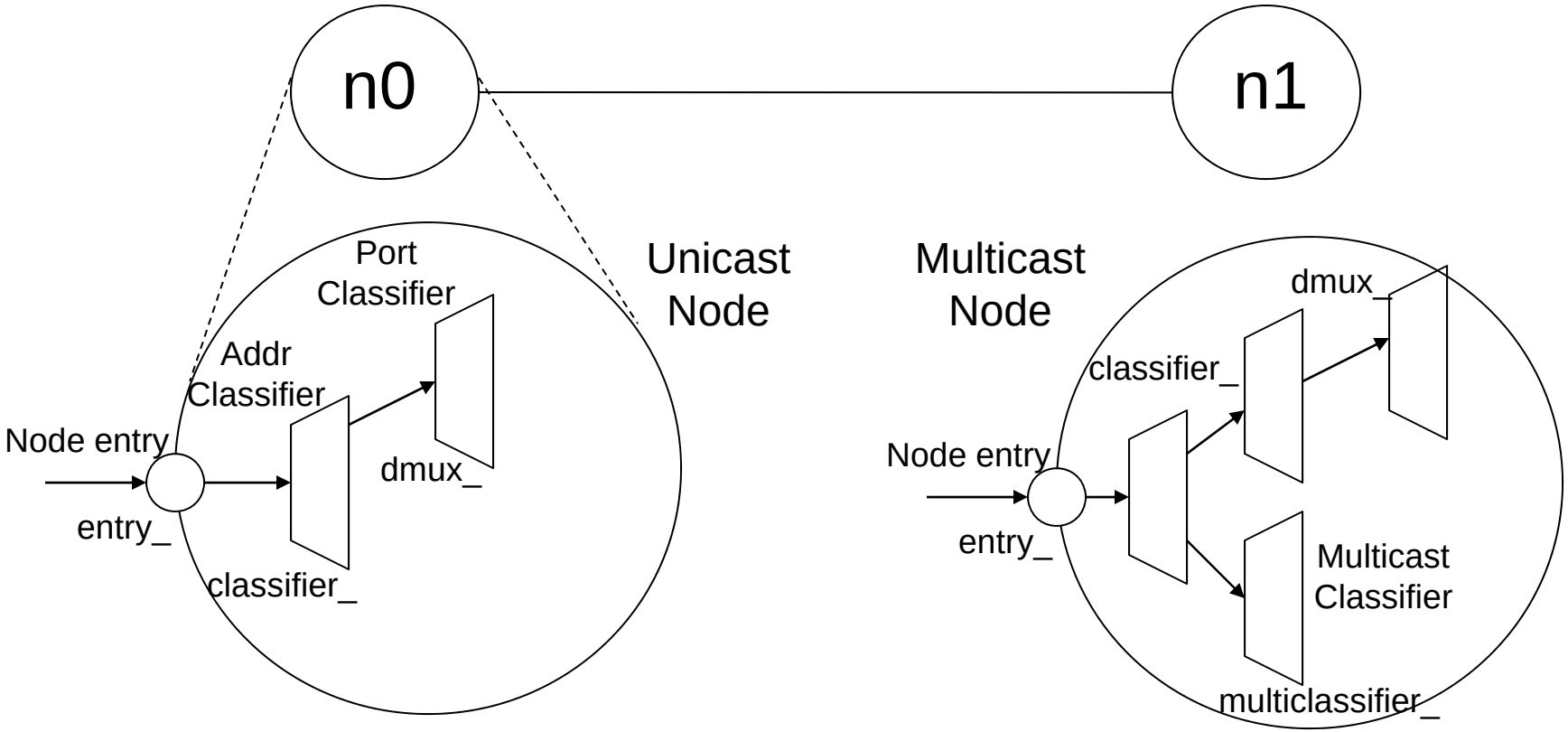
Source: [13]

Discrete event scheduler



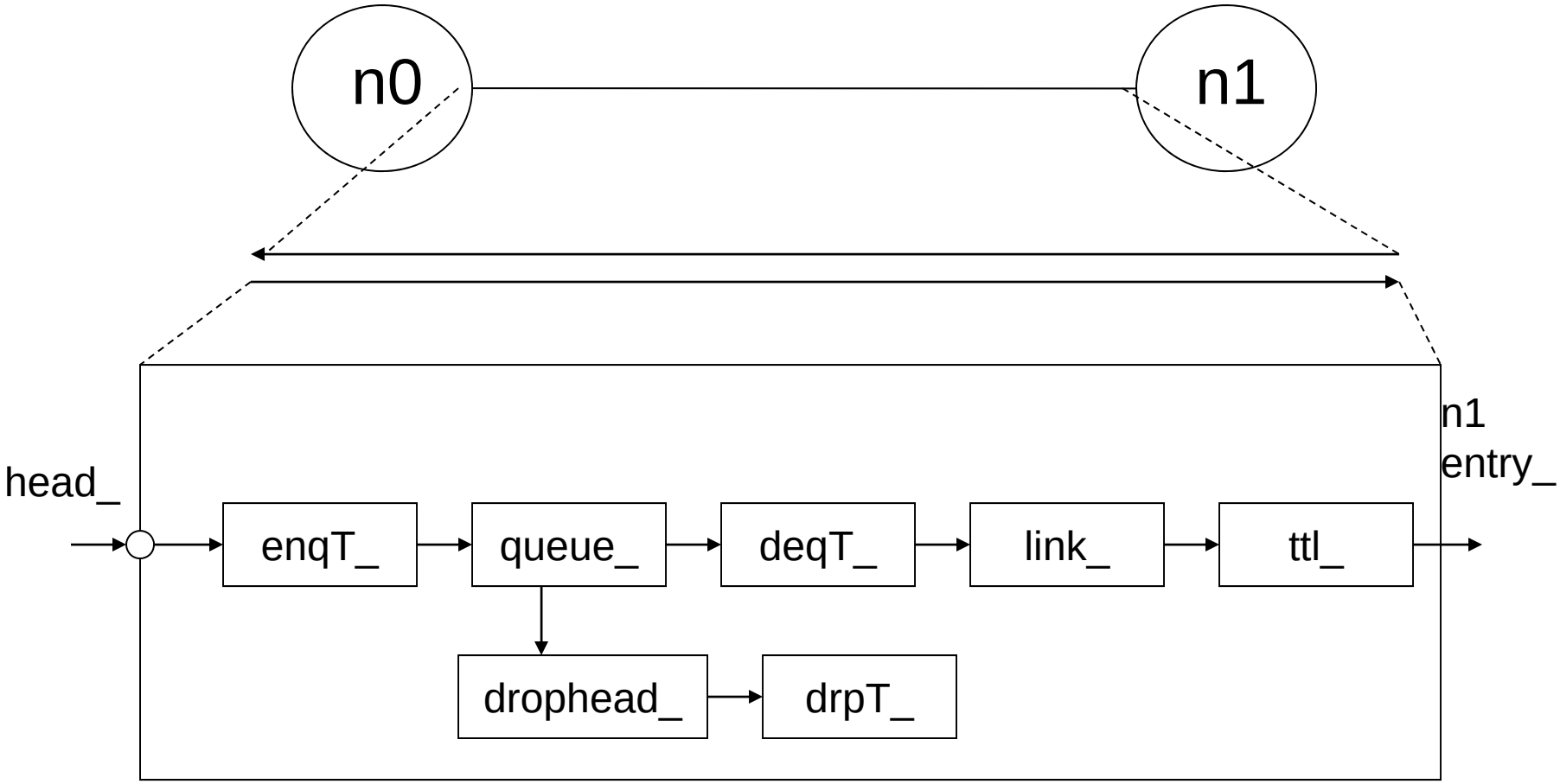
Source: [14]

Network topology - Node



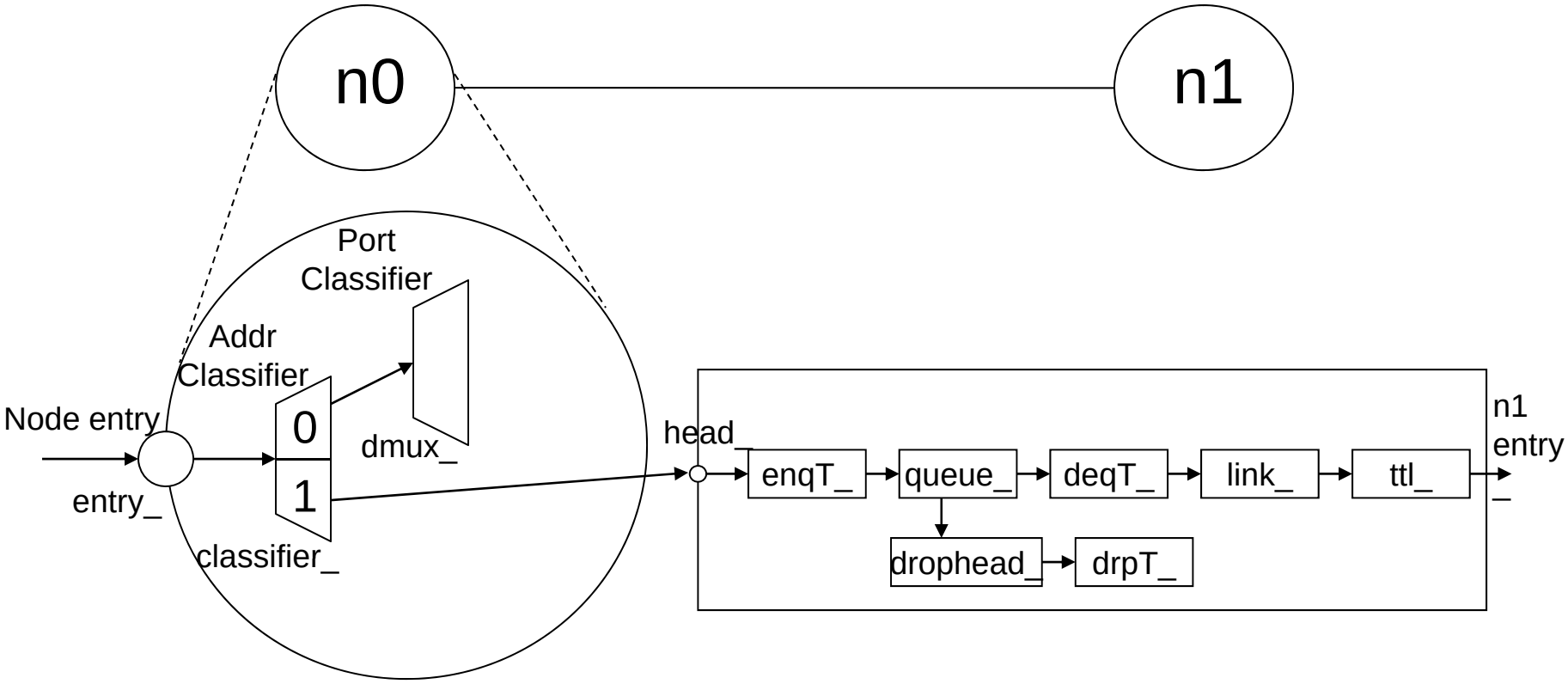
Source: [13]

Network topology - Link



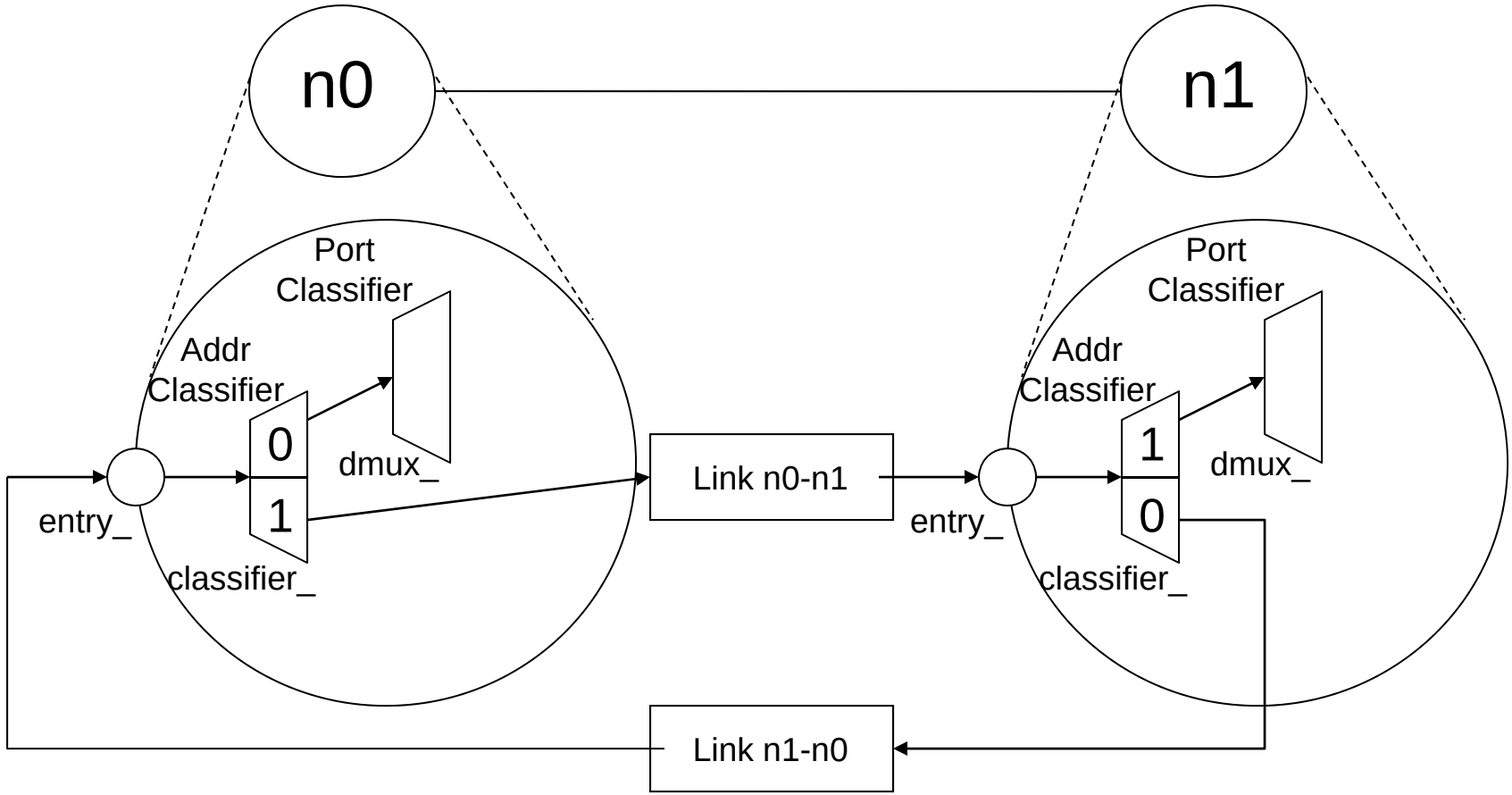
Source: [13]

Routing



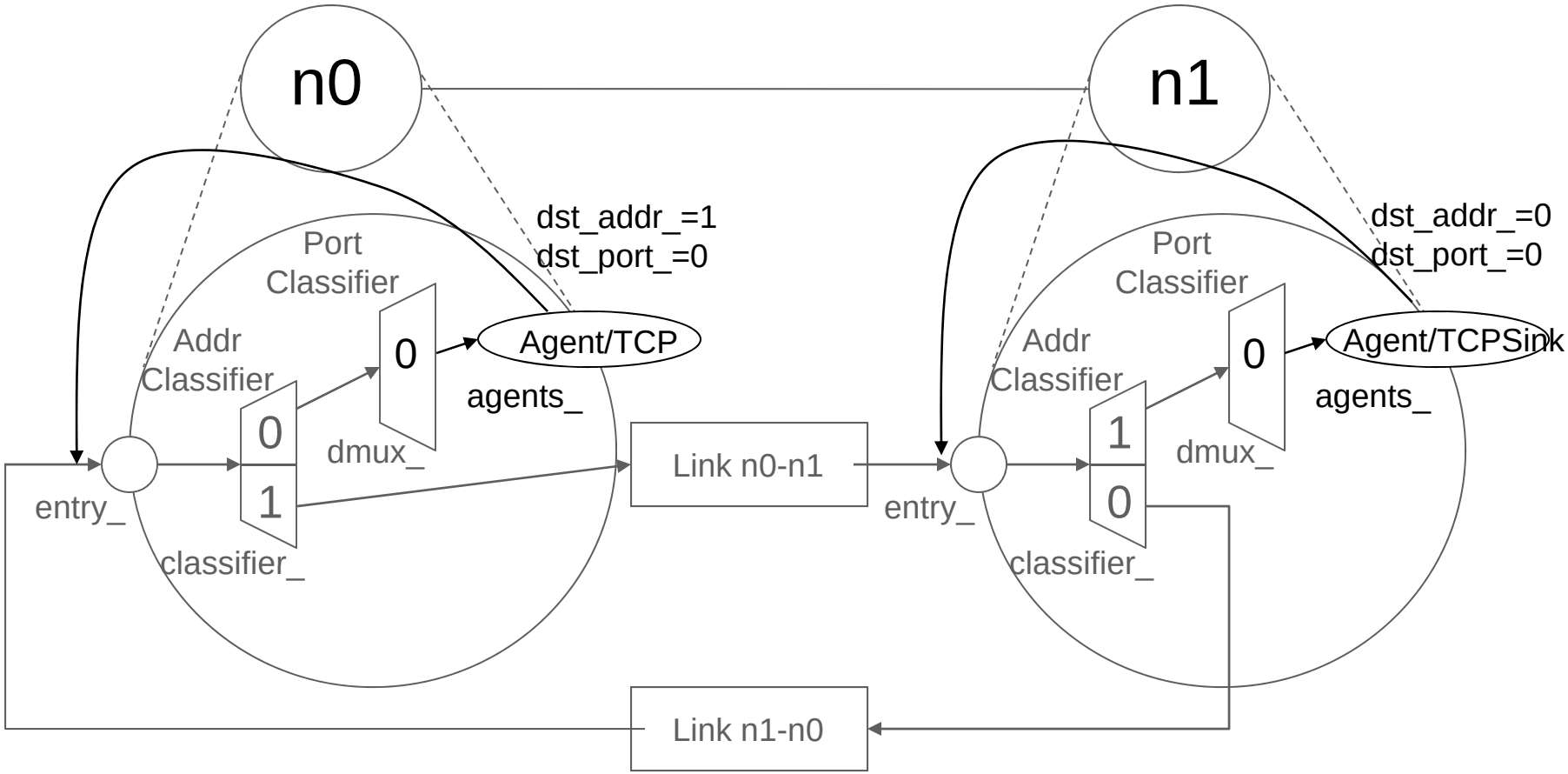
Source: [13]

Routing (cont'd)



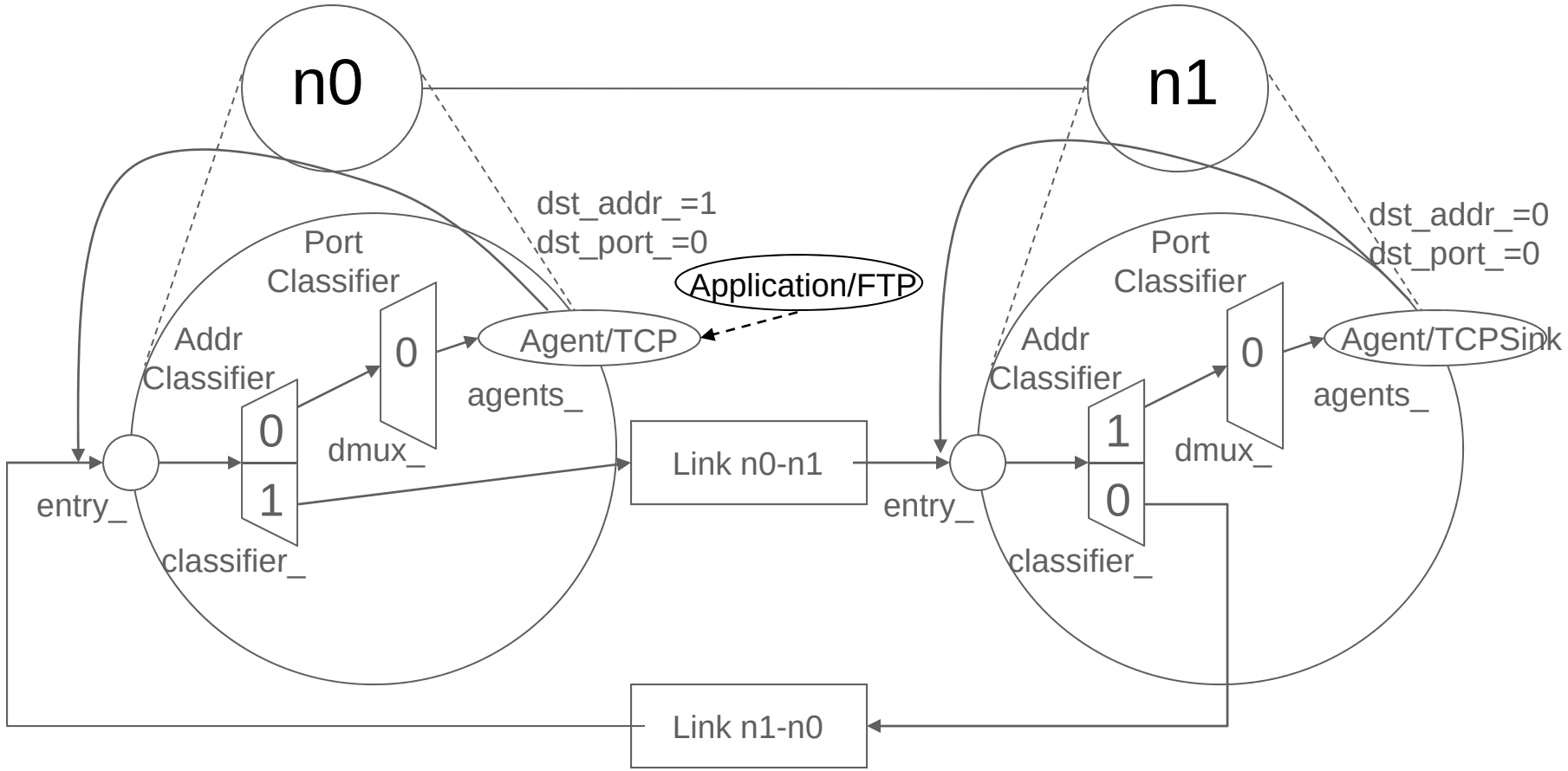
Source: [13]

Transport



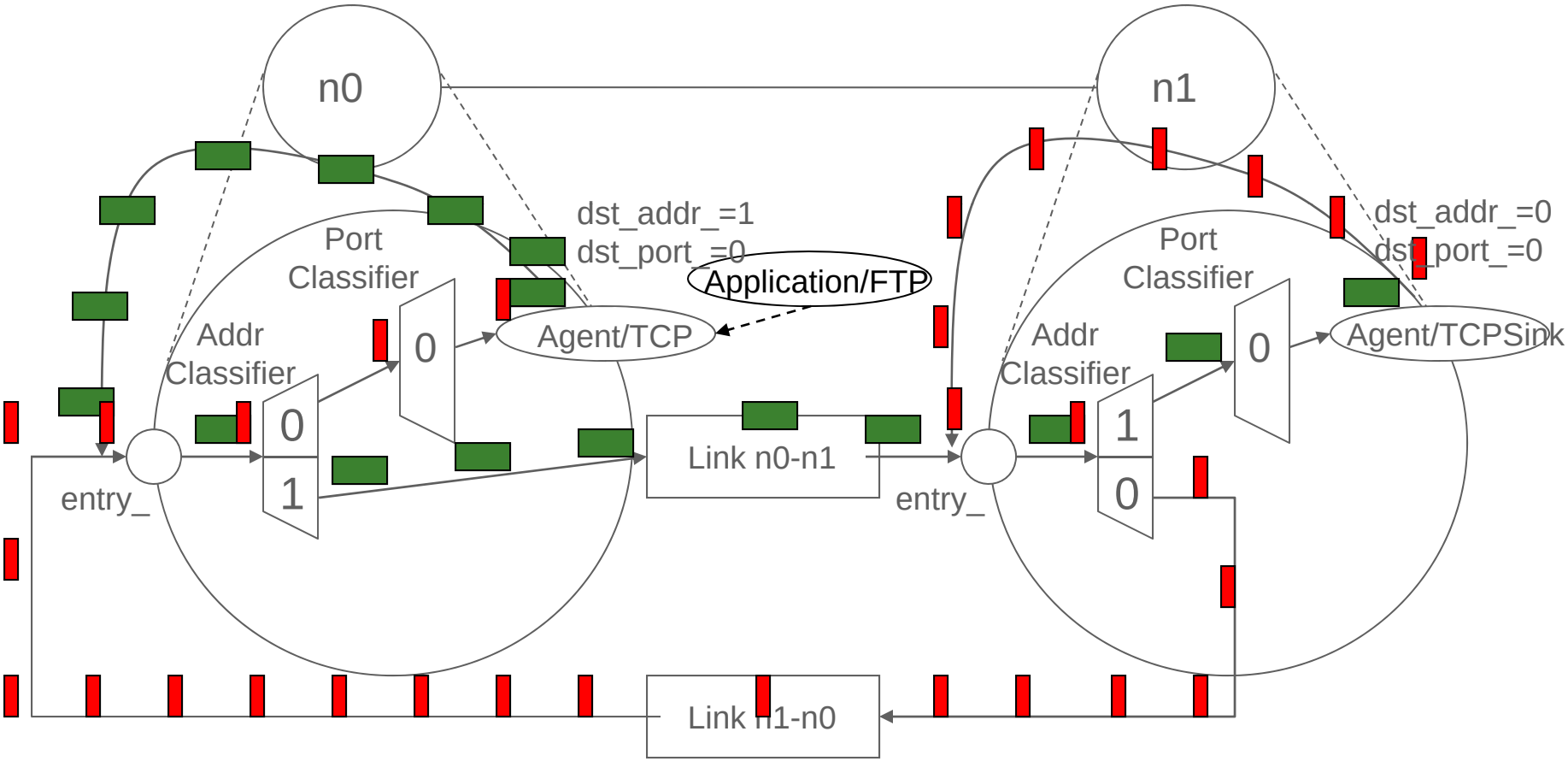
Source: [13]

Application



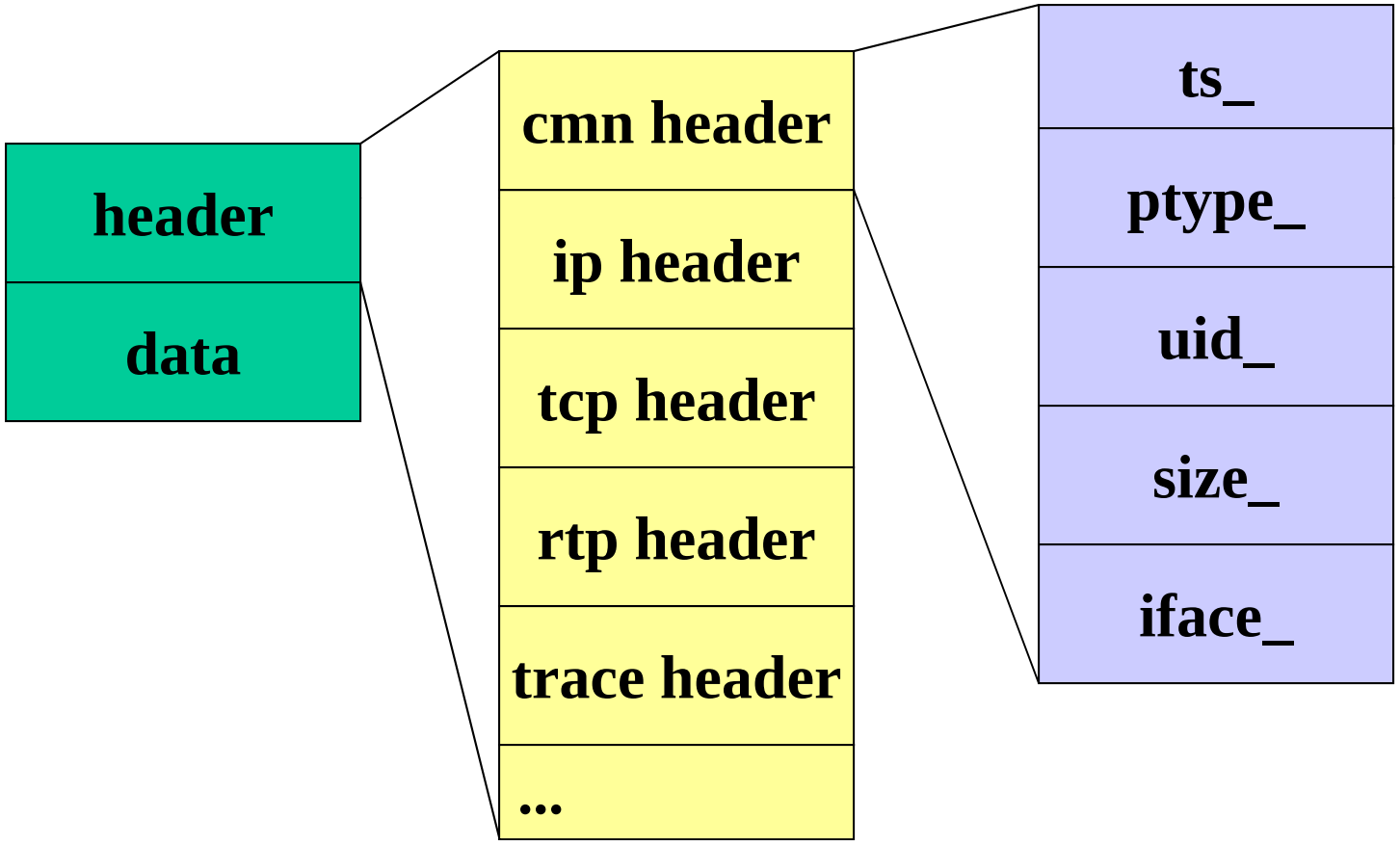
Source: [13]

Packet Flow



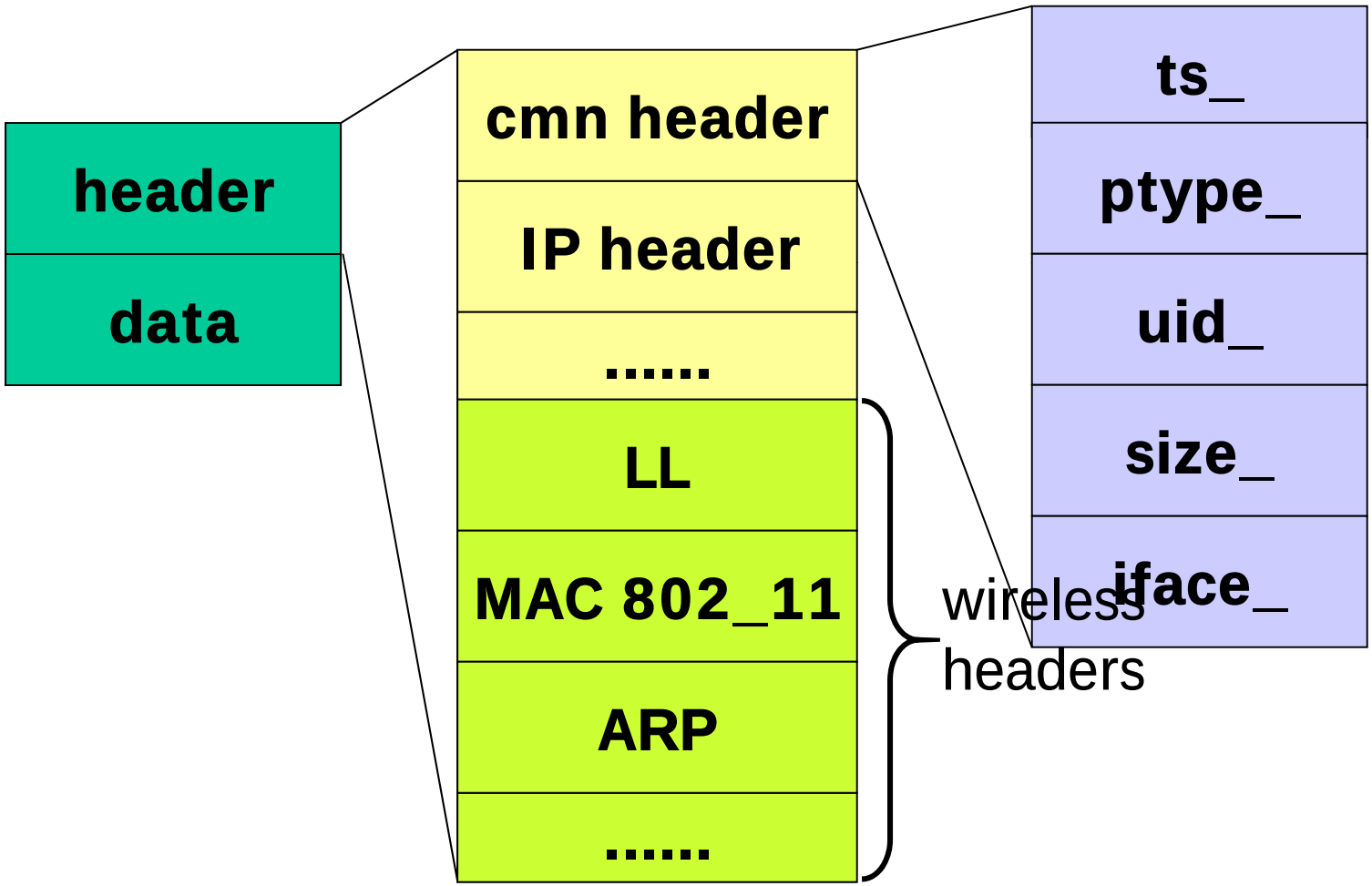
Source: [13]

Packet format



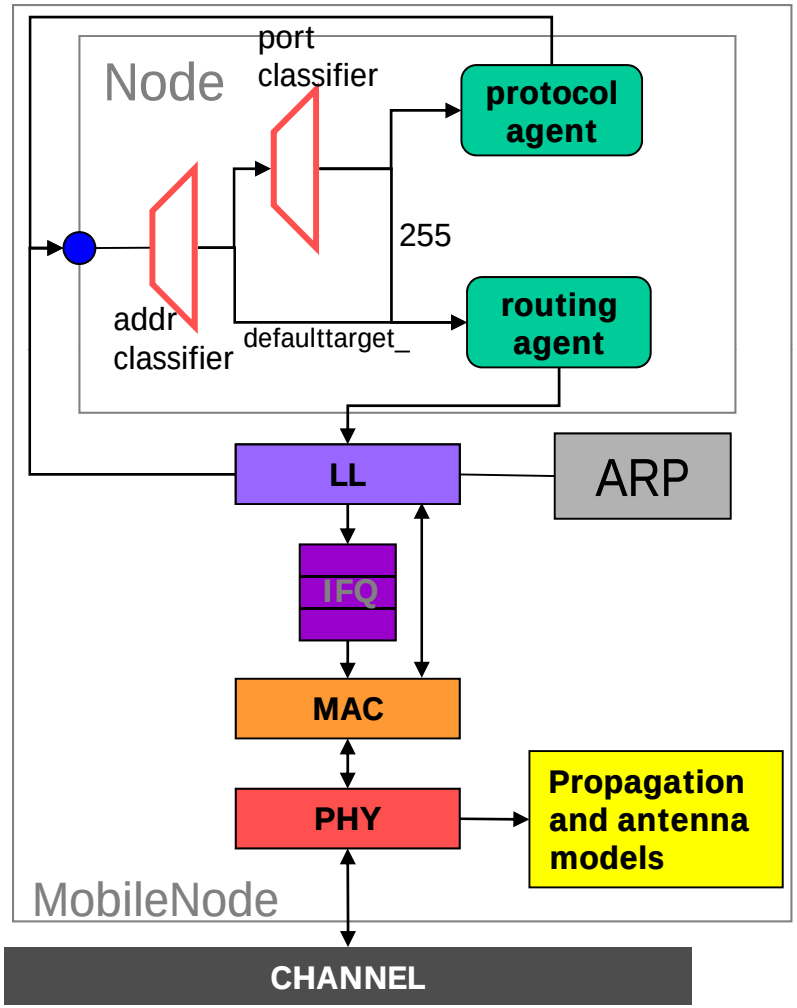
Source: [13]

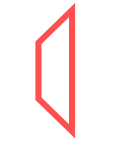
Wireless packet format



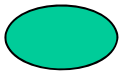
Source: [13]

Wireless node (Mobile node)





Classifier: Forwarding



Agent: Protocol Entity

Node Entry

LL: Link layer object

IFQ: Interface queue

MAC: Mac object

PHY: Net interface

Source: [13]

What We Will Learn

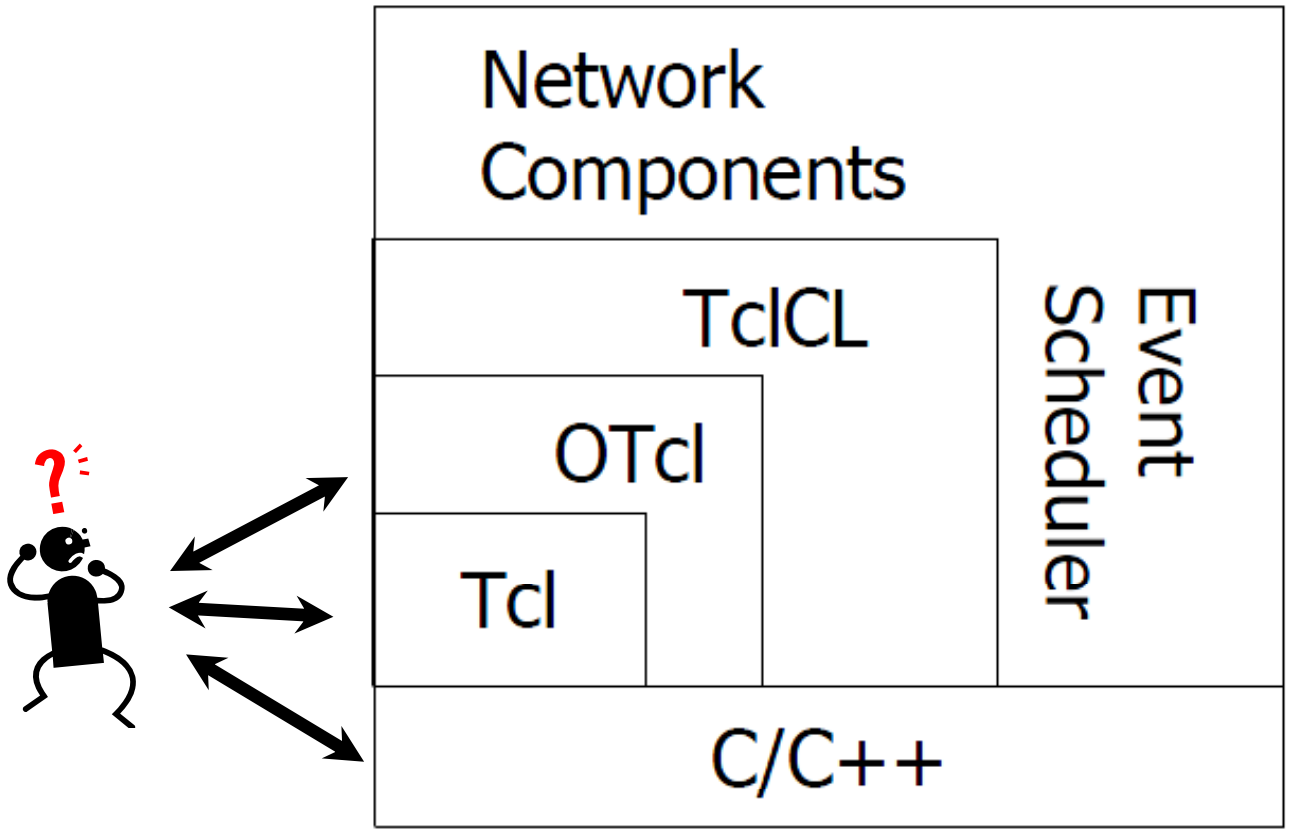
- Introduction
- Installation
- Architecture
- TCL & OTcl examples
- Simulation Steps
- The first wired example
- Simulation results
- More details
- Advanced wired examples
- The first wireless example
- More details
- Advanced wireless examples
- Look at implementations
- Extending the NS

Extending the NS

- Who are the developer?
- Other guys
  - The code is ready
  - We should patch NS
    - There is not a common approach for all patches
    - Refer to the manual of the patch
  - General tips
    - Try to use the NS version which the patch was developed for it
    - Try to use the same GCC version

Extending the NS

- We are the developer



New module development

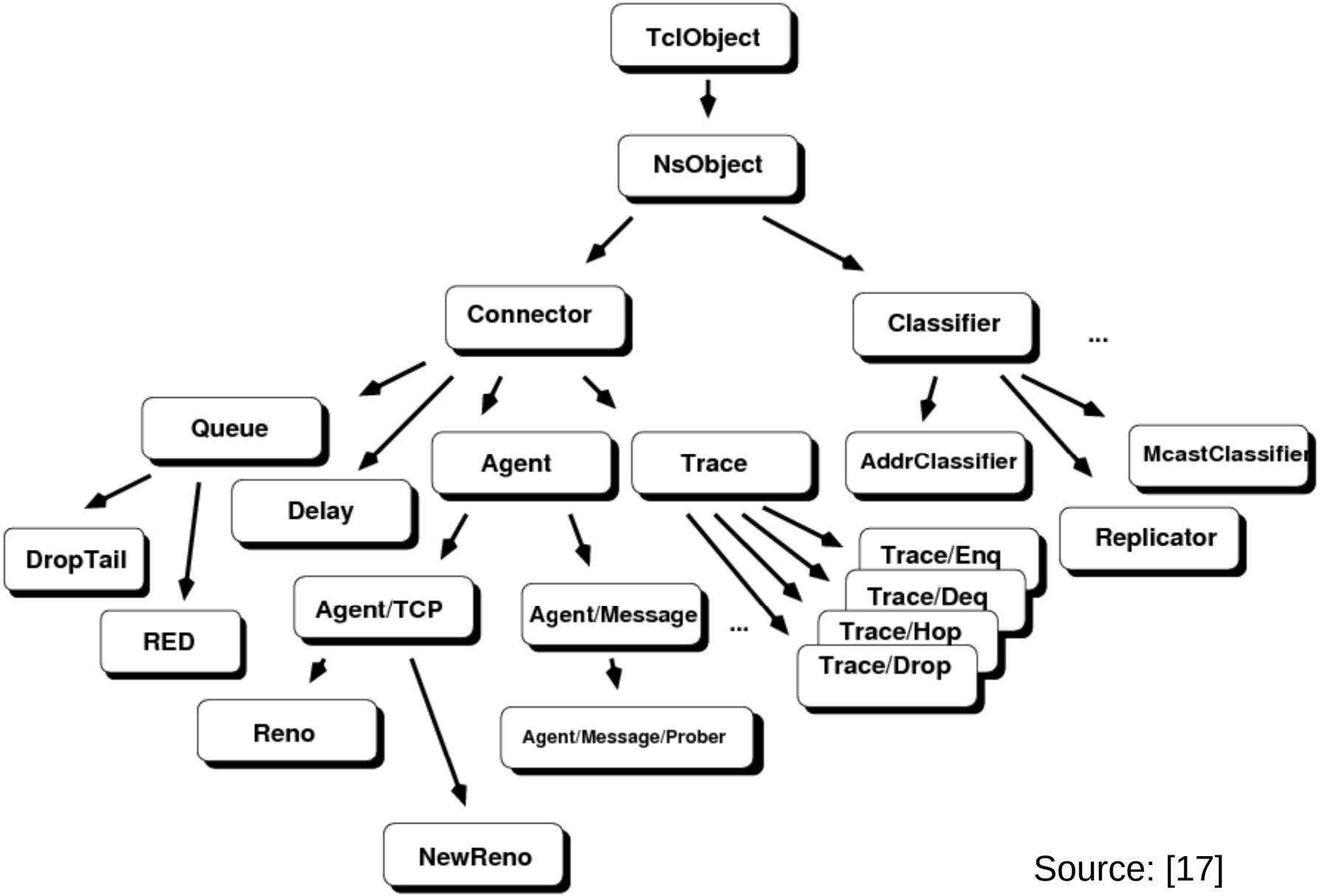
- Two approaches
- In pure OTcl
  - Is not suitable for complex and fast processing
  - We don't discuss here
    - An example in slides 29-43 in [13]
- In C++ and corresponding OTcl codes
  - The C++ and OTcl linkage

C++ & OTcl coding terminology

	The interpreted hierarchy	The compiled hierarchy
Base class	`Agent`	`Agent`
Derived class	`Agent/TCP`	`TcpAgent`
Derived class (2^{nd} level)	`Agent/Tcp/Reno`	`RenoTcpAgent`
Class functions	`installNext`	`install_next`
Class variables	`windowOption_`	`wnd_option_`

Source: [8]

Class Hierarchy



Source: [17]

C++ & OTcl linkage

- Data path is implemented in C++
- Control path & User interface is in OTcl
- We need a communication facility between C++ and OTcl
  - It is the tclcl
- From developers' point of
  - Two class hierarchies
  - For each C++ class, we need a OTcl class

C++ & OTcl linkage: Example

- Lets do it by an example
- Data path

```
class TestClass : public TclObject {
  public:
      TestClass() {
           printf("\nHey!!! I am here\n");
      }
};
```

C++ & OTcl linkage: Example (cont'd)

- Exporting the class to OTcl

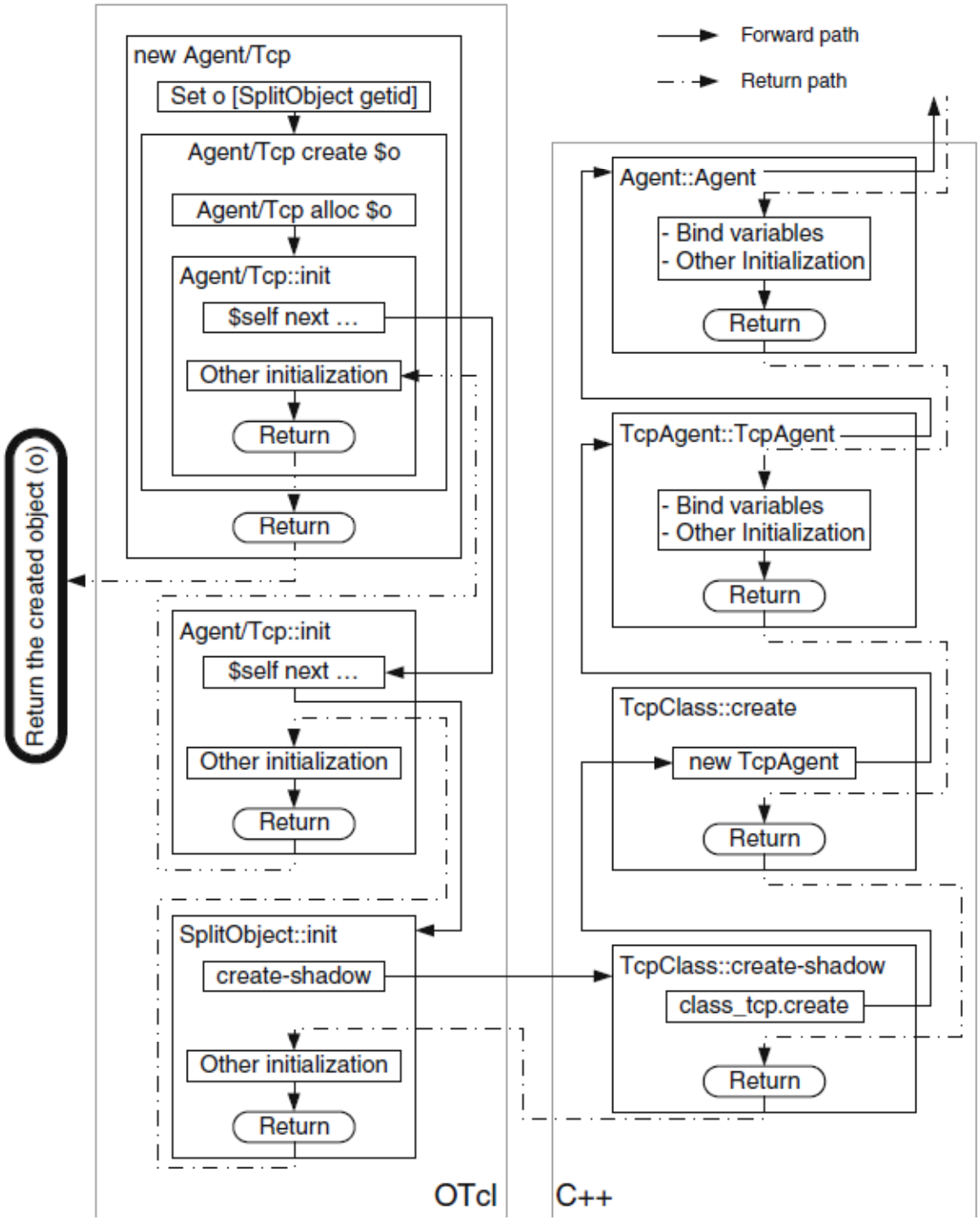
```
static class OtclTestClass : public TclClass{
  public:
     OtclTestClass(): TclClass("LabTestClass") {
     }
     TclObject* create(int, const char*const*){
          return (new TestClass());
     }
} otcl_test_class;
```

C++ & OTcl linkage: Example (cont'd)

- **`TestClass`** is the class name in data path, we don't see it
- **`OtclTestClass`** is the name of the shadow class corresponding to **`TestClass`**, we don't see the name
- **`LabTestClass`** is the name that is available in simulation scripts
  - We only need this one

C++ & OTcl linkage: Example (cont'd)

- We are ready to check it
  - Create file **`TestClass.cc`**
  - Include **`stdio`**, **`string`** and **`object`** in the file
  - Put the classes in the file
  - Put the file in **`$NS/ns-2.34`**
  - Add **`TestClass.o`** to the end of **`OBJ_CC`** list in **`Makefile`**
  - Compile ns: **`make`**
- In the ns
  - **`set mytest [new LabTestClass]`**


Forward path
Return path
new Agent/Tcp
Set o [SplitObject getid]
Agent/Tcp create $o
Agent/Tcp alloc $o
Agent/Tcp::init
$self next ...
Other initialization
Return
Return
Return the created object (o)
Agent/Tcp::init
$self next ...
Other initialization
Return
SplitObject::init
create-shadow
Other initialization
Return
Agent::Agent
- Bind variables
- Other Initialization
Return
TcpAgent::TcpAgent
- Bind variables
- Other Initialization
Return
TcpClass::create
new TcpAgent
Return
TcpClass::create-shadow
class_tcp.create
Return
OTcl
C++

Source: [8]

C++ & OTcl linkage: Variable Binding

- For each variable in C++ class
  - We need a corresponding variable in OTcl
    - They are bounded to each other
  - Because we don't access C++ classes/objects
- Bind these variables in the constructor of the C++ class

```
bind("OTcl_name", &CPP_var);
```

  - The **`CPP_var`** is accessable by **`OTcl_name`** in simulation scripts

C++ & OTcl linkage: Variable Binding

- To avoid a warning messages, initialize the OTcl variable in **`~ns/tcl/lib/ns-default.tcl`**

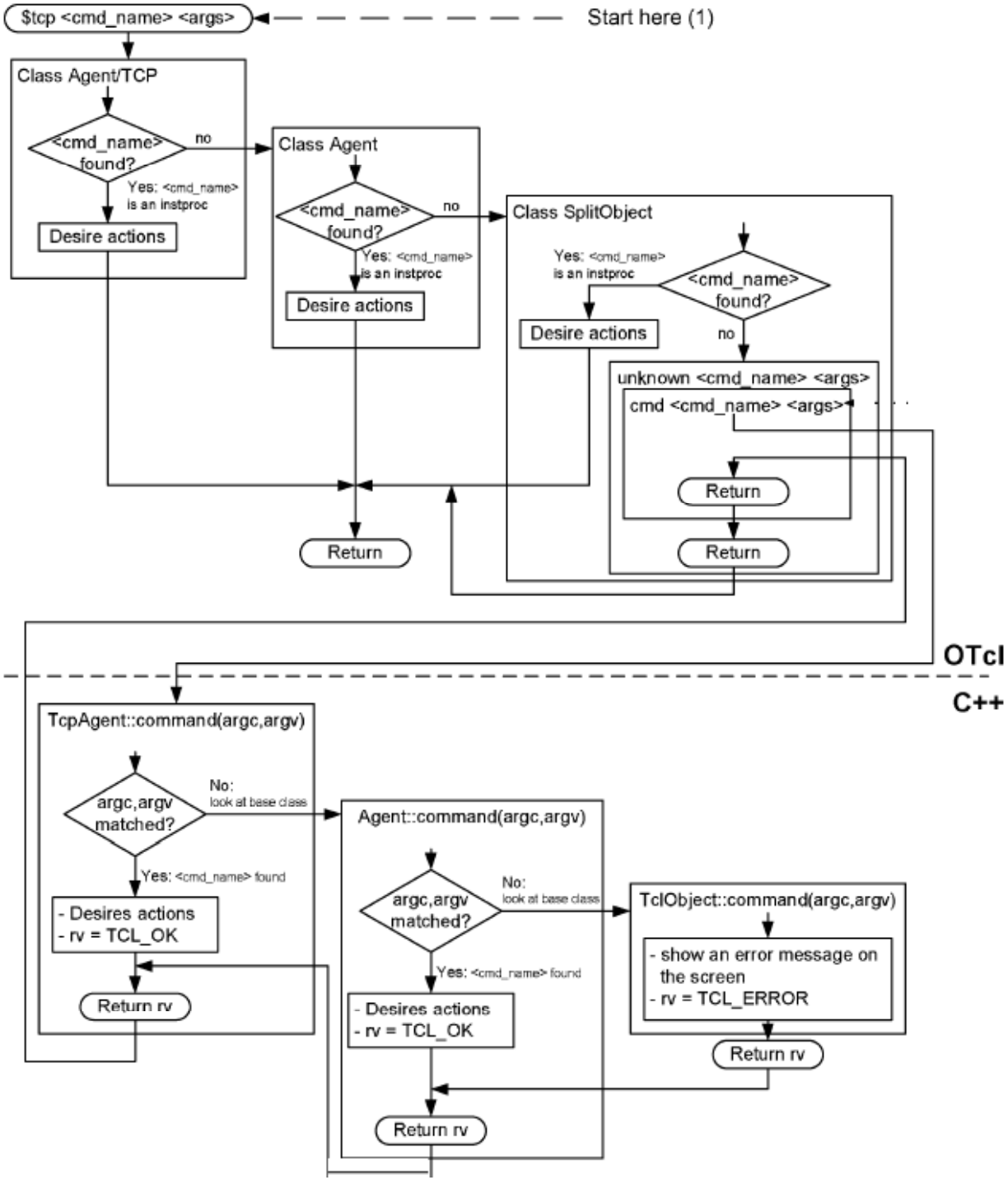
```
OTcl_Shadow_Class set OTcl_name 0
```

Variable binding: Example

- Add 2 variables to TestClass
  - `int cpp_var_1;` `double cpp_var_2`
- Bind them
  - `otcl_var_1; otcl_var_2`
- Initialize them in ns-default.tcl and make
- In ns
  - Set and get the values of variables
  - Be careful, OTcl is crazy
    - If you set a nonexistent variable → it will be created

Command processing

- When a procedure of an OTcl object is invoked
  - The object is checked
  - If not found → its parent is checked, and so on
- If the procedure is not found in OTcl space
  - It is passed to C++ space
  - C++ class should have **`command`** method
  - It is the interpreter of message comes from OTcl
    - There is **`argc`** and **`argv`**
    - Traditional parameter checking problem


$tcp <cmd_name> <args>
Start here (1)
Class Agent/TCP
<cmd_name> found?
no
Yes: <cmd_name> is an instproc
Desire actions
Class Agent
<cmd_name> found?
no
Yes: <cmd_name> is an instproc
Desire actions
Return
Class SplitObject
Yes: <cmd_name> is an instproc
<cmd_name> found?
Desire actions
no
unknown <cmd_name> <args>
cmd <cmd_name> <args>
Return
Return
OTcl
C++
TcpAgent::command(argc,argv)
argc,argv matched?
No: look at base class
Yes: <cmd_name> found
- Desires actions
- rv = TCL_OK
Return rv
Agent::command(argc,argv)
argc,argv matched?
No: look at base class
Yes: <cmd_name> found
- Desires actions
- rv = TCL_OK
Return rv
TclObject::command(argc,argv)
- show an error message on the screen
- rv = TCL_ERROR
Return rv

Source: [8]

Command processing: Example

- Add a method to **`TestClass`** to reinitialize the variables

```
class TestClass : public TclObject {
  public:
     int command(int argc, const
  char*const* argv);
     void setValues(int, double);
     ...
  };
```

```
void TestClass::setValues(int i, double d){
  cpp_var_1 = i;
  cpp_var_2 = d;
}
int TestClass::command(int argc, const char*const*
  argv) {
  if(argc == 4){
     if(strcmp(argv[1], "set-all-values") == 0){
          setValues(atoi(argv[2]), atof(argv[3]));
          return(TCL_OK);
     }
  }
  return(TCL_ERROR);
}
```

Command processing: Example

- Add the methods to the class
- Recompile
- In ns

```
set mytest [new LabTestClass]
$mytest set-all-values 10 20
puts "[$mytest set otcl_var_1]"
$mytest set otcl_var_2
```

Calling OTcl command from C++

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

char s[128];

strcpy(s,"puts \"A message from eval\"");

tcl.eval(s);

tcl.evalc("puts \"A message from evalc\"");

sprintf(tcl.buffer(),"puts \"A message from
eval using interal buffer\"");

tcl.eval();

tcl.evalf("puts \"%s\"", "A message from
evalf");
```

General guides for adding a new facility

- Completely depends on what you want to do
  - Application? Agent? Routing? MAC? …
- Look for similar projects' or modules' code and modify:
  - Starting from scratch is difficult and prone to errors
  - Faults are hard to debug due to C++/OTcl duality

Example: Adding a new agent

- We want to add a new ping agent
  - There is already implemented ping agent in NS
  - Our code is based on it, simpler
- In ping, both sides are sender and also receiver
  - We use the same agent for both of them
- Distinguish between request and reply by a field in packet header

TestPing.h

```
#ifndef ns_test_ping_h
#define ns_test_ping_h

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"

struct hdr_test_ping {
  char ret;
  double send_time;
  double rcv_time;
  int seq;
  static int offset_;
```

```
  inline static int& offset() { return offset_; }
  inline static hdr_test_ping* access(const Packet* p){
      return (hdr_test_ping*) p->access(offset_);
  }
};

class TestPingAgent : public Agent {
public:
  TestPingAgent();
  int seq;
  virtual int command(int argc, const char*const*
  argv);
  virtual void recv(Packet*, Handler*);
};
#endif // ns_ping_h
```

TestPing.cc

```
#include "TestPing.h"

int hdr_test_ping::offset_;

static class TestPingHeaderClass : public
  PacketHeaderClass {

public:

  TestPingHeaderClass() :
  PacketHeaderClass("PacketHeader/TestPing",
                                sizeof(hdr_test_ping)) {

      bind_offset(&hdr_test_ping::offset_);

  }

} class_test_pinghdr;
```

```
static class TestPingClass : public TclClass {
public:
  TestPingClass() : TclClass("Agent/TestPing") {}
  TclObject* create(int, const char*const*) {
      return (new TestPingAgent());
  }
} class_test_ping;



TestPingAgent::TestPingAgent() : Agent(PT_TEST_PING),
  seq(0)
{
  bind("packetSize_", &size_);
}
```

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

```
void TestPingAgent::recv(Packet* pkt, Handler*){
  hdr_ip* hdrip = hdr_ip::access(pkt);
  hdr_test_ping* hdr = hdr_test_ping::access(pkt);
  if (hdr->ret == 0) {
      double stime = hdr->send_time;
      int rcv_seq = hdr->seq;
      Packet::free(pkt);
      Packet* pktret = allocpkt();
      hdr test ping* hdrret =
      hdr_test_ping::access(pktret);
      hdrret->ret = 1;
      hdrret->send_time = stime;
      hdrret->rcv_time = Scheduler::instance().clock();
      hdrret->seq = rcv_seq;
      send(pktret, 0);
  }
```

```
else {
        printf("Node %d received reply from %d, seq = %d,
   RTT = %3.1f\n",
        hdrip->dst_.addr_ >>
        Address::instance().NodeShift_[1],
        hdrip->src_.addr_ >>
        Address::instance().NodeShift_[1],
        hdr->seq,
        (Scheduler::instance().clock()-hdr->send_time) *
        1000);
        Packet::free(pkt);
   }
}
```

Adding the ping example to NS

- Put the .cc and .h files in **`$NS/ns-2.34`**
- Add **`TestPing.o`** to the end of **`OBJ_CC`** list in **`Makefile`**
- Initialize packet size in **`ns-default.tcl`**
  - **`Agent/TestPing set packetSize_ 64`**
- Define the new packet type in **`packet.h`**
  - **`static const packet_t PT_TEST_PING = 62;`**
  - **`name_[PT_TEST_PING] = "Test_Ping";`**

Compile

- In **`$NS/ns-2.34`**
  - **`make clean`**
  - **`make`**

- For new packet type
  - Add it to **`packet.h`**
  - **`make clean`** is needed

Test the agent

```
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
set tp0 [new Agent/TestPing]
$ns attach-agent $n0 $tp0
set tp1 [new Agent/TestPing]
$ns attach-agent $n1 $tp1
$ns connect $tp0 $tp1
$ns at 0.5 "tp0 send"
$ns at 5 "finish"
$ns run
```

More powerful ping

```
set sentPkt 0
proc ping {agent cnt size interval} {
   global ns sentPkt
   if {$sentPkt < $cnt} {
       $agent set packetSize_ $size
       $agent send
       incr sentPkt
       $ns at [expr [$ns now] + $interval] "ping $agent
   $cnt $size $interval"
   }
}
$ns at 0.5 "ping $tp0 5 1000 10"
```

Playing with the ping agent

- Check the output
- Check the trace file
- Change the parameters of links
- Use different values of size and interval

Additional references for adding a new facility

- A complete guide to implementing MANET routings can be found in [15]
- The NS manual
  - Section 10.6 provides more details
- The most complete development guide is [8]
  - Chapter 14 develops ARQ module
- A message agent is developed in [13]

Memory conservation tips

- Remove unused packet headers
- Avoid **`trace-all`**
- Use arrays for a sequence of variables:
  - instead of **`n$i`**, say **`n($i)`**
- Avoid OTcl temporary variables
- See tips for running large simulations in ns at
  - www.isi.edu/ns/nsnam/ns-largesim.html

References (1)

[1] Jianping Wang, "ns-2 Tutorial (1)," Multimedia Networking Group, The Department of Computer Science, UVA

[2] Noun Choi, "Introduction to ns-2”

[3] Zhibin WU, “Introduction to NS-2,” WINLAB, ECE Dept. Rutgers U.

[4] Ke Liu, “Network Simulator 2: Introduction,” Dept. Of Computer Science SUNY Binghamton

[5] Eitan Altman, Tania Jimenez, “NS Simulator for beginners,” Univ. del Los Andes, 2003

[6] http://yctrtrc.ncku.edu.tw/site2/ocwCoursePPT/9610-2.pdf

[7] Paul Meeneghan and Declan Delaney, "An Introduction to NS, Nam and OTcl scripting," Department of Computer Science, National University of Ireland

[8] Teerawat Issariyakul and Ekram Hossain, “Introduction to Network Simulator NS2,” Springer 2009.

[9] Richard Mortier, “ns-2 Introduction”

References (2)

[10] Gayatri Swamynathan, “An Introduction to NS-2”

[11] Marc Greis, “Tutorial for the UCB/LBNL/VINT Network Simulator ns”

[12] John Heidemann, “NS Tutorial”

[13] Polly Huang, “2nd European ns-2 Workshop (Day 2)”

[14] Jae Chung, Mark Claypool, "NS by Example“

[15] Francisco J. Ros, Pedro M. Ruiz, “Implementing a New Manet Unicast Routing Protocol in NS2,” Dept. of Information and Communications Engineering University of Murcia, 2004

[16] Brent Welch, "Practical Programming in Tcl and Tk"

[17] Kevin Fall, Kannan Varadhan, “The NS (v2) Simulator Workshop,” 1997