CSC 5480 - NS2 Tutorial (Part 1)

Presenter: alix, lhchow@cse.cuhk.edu.hk

8 Oct, 2001

Outline

- Introduction to NS2
- In-stallion
- NS2 Overview
- Simple Simulation Examples
- NS2 Basics
- Post Simulation

What is NS2

- A Network Simulator, most recent release ns-2.1b8 released June 06, '01
- An object-oriented, discrete event driven network simulator developed at UC Berkely
- Written in C++ and OTcl
- For simulating local and wide area networks
- Provides substantial support for simulation

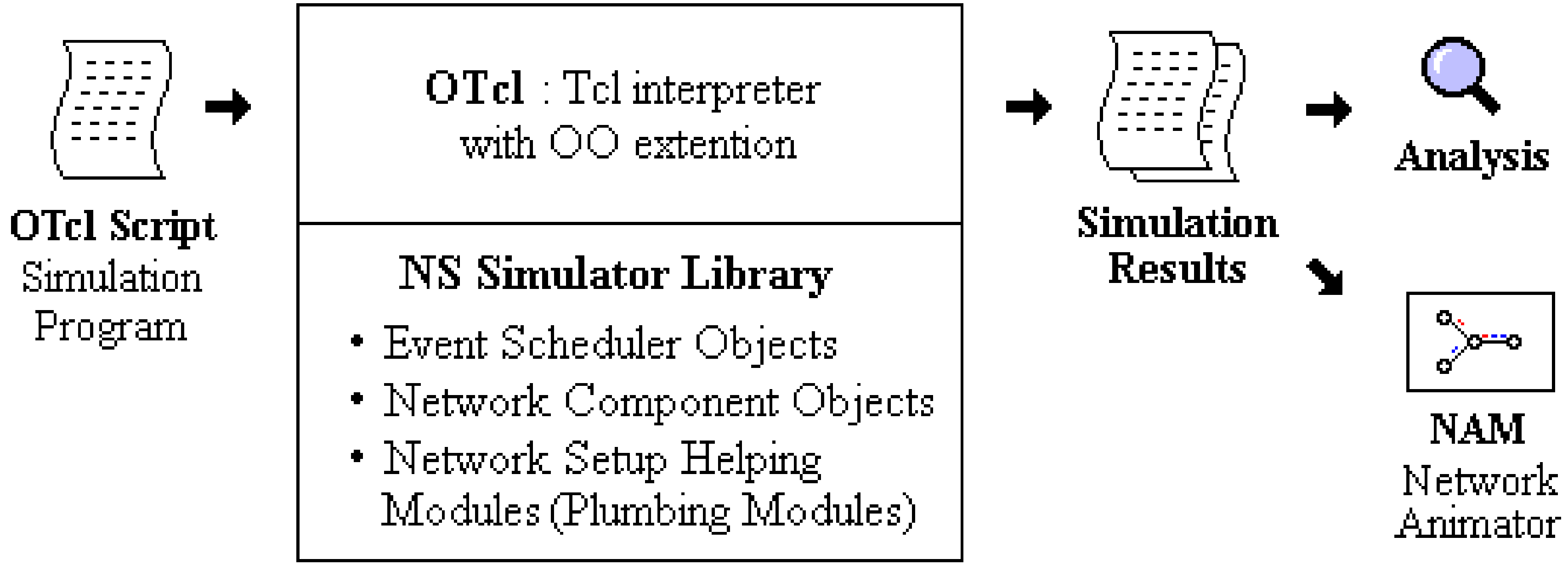
In-stallion

- 1 Download the ns-allinone package
  http://www.isi.edu/nsnam/dist/ns-allinone-2.1b8a.tar.gz
- 2 gtar zxvf ns-allinone-2.1b8a.tar.gz
- 3 Run the *install* script
- 4 Modify .bashrc

```
export LD_LIBRARY_PATH=~/ns-allinone-2.1b8a/otcl-1.0a7:~/ns-allinone-2.1b8a/lib:$LD_LIBRARY_PATH
export TCL_LIBRARY=~/ns-allinone-2.1b8a/tcl8.3.2/library:$TCL_LIBRARAY
PATH=~/ns-allinone-2.1b8a/bin:~/ns-allinone-2.1b8a/tcl8.3.2/unix:
     ~/ns-allinone-2.1b8a/tk8.3.2/unix:${PATH}
```

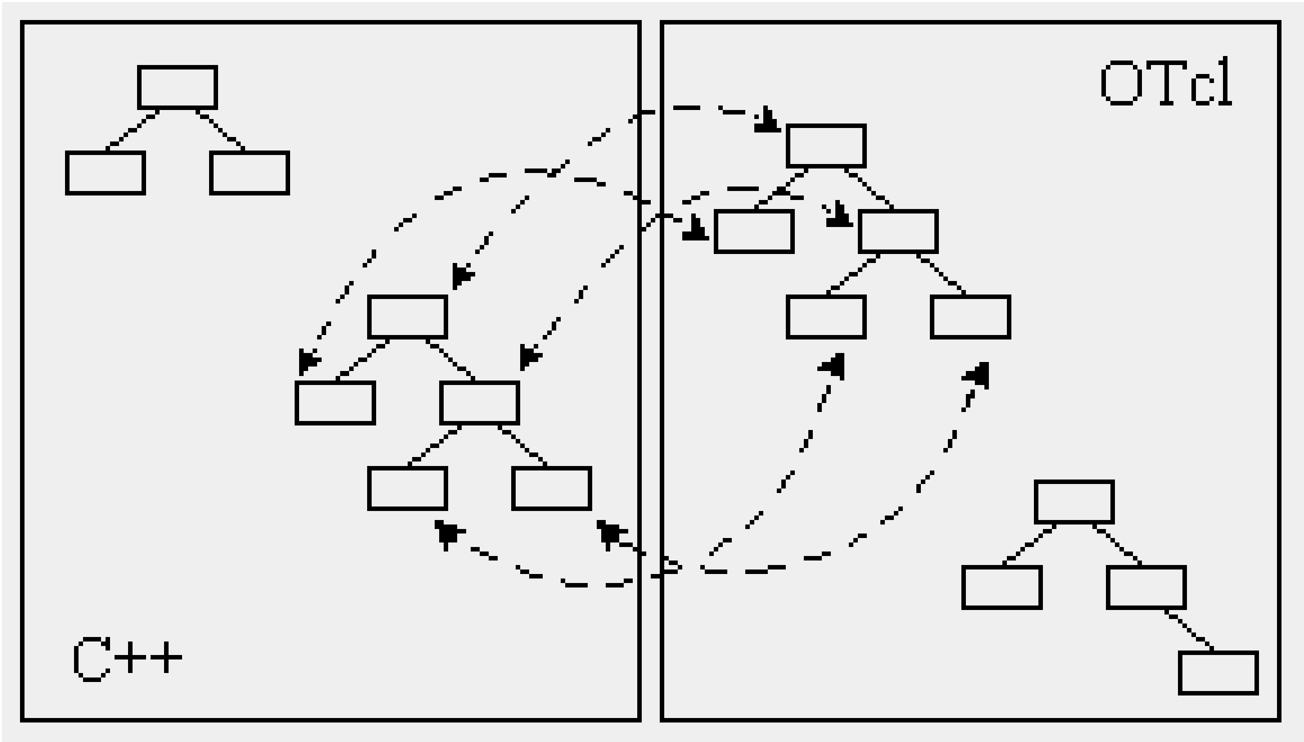
NS2 Overview

- An Object-oriented Tcl (OTcl) script interpreter containing:
  - Simulation event scheduler
  - Network component object libraries
  - Network setup (plumbing) module libraries

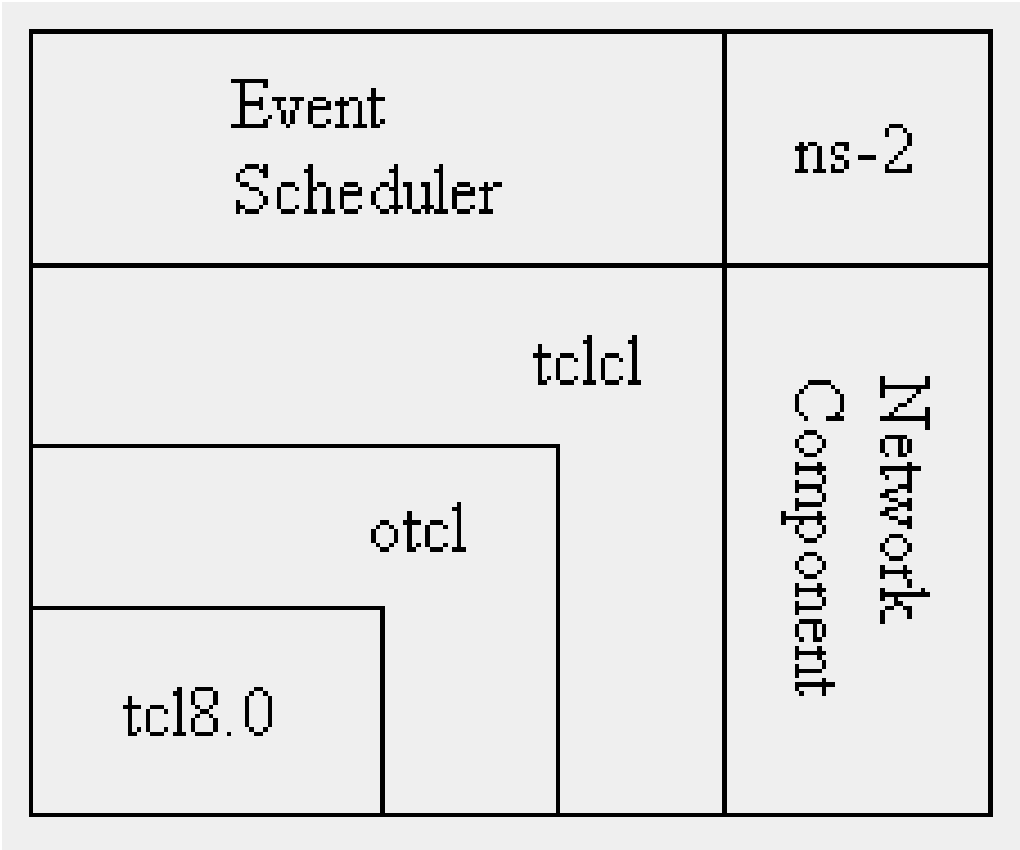

OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

- Simple Steps to Setup and run a simulation network
  - Write an OTcl script that initiates an event scheduler
  - Sets up the network topology using the network objects and the plumbing functions in the library
  - Tells traffic sources when to start and stop transmitting packets through the event scheduler
- Overview
  - Network components communicate with one another passing packets
  - Event Schduler - Schdule Event, Simulate Network Delay, Processing Delay, TCP Timer ...

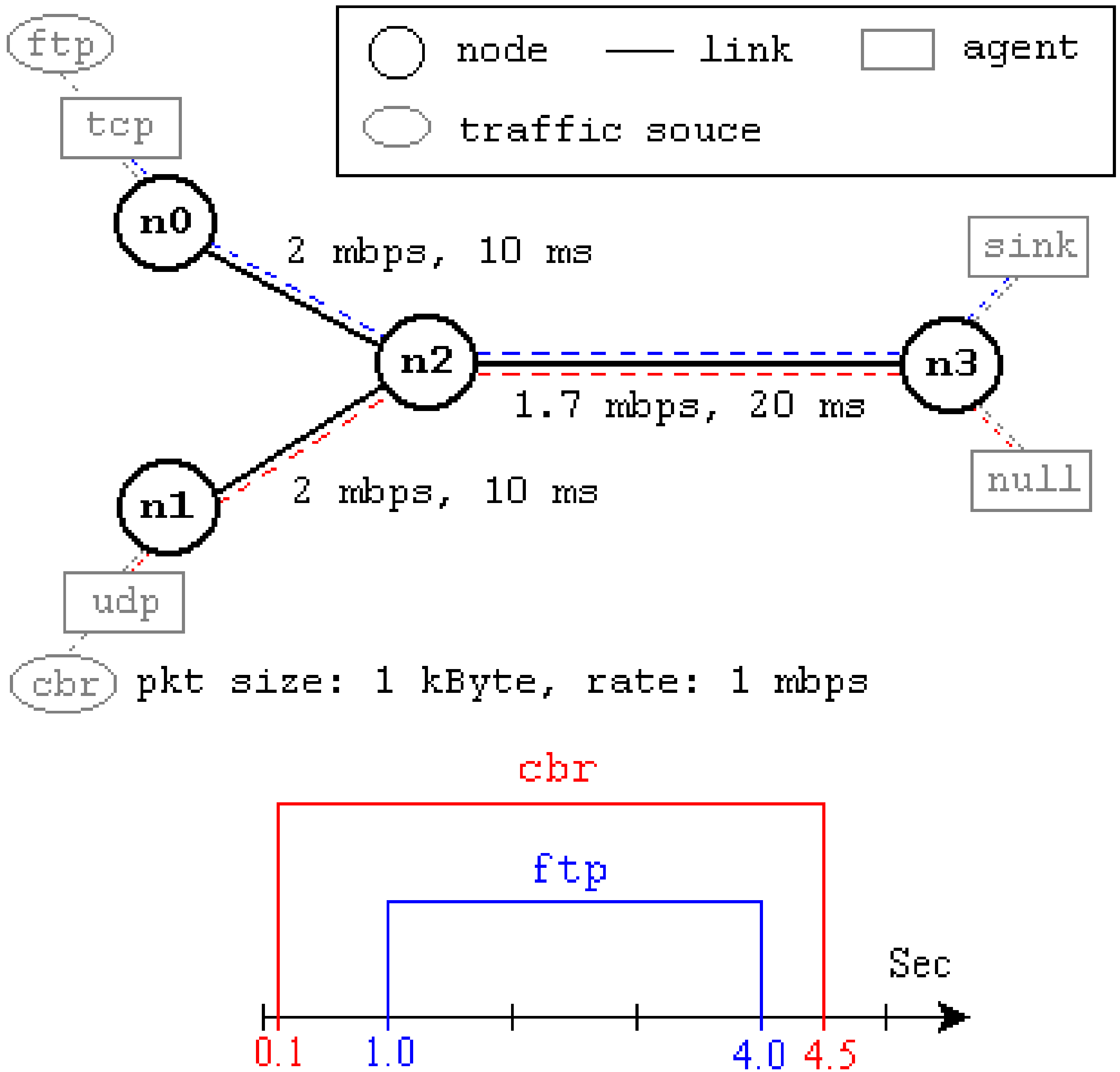
- OTcl <=> C++
  - For efficiency reason, NS separates the data path implementation from control path implementations
  - Event scheduler and the basic network component objects in the data path are written and compiled using C++
  - Controls between the C++ objects and OTcl
    - Through an OTcl linkage
    - e.g. OTcl controls C++
    - Creates a matching OTcl object for each of the C++ objects
    - Makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object


OTcl
C++

OO extended Tcl interpreter with network simulator libraries



Simple Simulation Examples


node
link
agent
traffic souce
ftp
tcp
n0
2 mbps, 10 ms
sink
n2
n3
1.7 mbps, 20 ms
null
n1
2 mbps, 10 ms
udp
cbr
pkt size: 1 kByte, rate: 1 mbps
cbr
ftp
Sec
0.1
1.0
4.0
4.5

- The Simulator object just acts as a user interface, and it actually calls the member functions of network objects or a scheduler object that does the real job

- "null" agent - frees the packets received

- Tcp "sink" agent - generates and sends ACK packets to the sender (tcp agent) and frees the received packets

- Output queue of a node is implemented as a part of a link

- Instead of create Node or Link, users can create the sub-objects and connect them

- See *"ns-2/tcl/libs/ns-default.tcl"* for avaliable network objects and their configurable parameters

- Usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit
- The agent, just knowing how much of the data to transfer, creates packets and sends them

Event Scheduler

- Used for network components that simulate packet-handling delay or that need timers
- Real-time and non-real-time schedulers

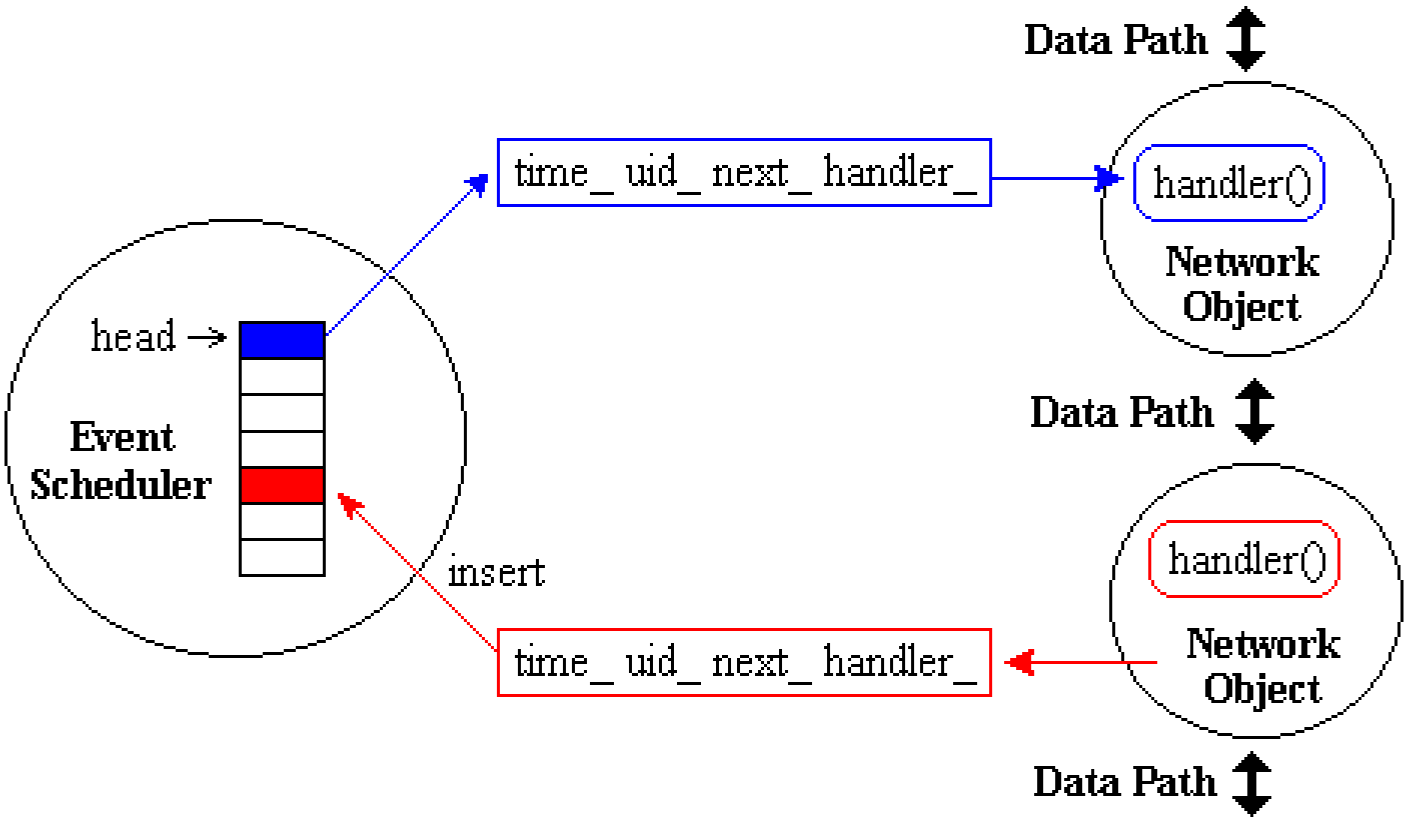
\- Calendar non-real-time scheduler is set as the default

\- *$ns use-scheduler Heap*

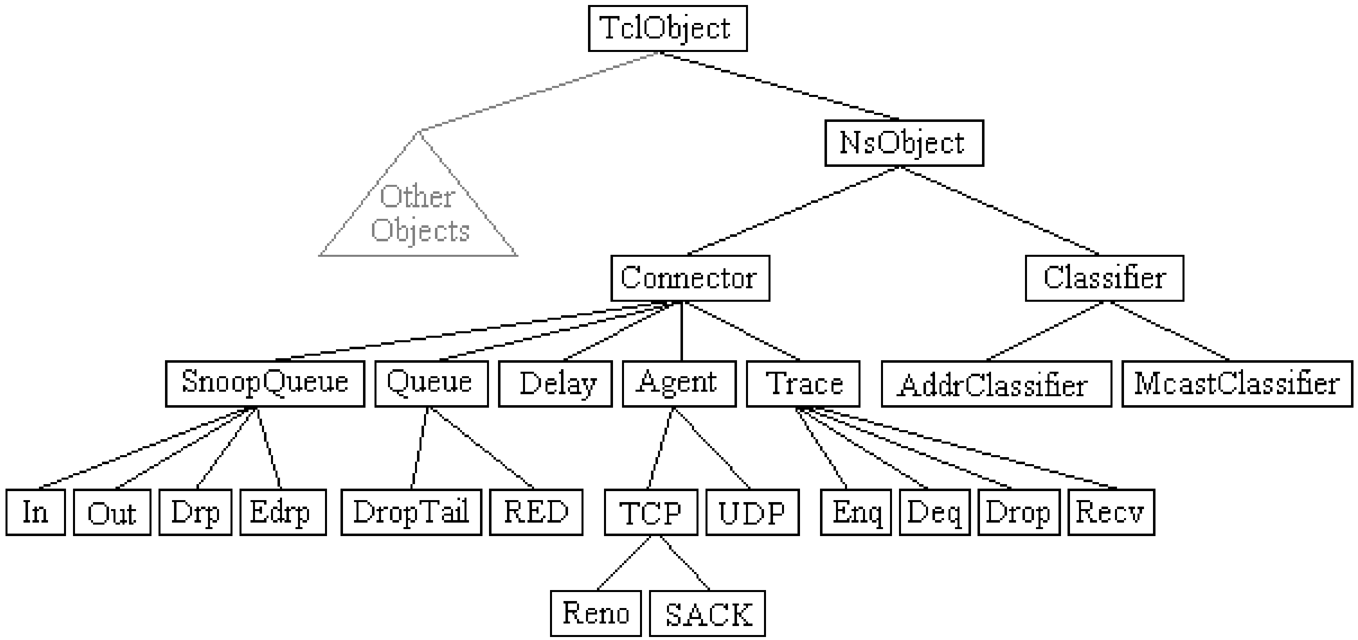
\- Reat-time scheduler(under development) allow the simulator to interact with a real network

- Another use of an event scheduler is to schedule simulation events

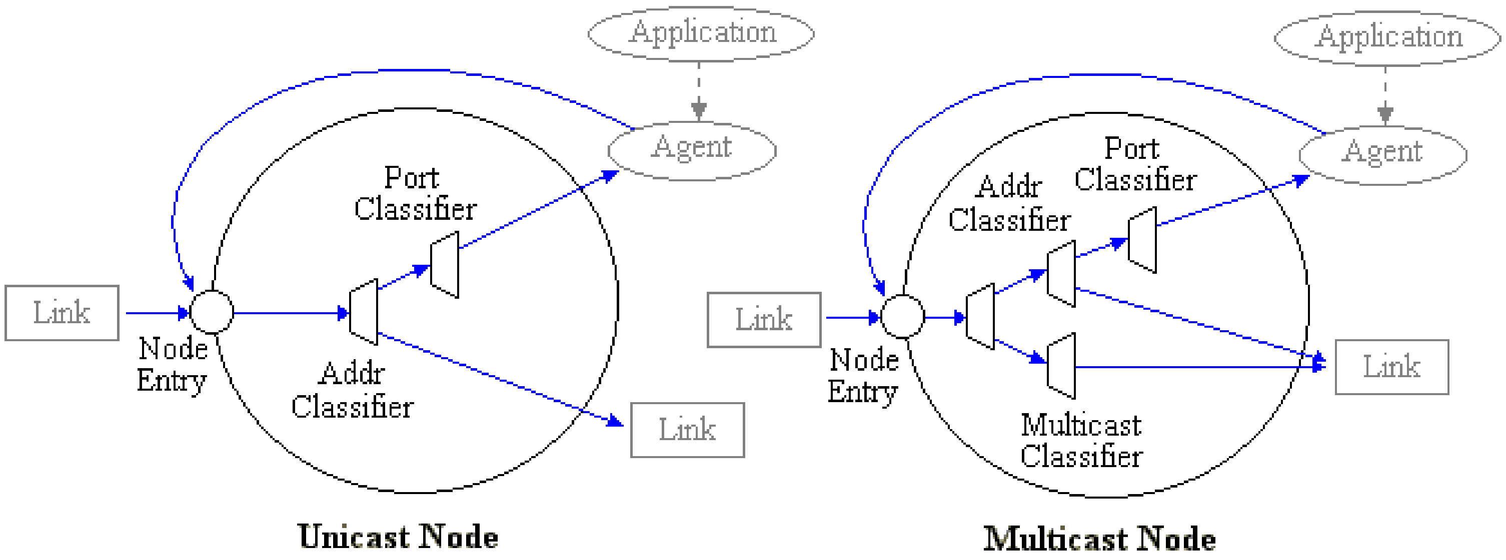
- Issue a special event called AtEvent at a specified simulation time



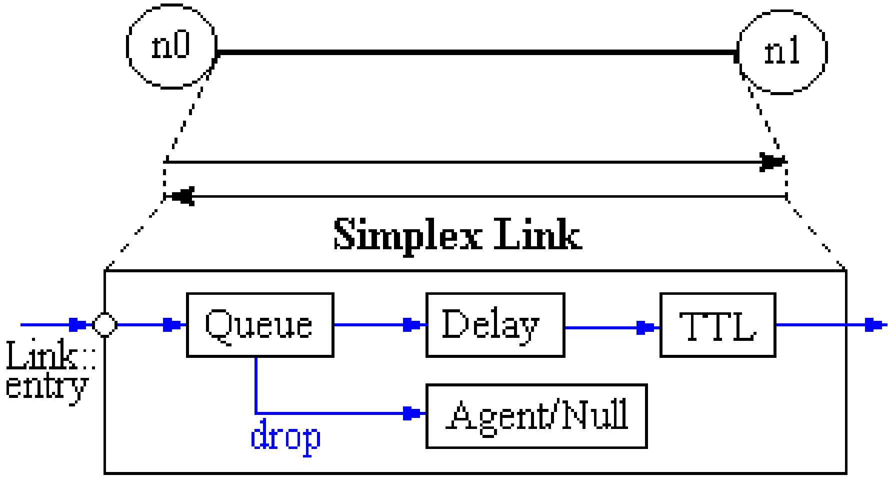
Network Components



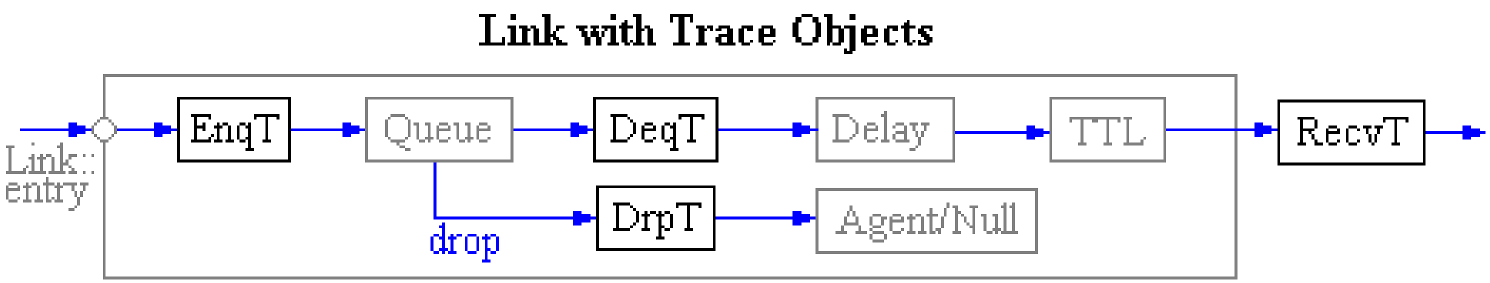
- Connector class - one output data path
- Classifier class - switching objects that have possible multiple output data paths



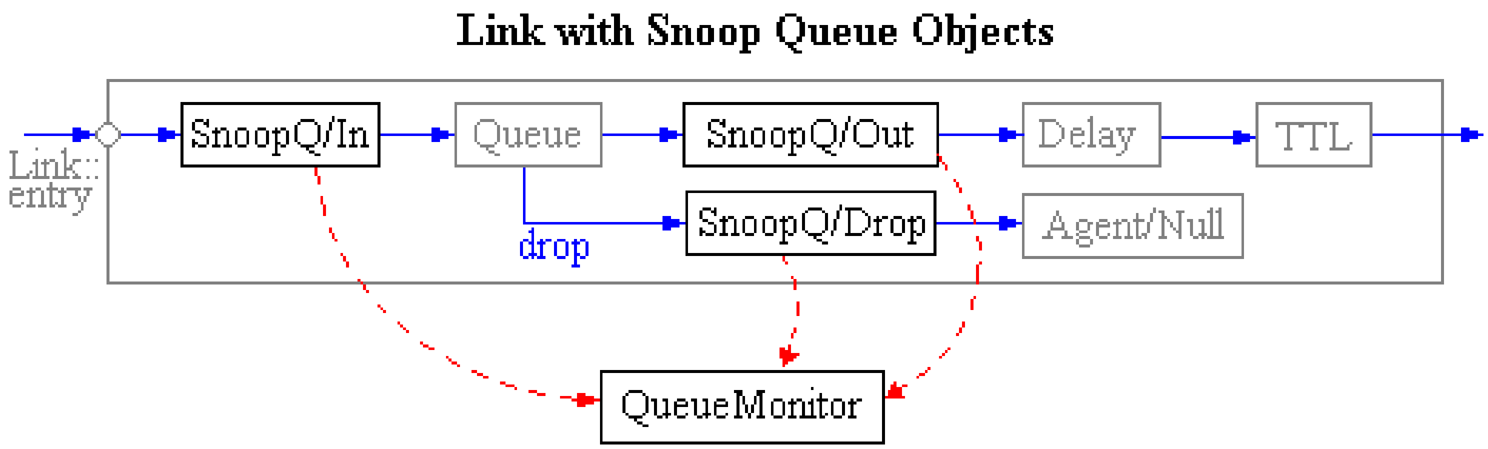
- Unicast nodes are the default nodes
- Multicast nodes must be explicitly notified in the input OTcl script



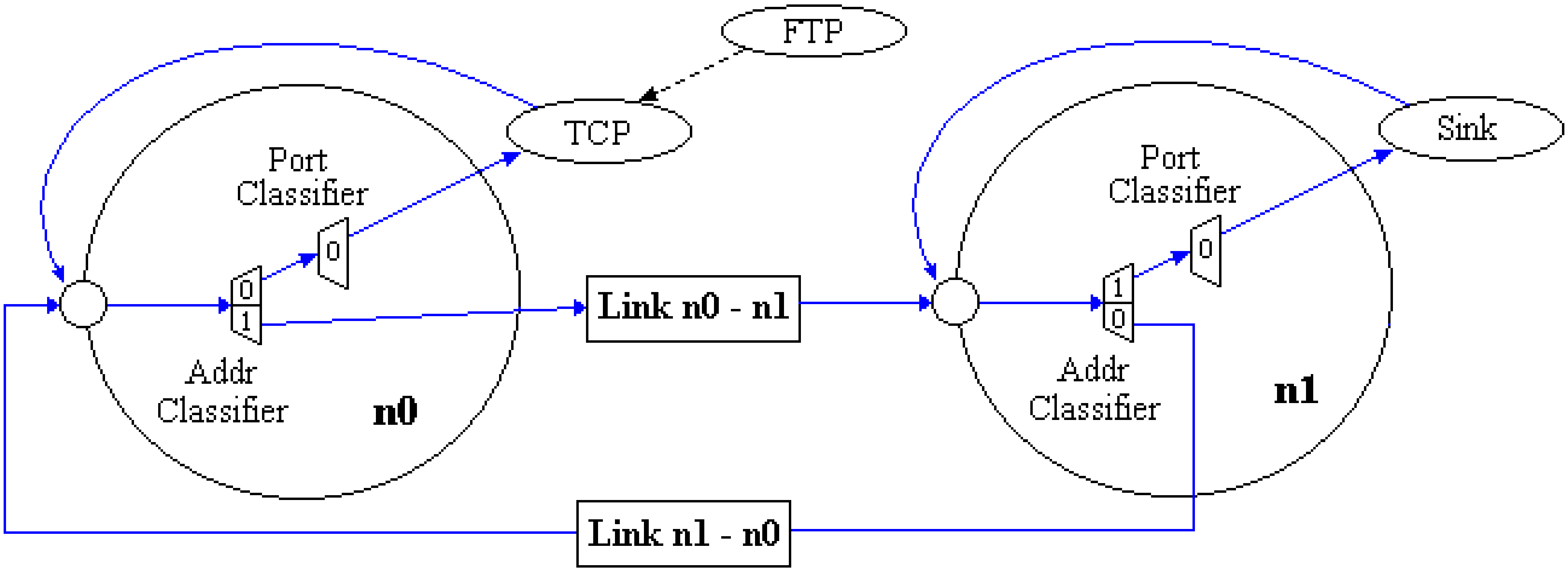
- Duplex-link = Two simplex links in both directions
- Queue as a part of simplex link



- Writes to the specified trace file without consuming any simulation tim

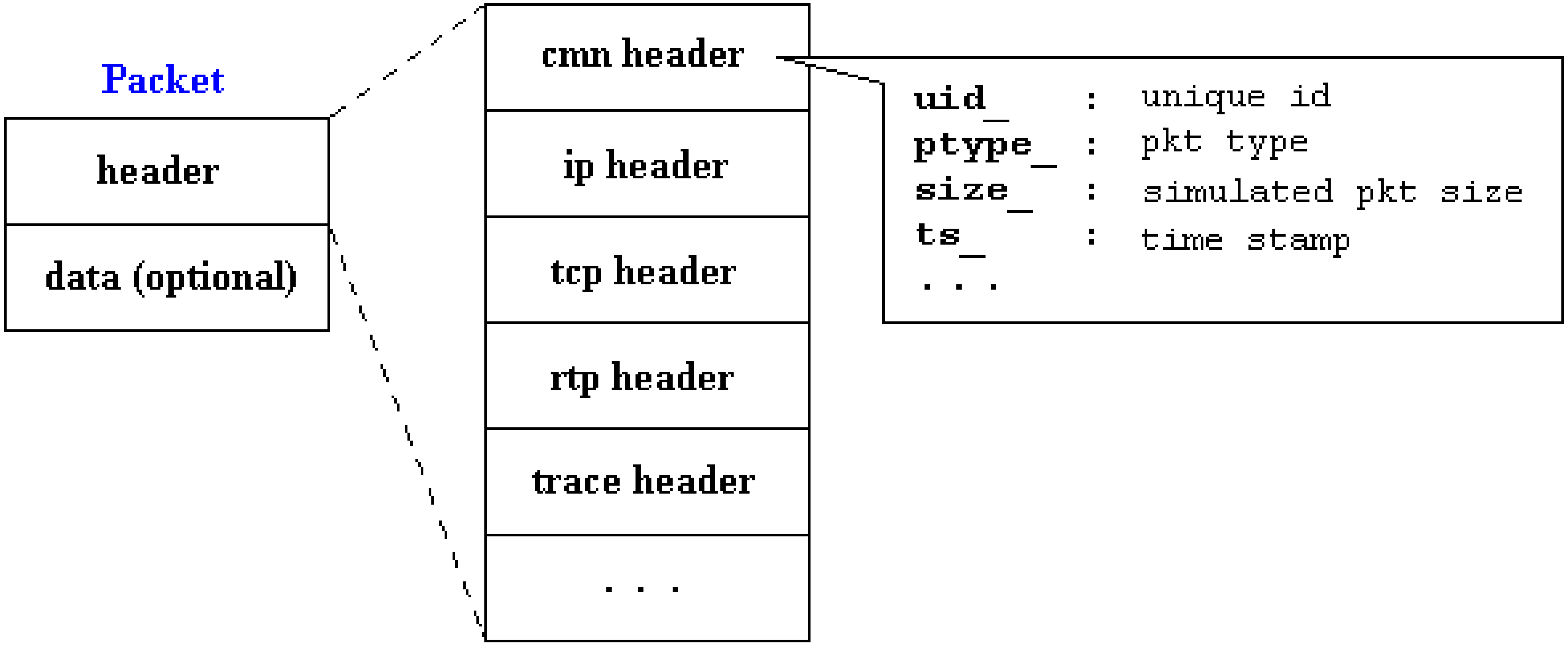


- Monitor what is going on inside a specific output queue.
- e.g. measure the dynamics of average queue size and current queue size of a specific RED queue


FTP
TCP
Sink
Port Classifier
0
0
1
Addr Classifier
n0
Link n0 - n1
Port Classifier
0
1
0
Addr Classifier
n1
Link n1 - n0

Packet

- Packets are handed from one network object to another using send(Packet* p) and recv(Packet*, Handler* h = 0) method of the receiver
- Composed of a stack of headers, and an optional data space
- Whether or not a specific header is used, a stack composed of all registered headers is created


Packet
header
data (optional)
cmn header
ip header
tcp header
rtp header
trace header
. . .
uid_ : unique id
ptype_ : pkt type
size_ : simulated pkt size
ts_ : time stamp
. . .

- A packet is allocated by an agent, and a network object can access any header in the stack of a packet it processes using the corresponding offset value
- Usually NOT to allocate data space, as meaningless to carry data around in a non-real-time simulation
- Usually create a new header for the application and modify the underlying agent to write data received from the application to the new header

Post Simulation

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
```

NS keeps track of UDP packet sequence number for analysis purposes

Reference

- NS2:
  http://www.isi.edu/nsnam/ns/
- The ns Manual
  http://www.isi.edu/nsnam/ns/ns-documentation.html
- NS by Example:
  http://nile.wpi.edu/NS/
- Marc Greis's tutorial
  http://www.isi.edu/nsnam/ns/tutorial/index.html