# NS2
# What already exist?

Chang-Gun Lee (cglee@snu.ac.kr)

Assistant Professor

The School of Computer Science and Engineering

Seoul National University

## Comprehensive Documents

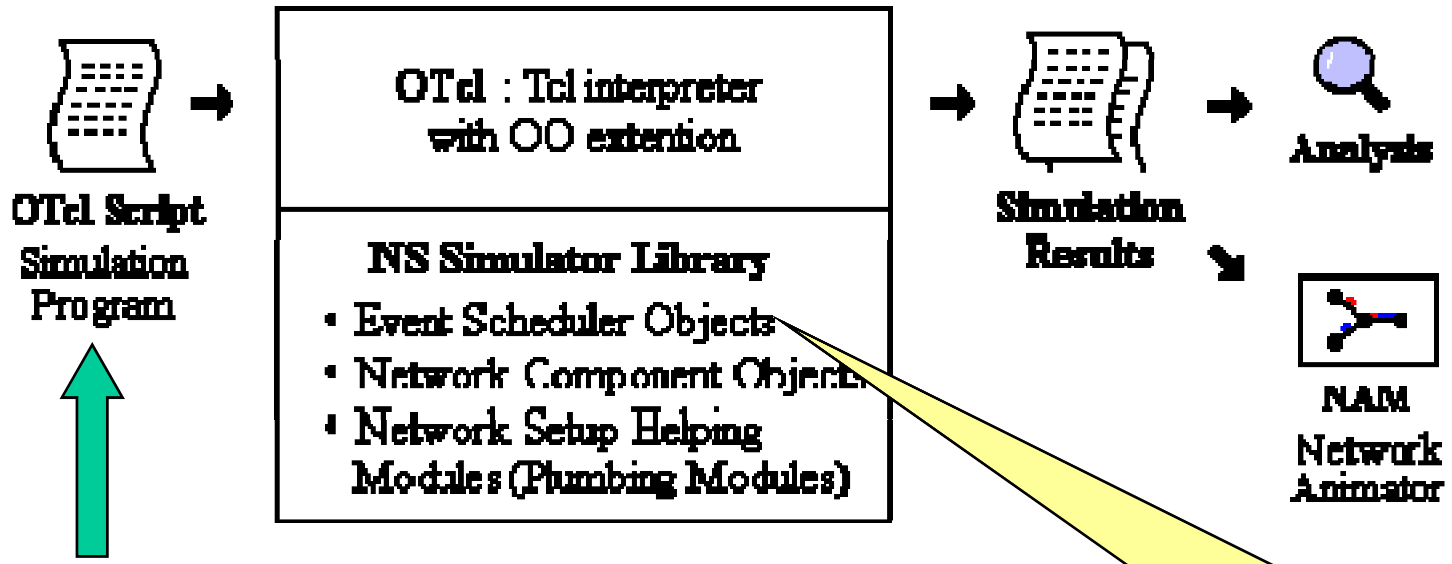
- NS2 Manual
  - http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf
- NS2 Class Hierarchy
  - http://www.isi.edu/nsnam/nsdoc-classes/hierarchy.html
  - http://www.isi.edu/nsnam/nsdoc-classes/classes.html

# NS-2 Built-In IP protocols

- MAC
  - Multicasting, LAN
- Router queue management
  - DropTail, SFQ, RED, CBQ
- Routing algorithms
  - Static shortest path, DV
- Transport protocols
  - TCP, UDP
- Traffic sources
  - FTP, Telent, Web, CBR, VBR, Burst

See NS manual for the comprehensive list and their usage

# User’s View of NS2



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

•Initiate an event scheduler

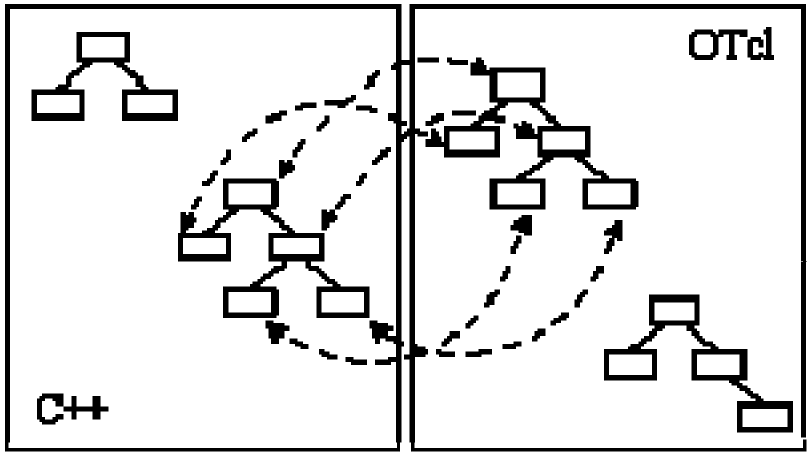
•Setup network topology using network component objects and plumbing functions

•Tells traffic sources when to start and stop through the event scheduler

NS core
•Manage global time
•Cause events and invoke appropriate components

# Why Two Languages (OTcl and C++)?

- For efficiency reasons
  - event scheduler and network component objects are written and compiled using C++
- For easy configuration
  - objects are configured by OTcl interpreter
  - For this, C++ objects are made available to the OTcl interpreter through OTcl linkage



NS is basically an OTcl interpreter with network simulation object libraries

# OTcl: The User Language

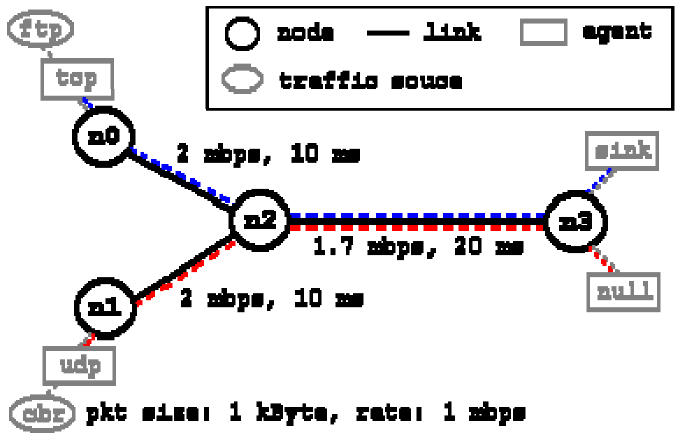
```
# Writing a procedure called 'test'
proc test {} {
     set a 43
     set b 27
     set c [expr $a + $b]
     set d [expr [expr $a - $b] * $c]
     for {set k 0} {$k < 10} {incr k} {
          if {$k < 5} {
               puts "k < 5, pow = [expr pow($d, $k)]"
          } else {
               puts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

# Calling the 'test' procedure created above
test
```

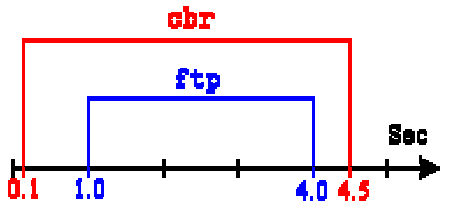
Tcl script example

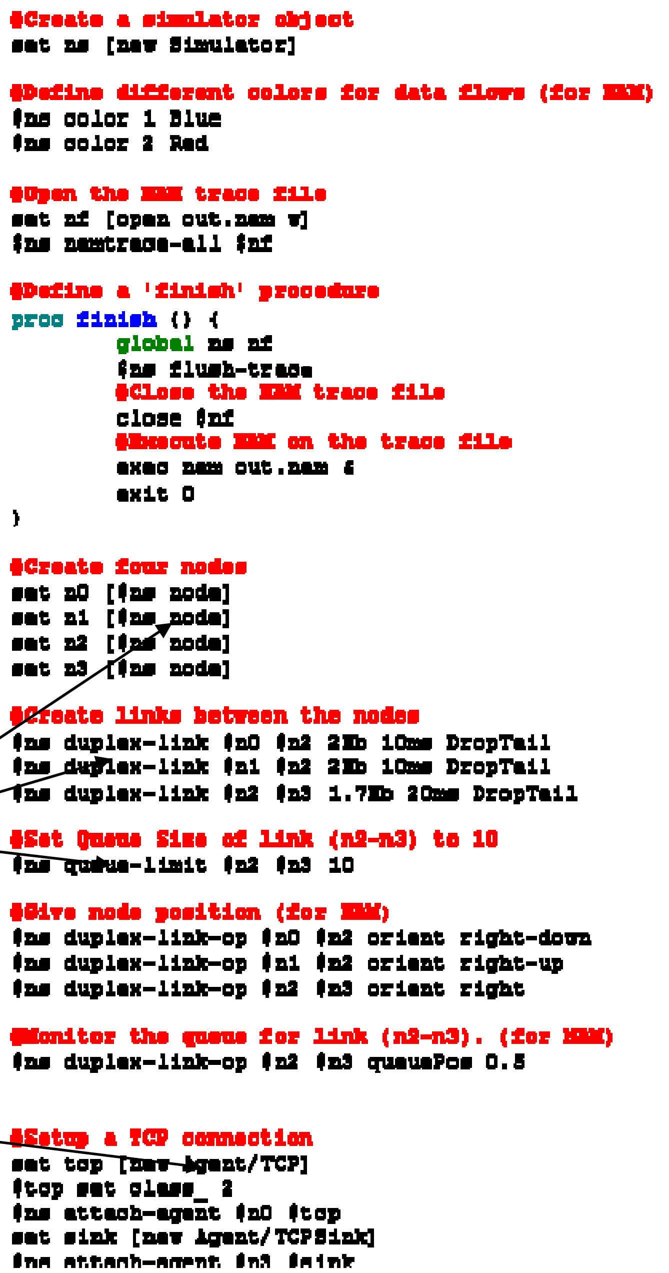
```
# add a member function call 'greet'
Class mom
mom instproc greet {} {
     $self instvar age_
     puts "$age_ year old mom say:
     How are you doing?"
}

# Create a child class of 'mom' called 'kid'
# and overide the member function 'greet'
Class kid -superclass mom
kid instproc greet {} {
     $self instvar age_
     puts "$age_ year old kid say:
     What's up, dude?"
}

# Create a mom and a kid object, set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function 'greet' of each object
$a greet
$b greet
```

OTcl script example

# Let’s see NS simulation script again







```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish () {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

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

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
```

Simulator object member functions: see ns-2/tcl/lib/ns-lib.tcl

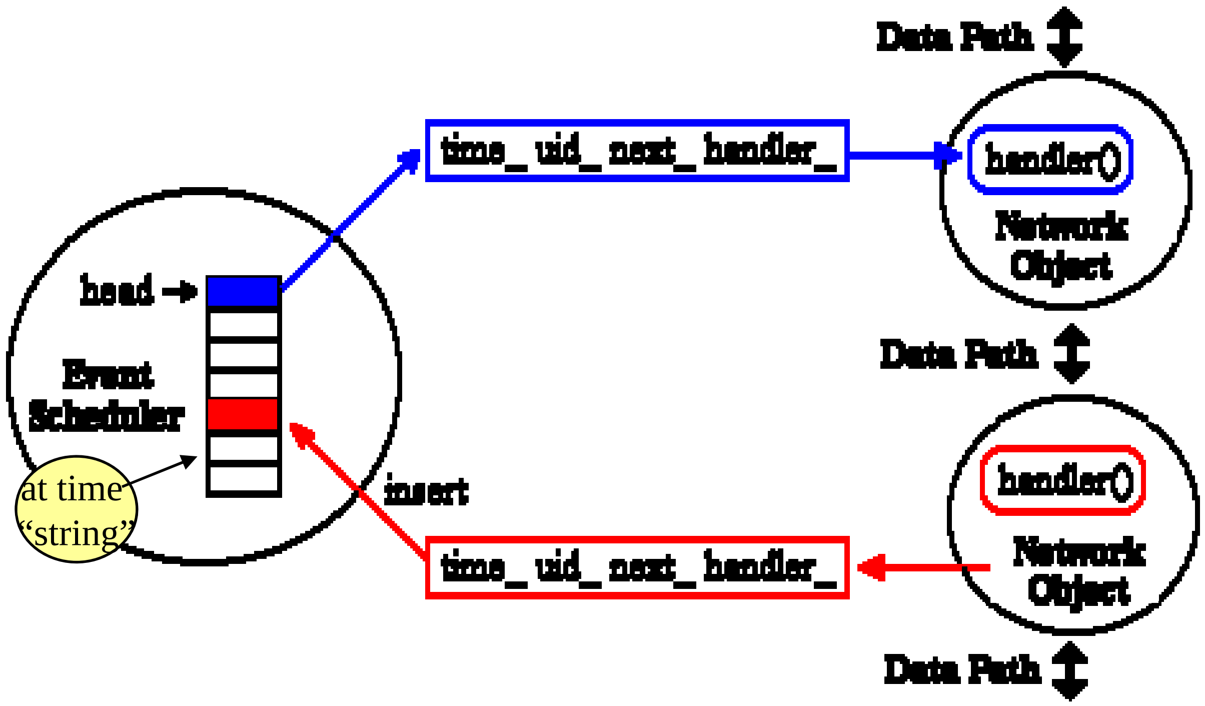
Agent object: to check what network objects are available, see ns-2/tcl/lib/ns-default.tcl

## What to do in OTcl script?

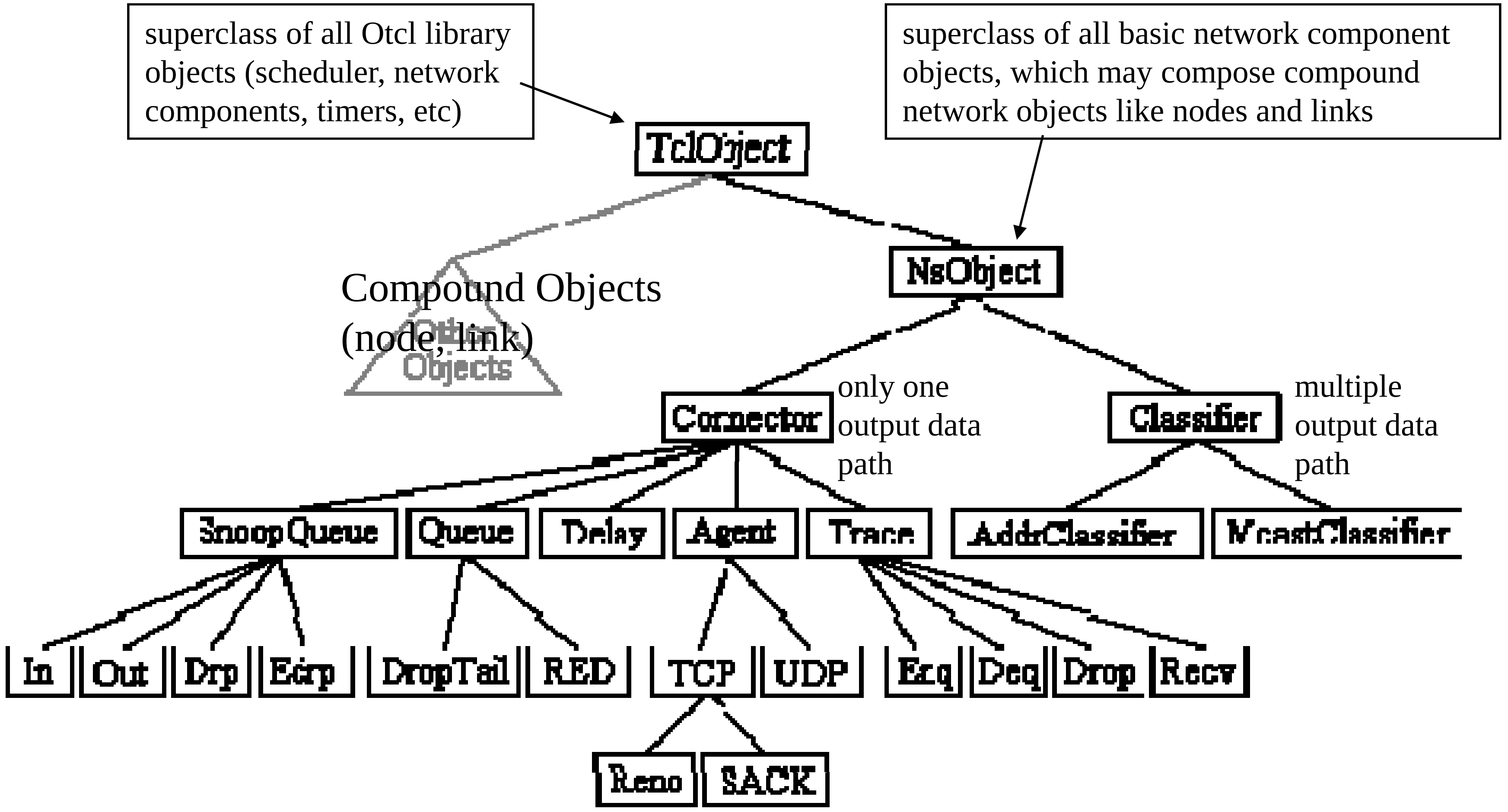
- Create Simulator object
- Specify output trace files and finish{}
- Network topology configuration
  - Create Node objects
  - Create Agents and attach them to nodes
  - Connect agents: build logical connections by setting the destination address to each others’ network and port address pair
  - Create “application” traffic sources and attach them to agents
- Write simulation scenario
  - Specify times when to start and stop traffic sources
  - $ns run

# How the scheduler works internally?

- Network components issue events for simulating packet-handling delay and timer
- The scheduler invokes the appropriate objects at the specified times
- The scheduler itself has a member function to issue an event
  - at time "$cbr start"

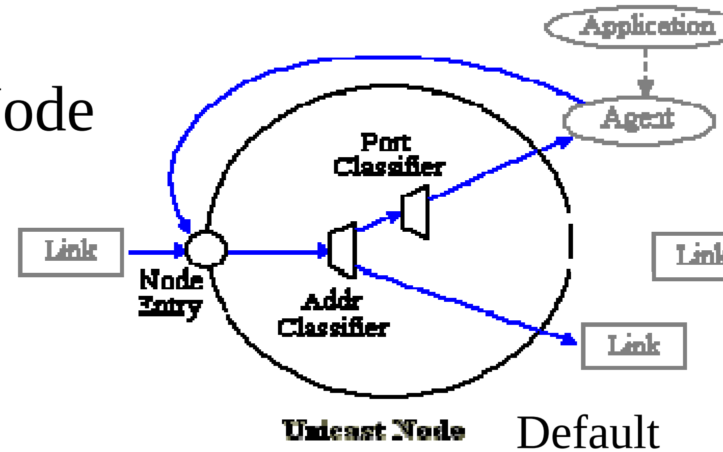


# Network Components


superclass of all Otcl library objects (scheduler, network components, timers, etc)
superclass of all basic network component objects, which may compose compound network objects like nodes and links
TclObject
NsObject
Compound Objects (node, link)
Other Objects
Connector
only one output data path
Classifier
multiple output data path
SnoopQueue
Queue
Delay
Agent
Trace
AddrClassifier
McastClassifier
In
Out
Drp
Edrp
DropTail
RED
TCP
UDP
Enq
Deq
Drop
Recv
Reno
SACK

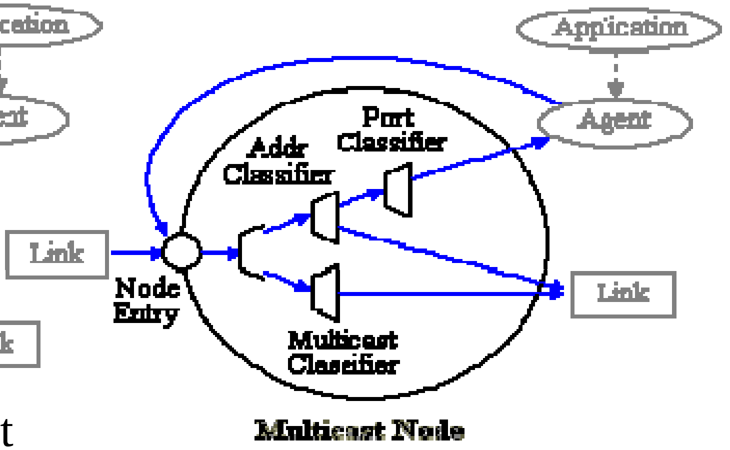
# Two Major Components (Compound Objects): Node and Link

Node



Default

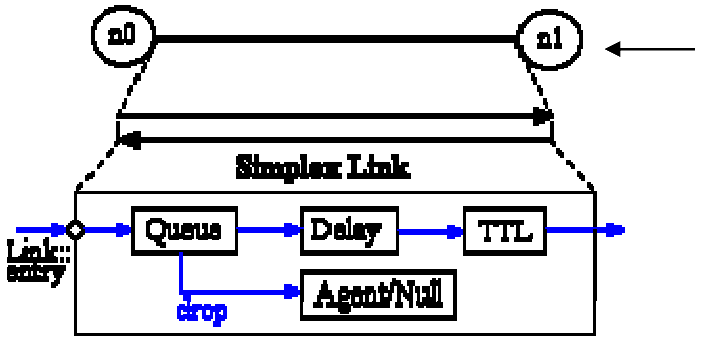
$ns rtproto type (Static, Session, DV, cost, multi-path)



$ns multicast (right after set $ns [new Scheduler]

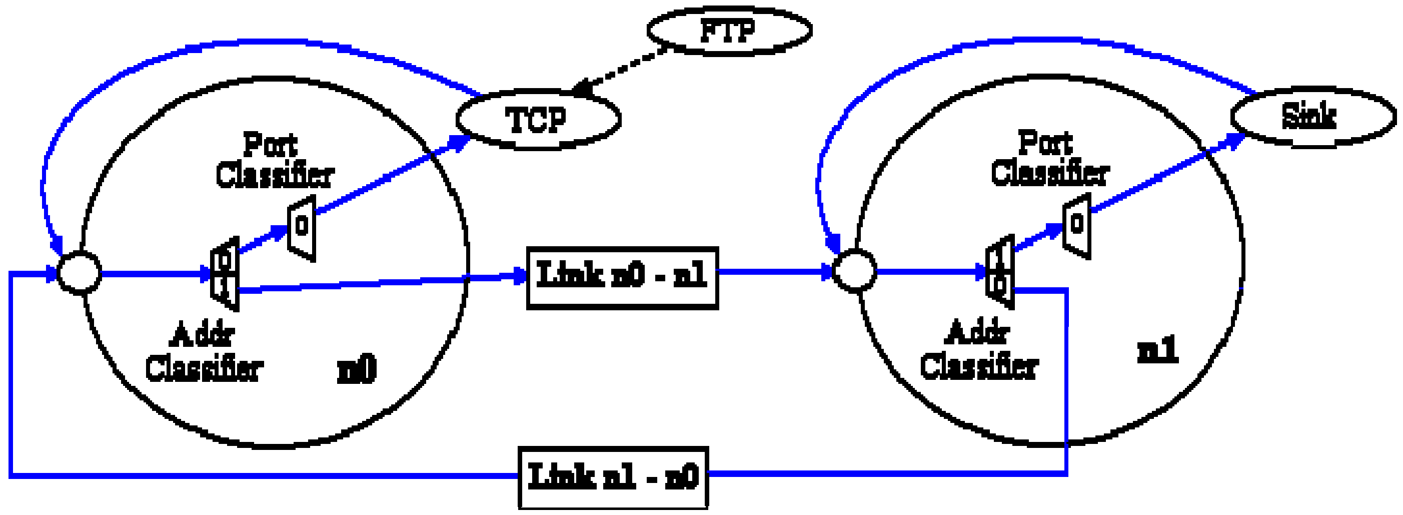
$ns mrtproto type (CtrMcast, DM, ST, BST)

Link



one duplex-link = TWO simplex-link

# Packet Flow



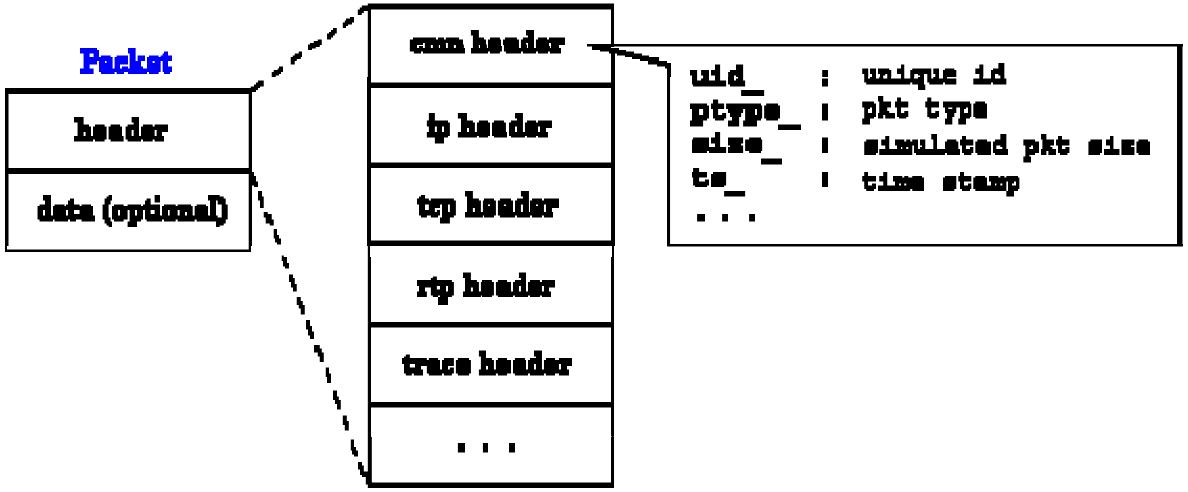
Packets are handed from one object to another using

send(Packet* p){target->recv(p)}: method of sender

recv(Packet*, Handler* h =0): method of receiver

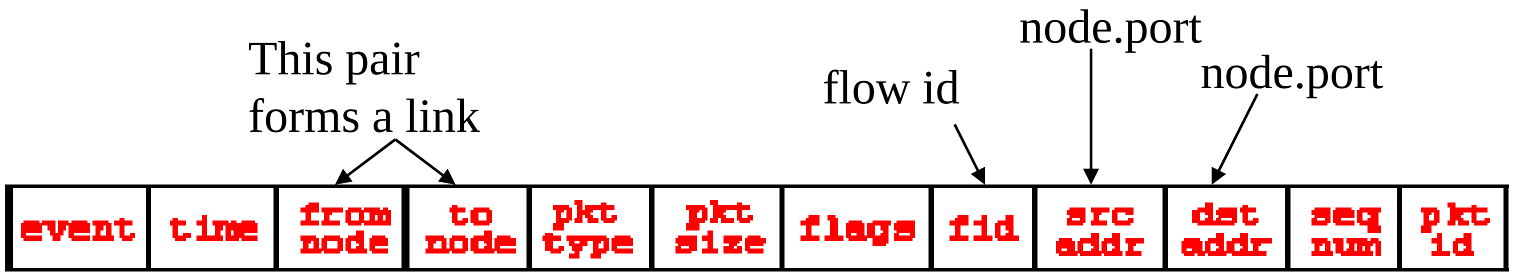
# Packet (We will revisit this later)

- A NS packet is composed of
  - a stack of headers for all the registered protocols (regardless whether any protocol is used or not)
  - an optional data space
- A network object can access any header of a packet using the corresponding offset value



- How to make a packet carry user-defined data?
  - modify the agent such that it can allocate the optional data space or
  - create new header and make the agent to use it as user-defined data

# Analyzing the trace output



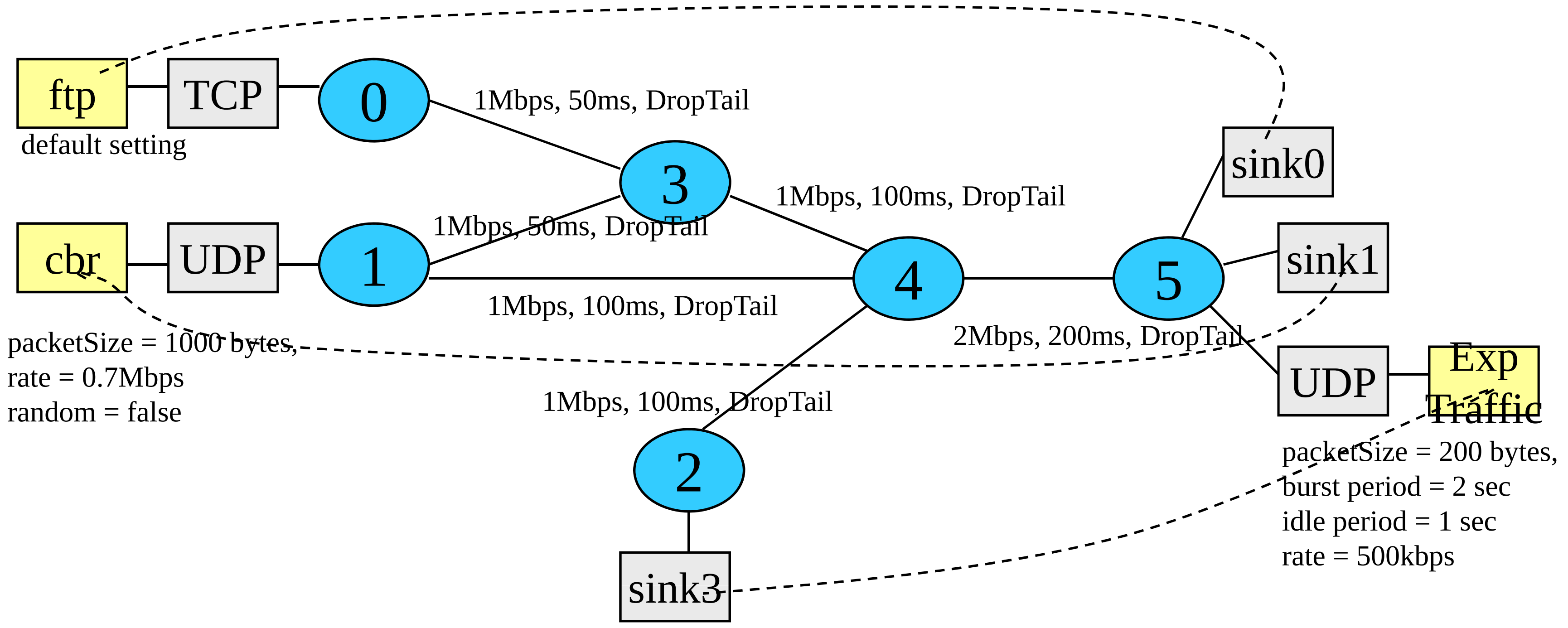
```
r : receive  (at to_node)
+ : enqueue  (at queue)
- : dequeue  (at queue)
d : drop     (at queue)
```

```
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
```

# Homework 7

- Simulate the following network



- Each application traffic source starts at 10 sec and stops at 20 sec.
- Plot the received bytes at each sink at every 0.5 sec period.