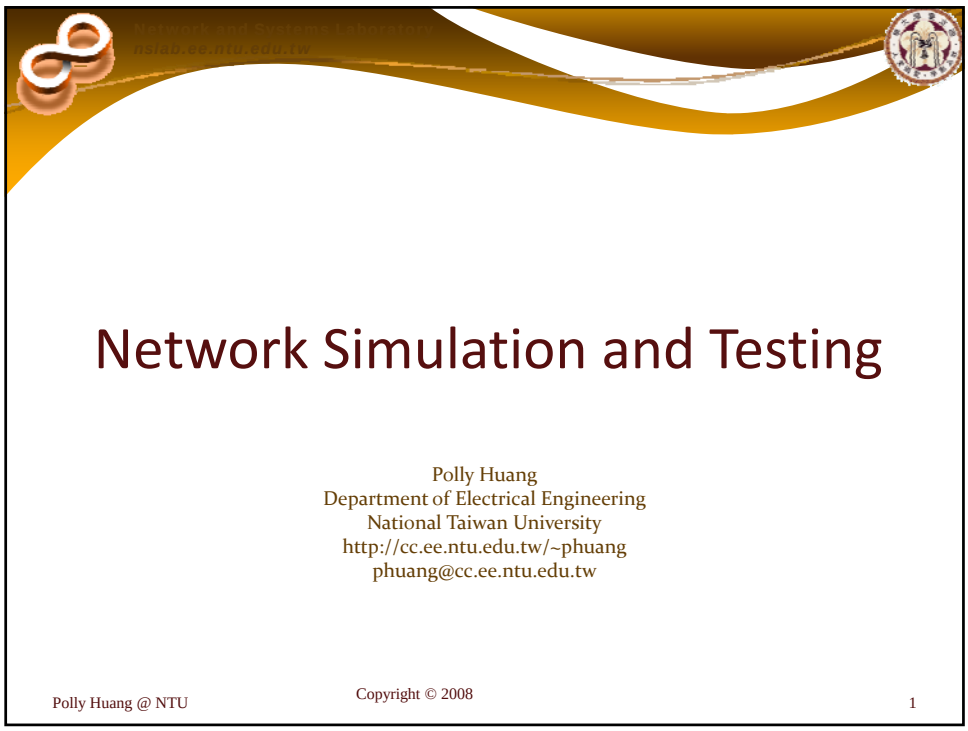
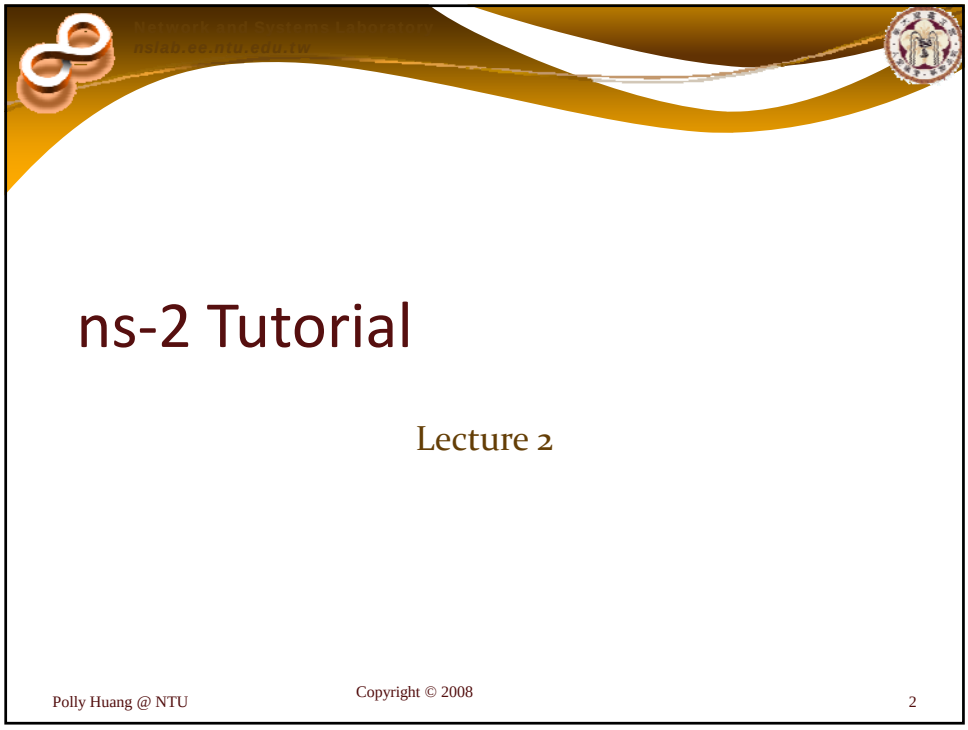


Network Simulation and Testing

Polly Huang
Department of Electrical Engineering
National Taiwan University
http://cc.ee.ntu.edu.tw/~phuang
phuang@cc.ee.ntu.edu.tw



ns-2 Tutorial

Lecture 2

Schedule: 3rd Week

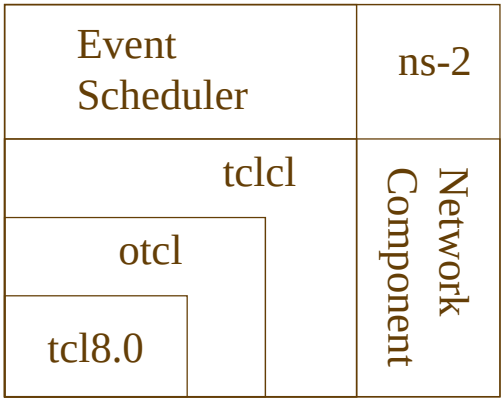
9.10-10.00 wired internal
10.20-11.10 wireless internal
11.20-12.10 extending ns-2

Schedule: 4th Week

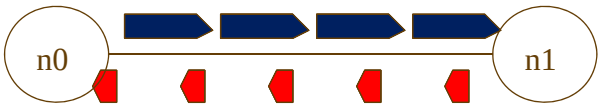
9.10-10.00 lab 4 intermediate ns-2 exercise
10.20-11.10 lab 5 getting data you want
11.20-12.10 lab 6 advanced topic

A Little Bit of Review

tcl Interpreter With Extensions



Example Script




```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1.5Mb
    10ms DropTail
set tcp [$ns create-connection
    TCP $n0 TCPSink $n1 0]
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.2 "$ftp start"
$ns at 1.2 ”exit"
$ns run
```


Basic ns-2: Covered

- wired & wireless
- unicast & multicast
- TCP & UDP
- errors & network dynamics
- ns & nam tracing
- application-level support

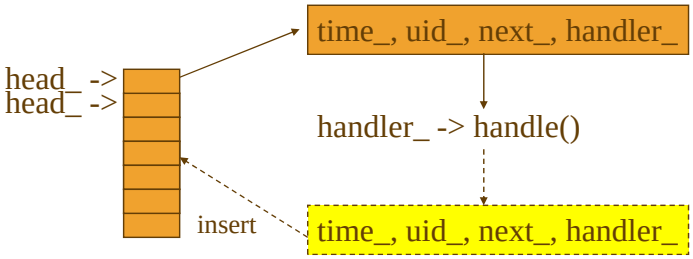
Outline for Today

- **ns-2 Internal**
- Making changes
- New components
  - in otcl
  - otcl and C++ Linkage
  - in C++
- Debugging

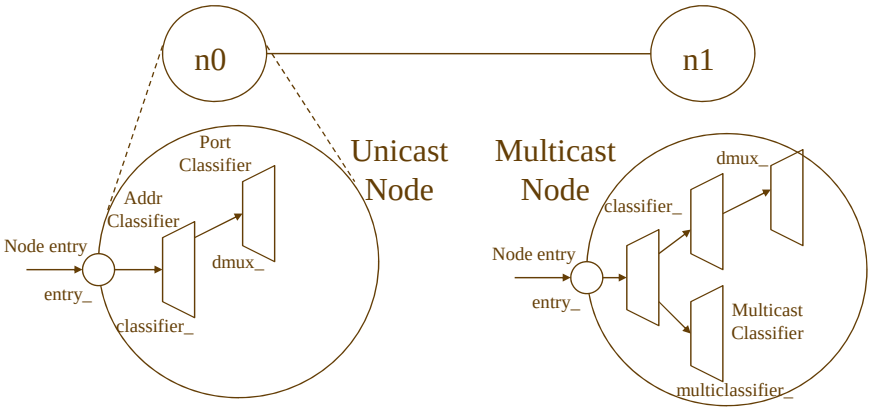
ns-2 Internals

- Discrete Event Scheduler
- Network Topology
- Routing
- Transport
- Application
- Packet Flow
- Packet Format

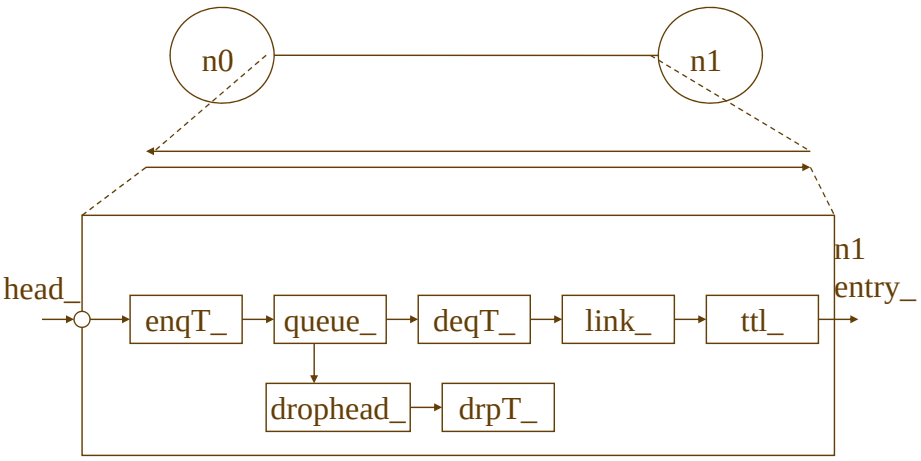
Discrete Event Scheduler



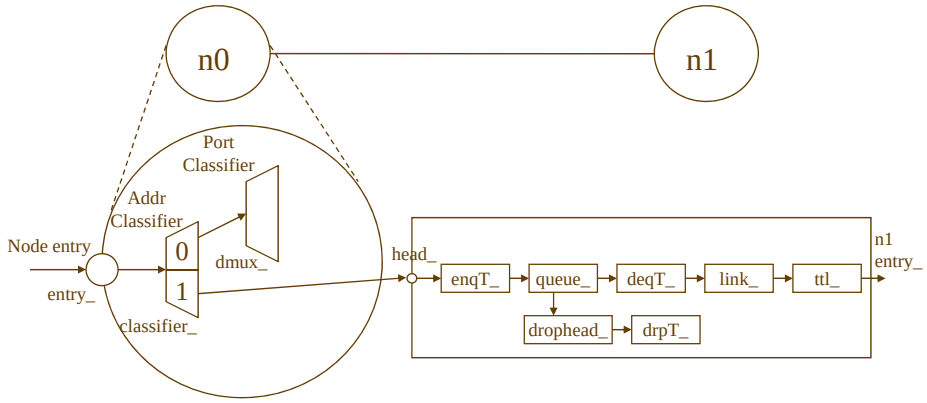
Network Topology - Node



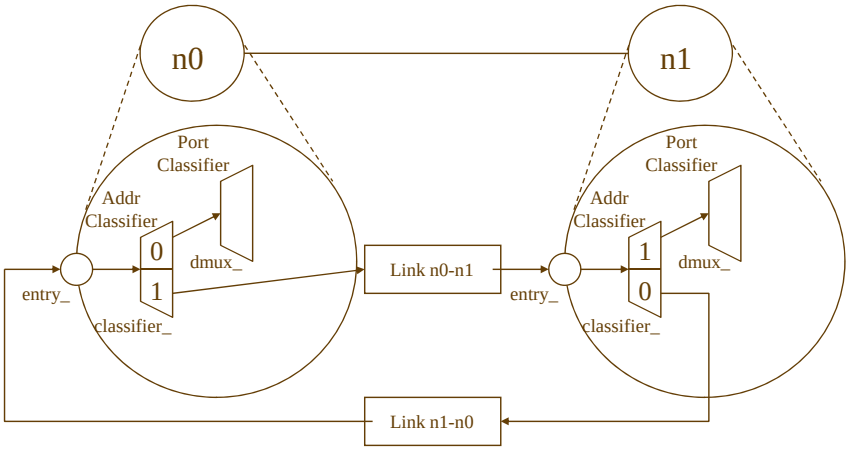
Network Topology - Link



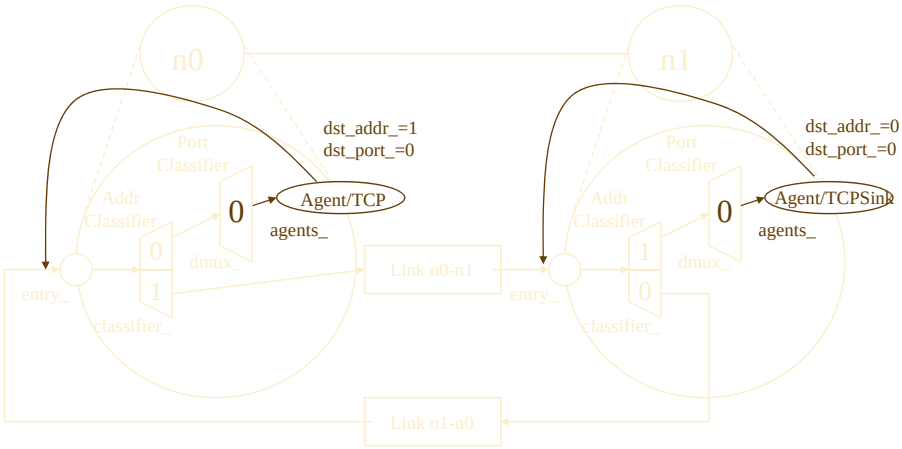
Routing



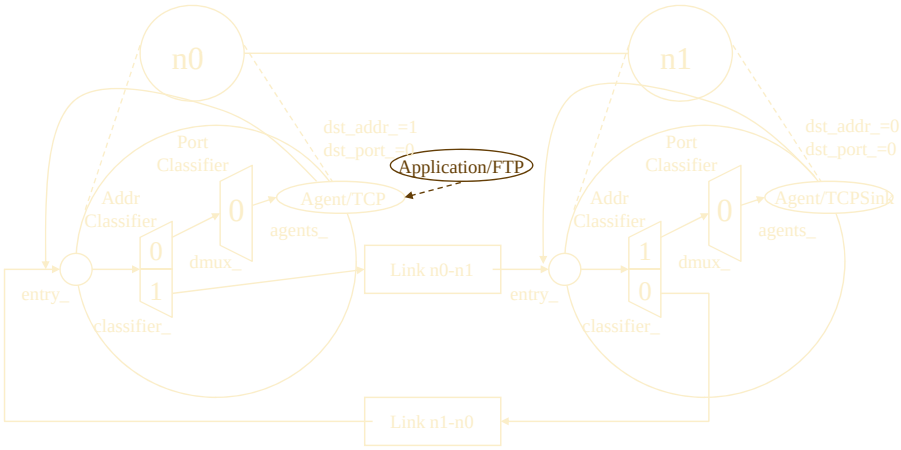
Routing (cont.)



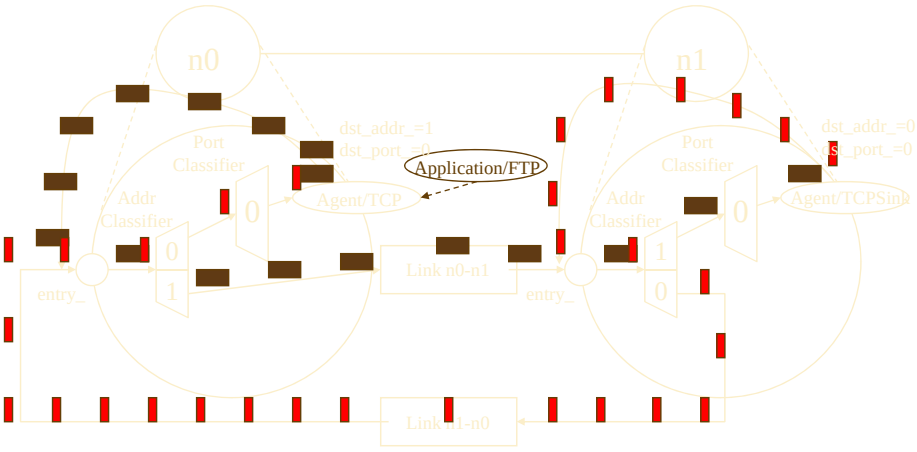
Transport



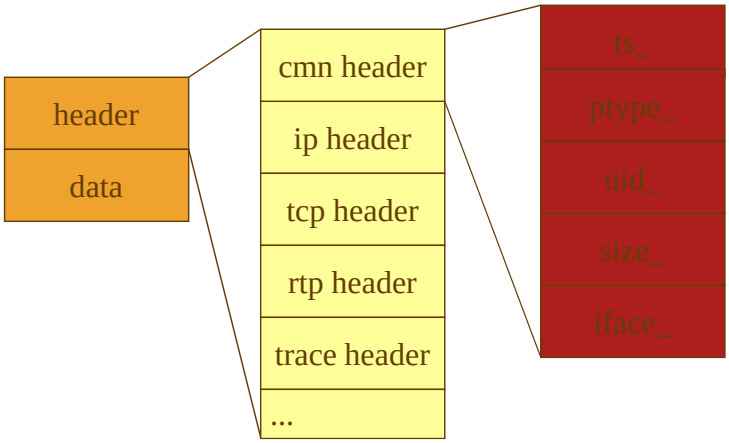
Application



Packet Flow



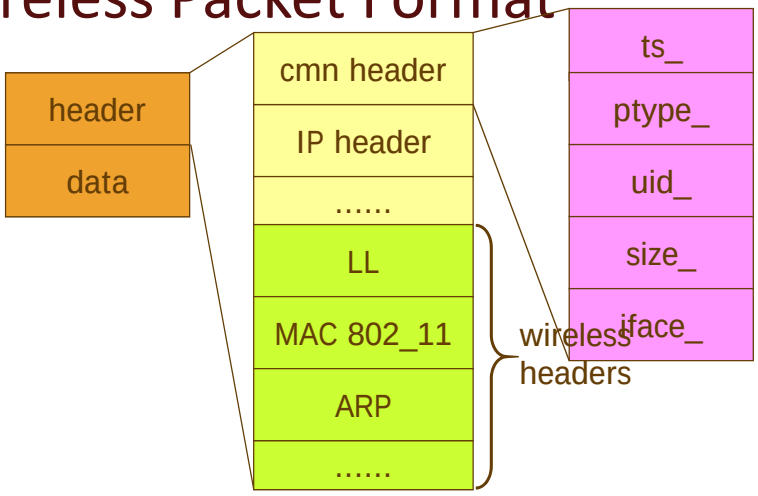
Packet Format



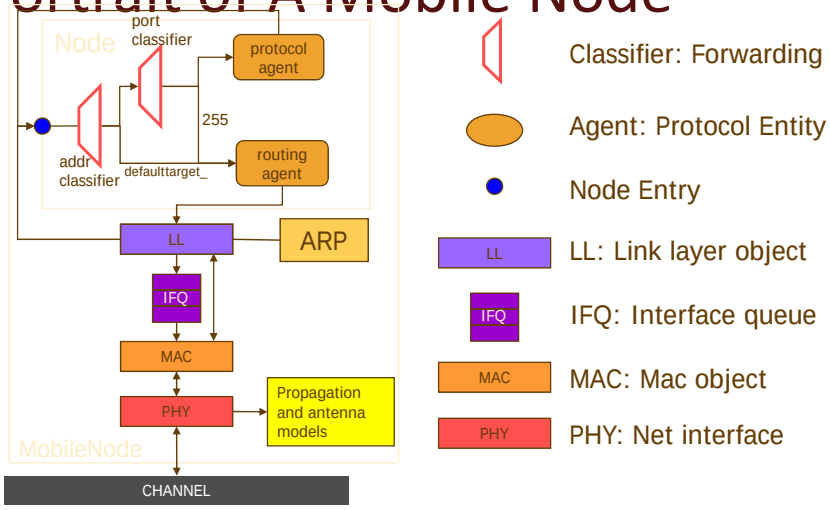
ns-2 Wireless Internal

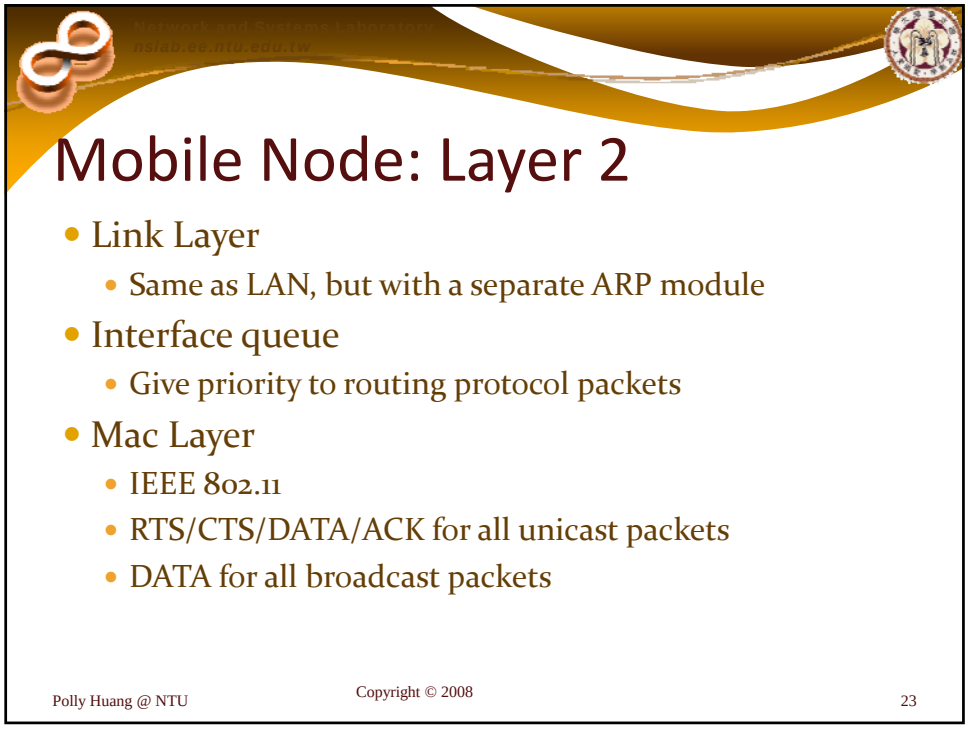
- Packet headers
- Mobile node
- Wireless channel

Wireless Packet Format



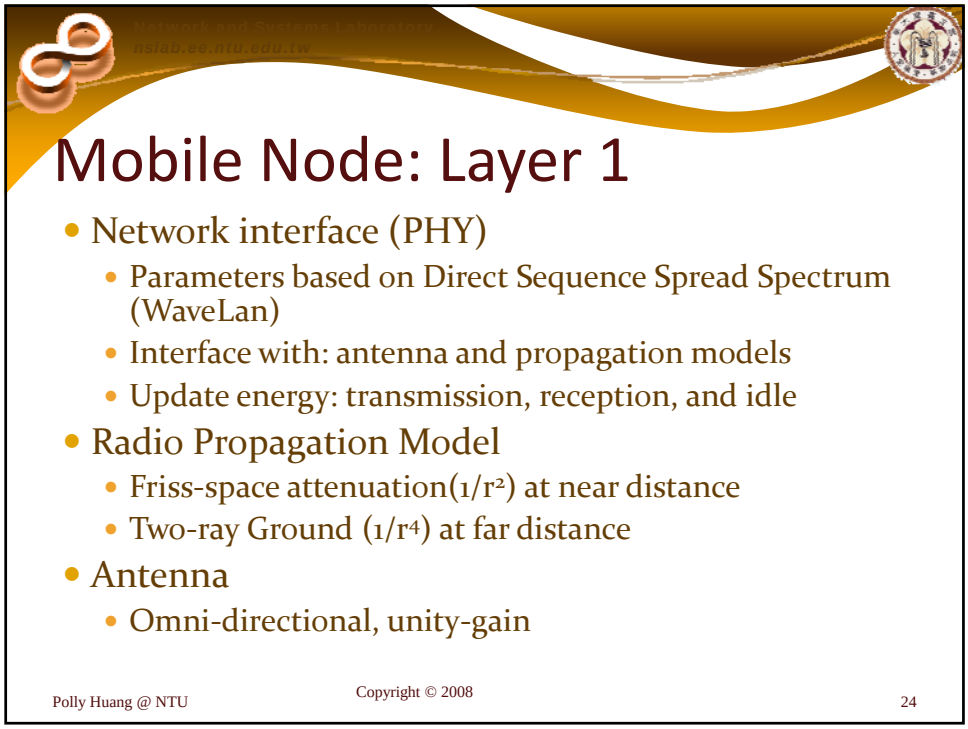
Portrait of A Mobile Node





Mobile Node: Layer 2

- Link Layer
  - Same as LAN, but with a separate ARP module
- Interface queue
  - Give priority to routing protocol packets
- Mac Layer
  - IEEE 802.11
  - RTS/CTS/DATA/ACK for all unicast packets
  - DATA for all broadcast packets



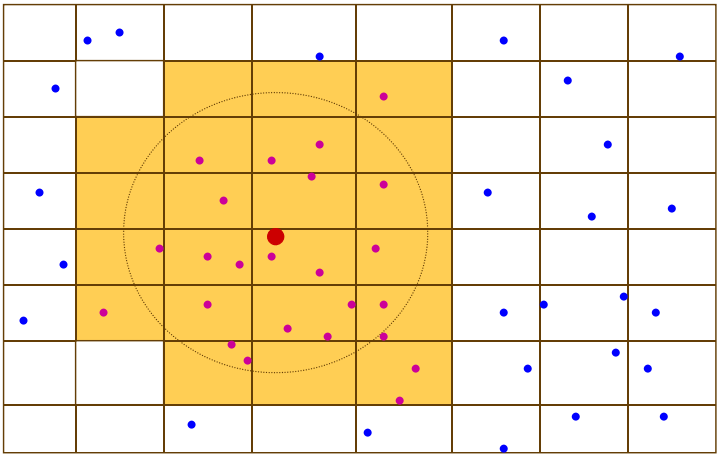
Mobile Node: Layer 1

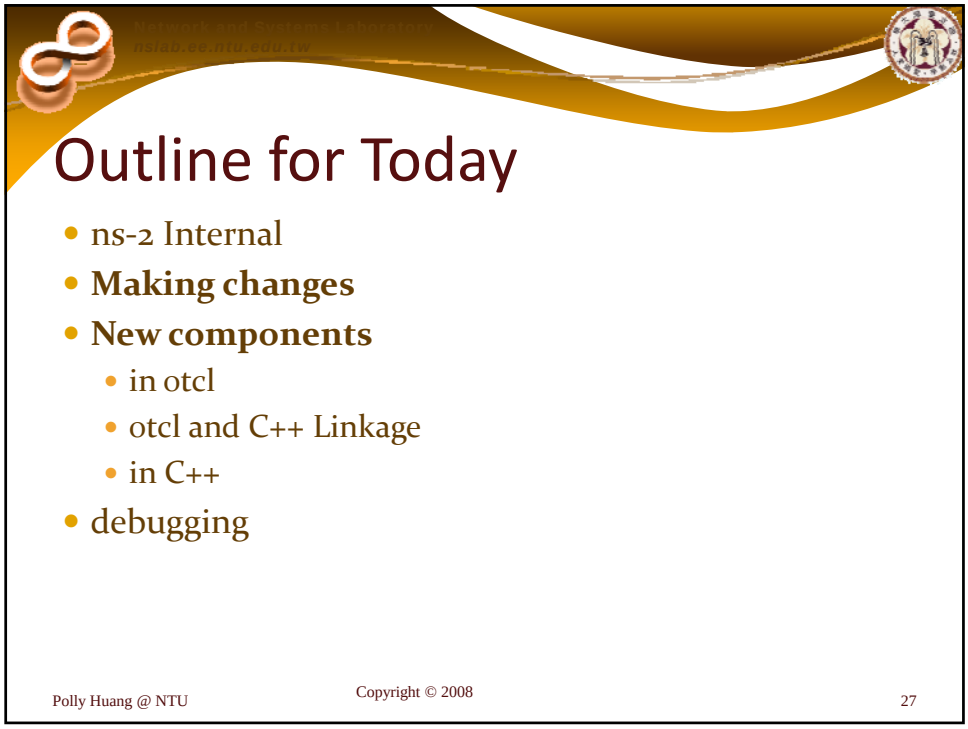
- Network interface (PHY)
  - Parameters based on Direct Sequence Spread Spectrum (WaveLan)
  - Interface with: antenna and propagation models
  - Update energy: transmission, reception, and idle
- Radio Propagation Model
  - Friss-space attenuation($1/r^2$) at near distance
  - Two-ray Ground ($1/r^4$) at far distance
- Antenna
  - Omni-directional, unity-gain

Wireless Channel

- Duplicate packets to all mobile nodes attached to the channel except the sender
- It is the receiver's responsibility to decide if it will accept the packet
  - Whether the sender is close enough
  - Collision is also handled at individual receivers
  - $O(N^2)$ messages → grid keeper

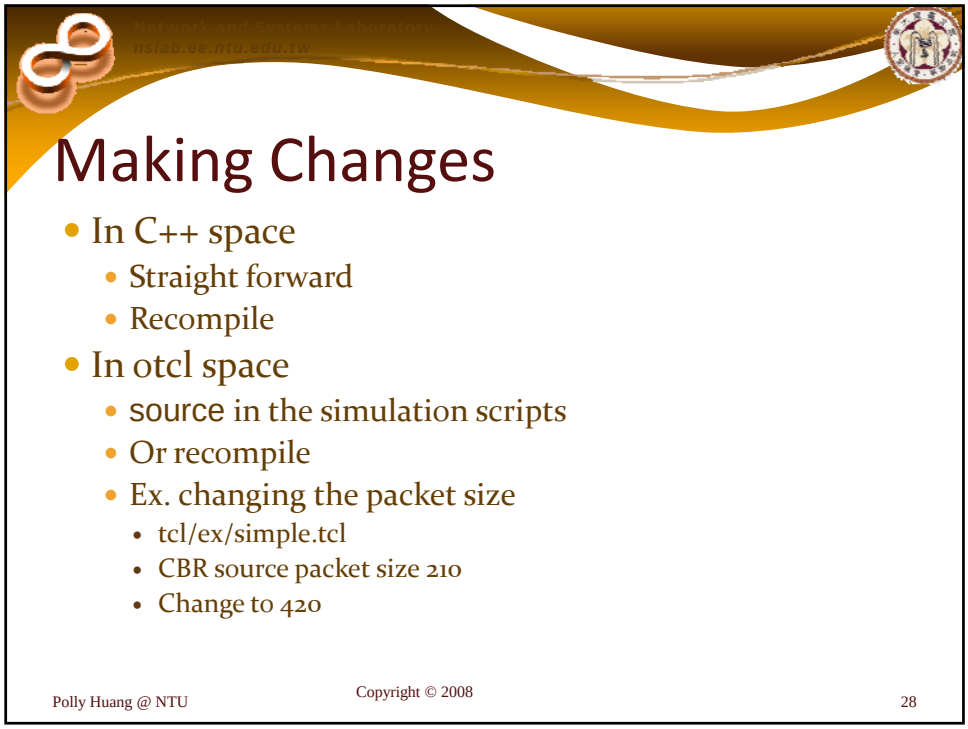
Grid-keeper: An Optimization





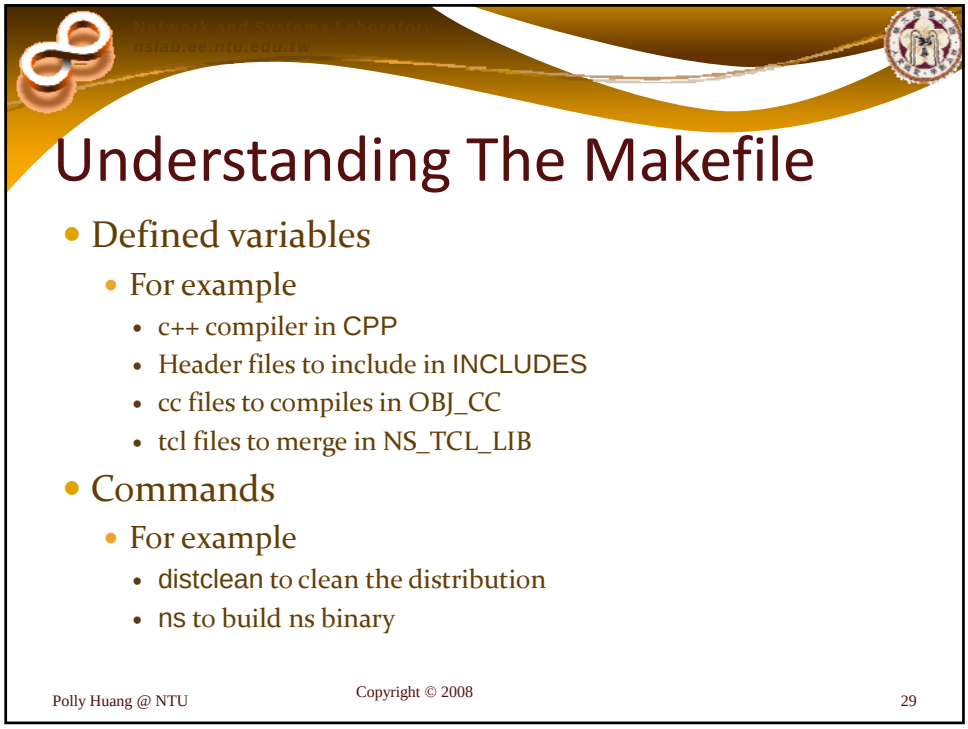
Outline for Today

- ns-2 Internal
- **Making changes**
- **New components**
  - in otcl
  - otcl and C++ Linkage
  - in C++
- debugging



Making Changes

- In C++ space
  - Straight forward
  - Recompile
- In otcl space
  - source in the simulation scripts
  - Or recompile
  - Ex. changing the packet size
    - tcl/ex/simple.tcl
    - CBR source packet size 210
    - Change to 420



Understanding The Makefile

- Defined variables
  - For example
    - c++ compiler in CPP
    - Header files to include in INCLUDES
    - cc files to compiles in OBJ_CC
    - tcl files to merge in NS_TCL_LIB
- Commands
  - For example
    - distclean to clean the distribution
    - ns to build ns binary

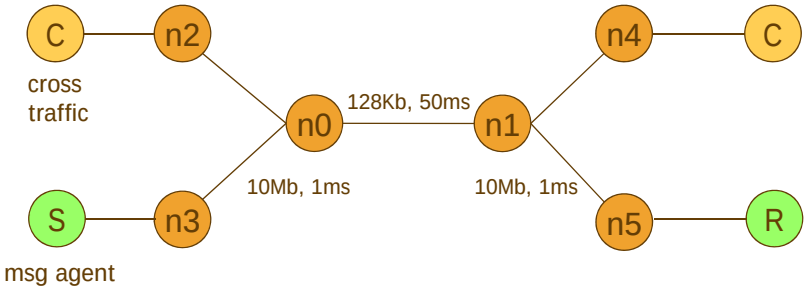
Adding New Components

- **in otcl**
- otcl and C++ linkage
- in C++

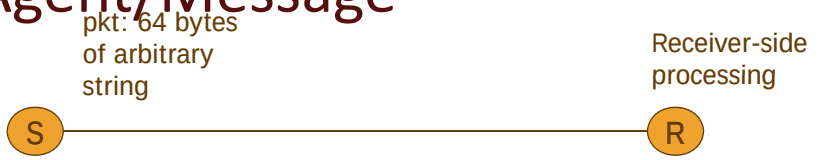
New Component Purely in otcl

- Additional <new_stuff>.tcl file
- Adding new files
  - change Makefile (NS_TCL_LIB)
  - source in tcl/lib/ns-lib.tcl
  - recompile

Example: Agent/Message



Agent/Message

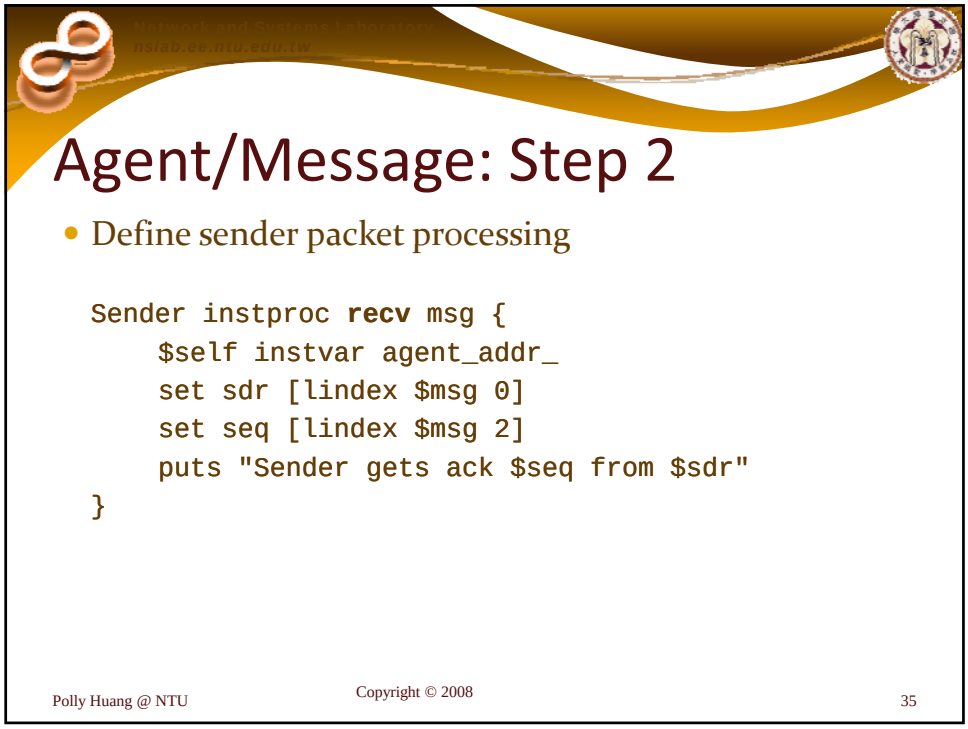


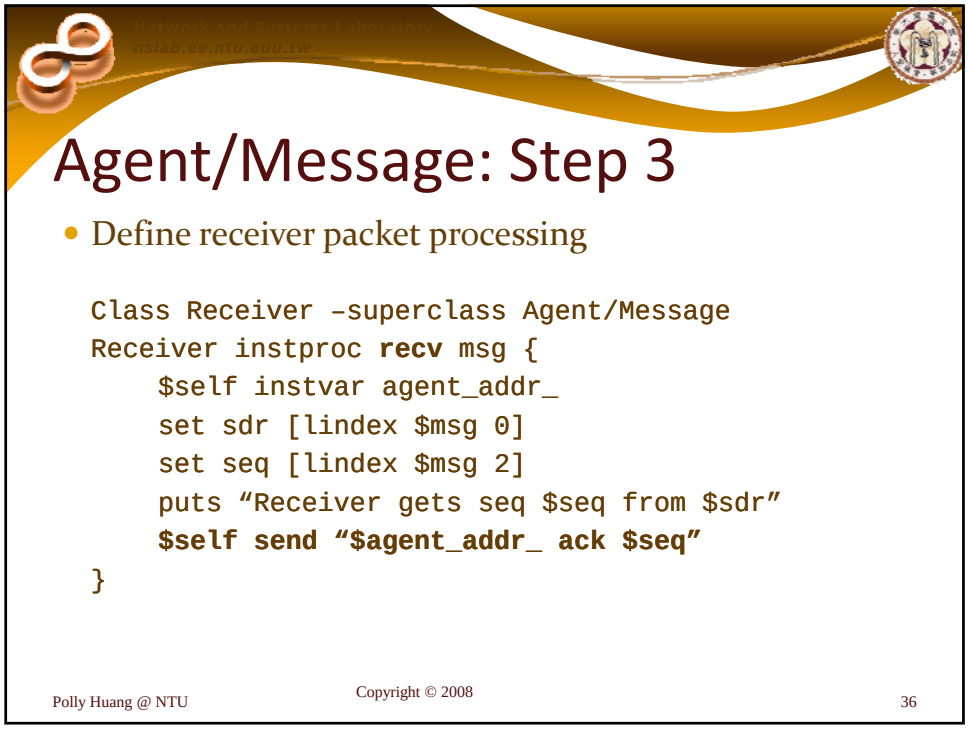
- A UDP agent (without UDP header)
- Up to 64 bytes user message
- Good for fast prototyping a simple idea
- Usage requires extending ns functionality

Agent/Message: Step 1

- Define sender

```
class Sender –superclass Agent/Message

# Message format: “Addr Op SeqNo”
Sender instproc send-next {} {
     $self instvar seq_ agent_addr_
     $self send “$agent_addr_ send $seq_”
     incr seq_
     global ns
     $ns at [expr [$ns now]+0.1] "$self send-next"
}
```


Network and Systems Laboratory
nslab.ee.ntu.edu.tw
Agent/Message: Step 2
Define sender packet processing
Sender instproc recv msg {
$self instvar agent_addr_
set sdr [lindex $msg 0]
set seq [lindex $msg 2]
puts "Sender gets ack $seq from $sdr"
}
Polly Huang @ NTU
Copyright © 2008
35


Network and Systems Laboratory
nslab.ee.ntu.edu.tw
Agent/Message: Step 3
Define receiver packet processing
Class Receiver –superclass Agent/Message
Receiver instproc recv msg {
$self instvar agent_addr_
set sdr [lindex $msg 0]
set seq [lindex $msg 2]
puts “Receiver gets seq $seq from $sdr”
$self send “$agent_addr_ ack $seq”
}
Polly Huang @ NTU
Copyright © 2008
36

Agent/Message: Step 4

- Scheduler and tracing

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

# Turn on Tracing
set fd [new “message.nam” w]
$ns namtrace-all $fd
```


Agent/Message: Step 5

- Topology

```
for {set i 0} {$i < 6} {incr i} {
     set n($i) [$ns node]
}
$ns duplex-link $n(0) $n(1) 128kb 50ms DropTail
$ns duplex-link $n(1) $n(4) 10Mb 1ms DropTail
$ns duplex-link $n(1) $n(5) 10Mb 1ms DropTail
$ns duplex-link $n(0) $n(2) 10Mb 1ms DropTail
$ns duplex-link $n(0) $n(3) 10Mb 1ms DropTail

$ns queue-limit $n(0) $n(1) 5
$ns queue-limit $n(1) $n(0) 5
```

Agent/Message: Step 6

- Routing

```
# Packet loss produced by queueing

# Routing protocol: let's run distance vector
$ns rtproto DV
```


Agent/Message: Step 7

- Cross traffic

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(2) $udp0
set null0 [new Agent/NULL]
$ns attach-agent $n(4) $null0
$ns connect $udp0 $null0

set exp0 [new Application/Traffic/Exponential]
$exp0 set rate_ 128k
$exp0 attach-agent $udp0
$ns at 1.0 "$exp0 start"
```

Agent/Message: Step 8

- Message agents

```
set sdr [new Sender]
$sdr set packetSize_ 1000

set rcvr [new Receiver]
$rcvr set packetSize_ 40

$ns attach $n(3) $sdr
$ns attach $n(5) $rcvr
$ns connect $sdr $rcvr
$ns connect $rcvr $sdr
$ns at 1.1 “$sdr send-next”
```


Agent/Message: Step 9

- End-of-simulation wrapper (as usual)

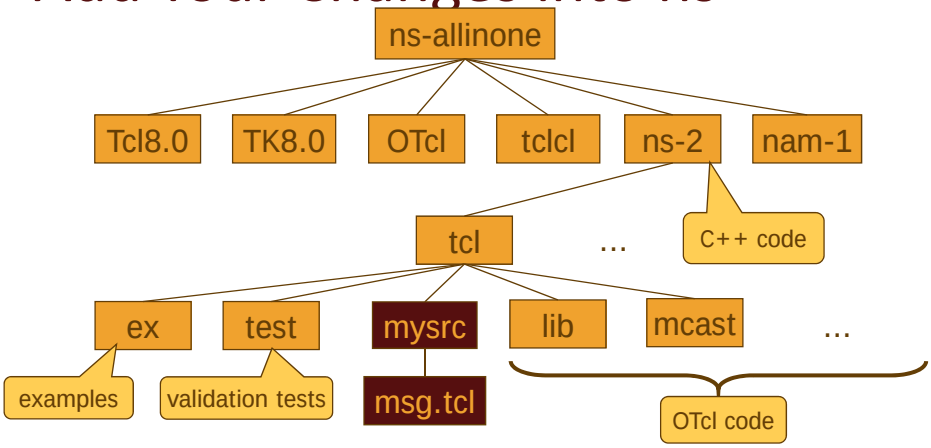
```
$ns at 2.0 finish
proc finish {} {
    global ns fd
    $ns flush-trace
    close $fd
    exit 0
}
```

Agent/Message: Result

- Example output

```
> ./ns msg.tcl
Receiver gets seq 0 from 0
Sender gets ack 0 from 1
Receiver gets seq 1 from 0
Sender gets ack 1 from 1
Receiver gets seq 2 from 0
Sender gets ack 2 from 1
Receiver gets seq 3 from 0
Sender gets ack 3 from 1
Receiver gets seq 4 from 0
Sender gets ack 4 from 1
Receiver gets seq 5 from 0
```


Add Your Changes into ns



Add Your Change into ns

- tcl/lib/ns-lib.tcl

```
Class Simulator
…
source ../mysrc/msg.tcl
```

- Makefile

```
NS_TCL_LIB = \
  tcl/mysrc/msg.tcl \
  …
```

  - Or: change Makefile.in, `make distclean`, then `./configure`

Adding New Components

- In otcl
- **otcl and C++ linkage**
- **In C++**

Extending ns in C++

- Adding code in <new_stuff>.{cc,h} files
  - Change Makefile
  - make depend
  - recompile

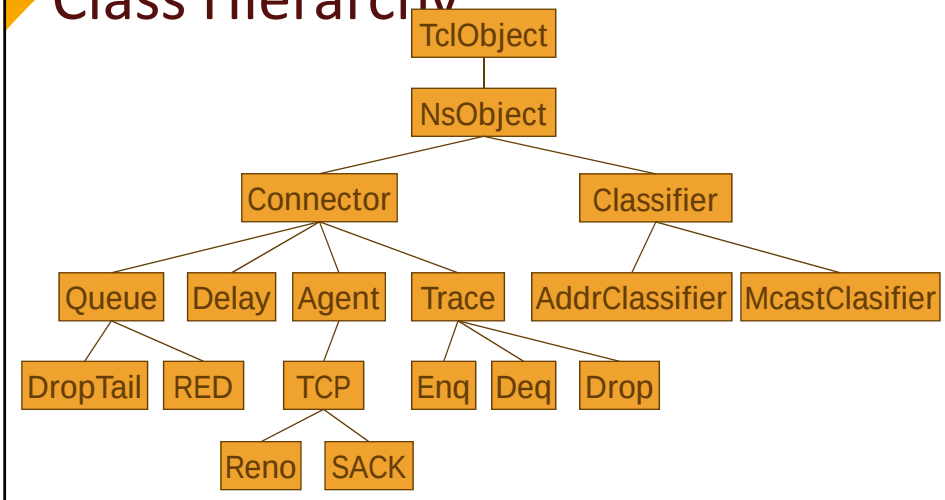
Guidelines

- Decide position in class hierarchy
  - I.e., which class to derive from?
- Create new packet header (if necessary)
- Create C++ class, fill in methods
- Define otcl linkage (if any)
- Write otcl code (if any)
- Build (and debug)

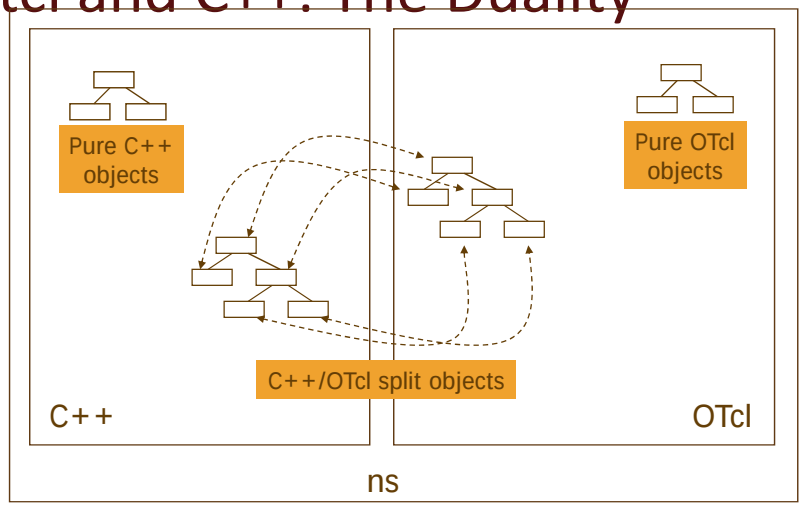
Important Basics

- class hierarchy
- otcl and C++ linkage

Class Hierarchy



otcl and C++: The Duality



C++/otcl Linkage

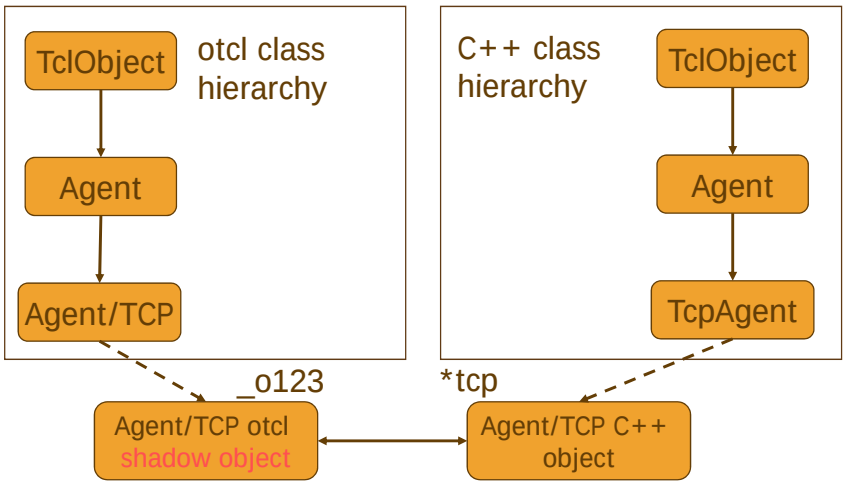
TclObject	Root of ns-2 object hierarchy
	bind(): link variable values between C++ and OTcl
	command(): link OTcl methods to C++ implementations
TclClass	Create and initialize TclObject's
Tcl	C++ methods to access Tcl interpreter
TclCommand	Standalone global commands
EmbeddedTcl	ns script initialization

TclObject

- Basic hierarchy in ns for split objects
- Mirrored in both C++ and otcl
- Example

```
set tcp [new Agent/TCP]
$tcp set window_ 200
$tcp advance 10
```


TclObject: Hierarchy and Shadowing



TclObject::bind()

- Link C++ member variables to otcl object variables
- C++

```
TcpAgent::TcpAgent() {
  bind("window_", &wnd_);
  … …
}
```

  - bind_time(), bind_bool(), bind_bw()
- otcl

```
set tcp [new Agent/TCP]
$tcp set window_ 200
```


Initialization of Bound Variables

- Initialization through otcl class variables

```
Agent/TCP set window_ 50
```

- Do all initialization of bound variables in tcl/lib/ns-default.tcl
  - Otherwise a warning will be issued when the shadow object is created

TclObject::command()

- Implement otcl methods in C++
- Trap point: otcl method cmd{}
- Send all arguments after cmd{} call to TclObject::command()

TclObject::command()

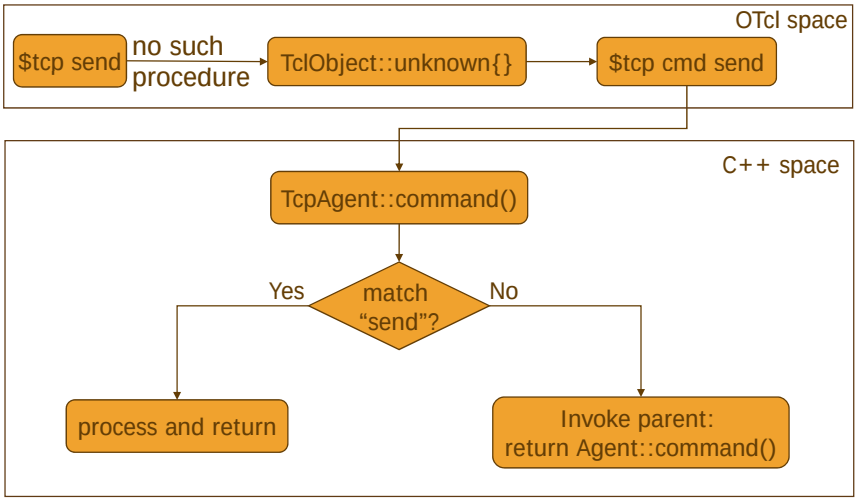
- otcl

```
set tcp [new Agent/TCP]
$tcp advance 10
```

- C++

```
int TcpAgent::command(int argc,
                      const char*const* argv) {
    if (argc == 3) {
      if (strcmp(argv[1], "advance") == 0) {
          int newseq = atoi(argv[2]);
              ……
          return(TCL_OK);
        }
    }
    return (Agent::command(argc, argv);
}
```

TclObject::command()

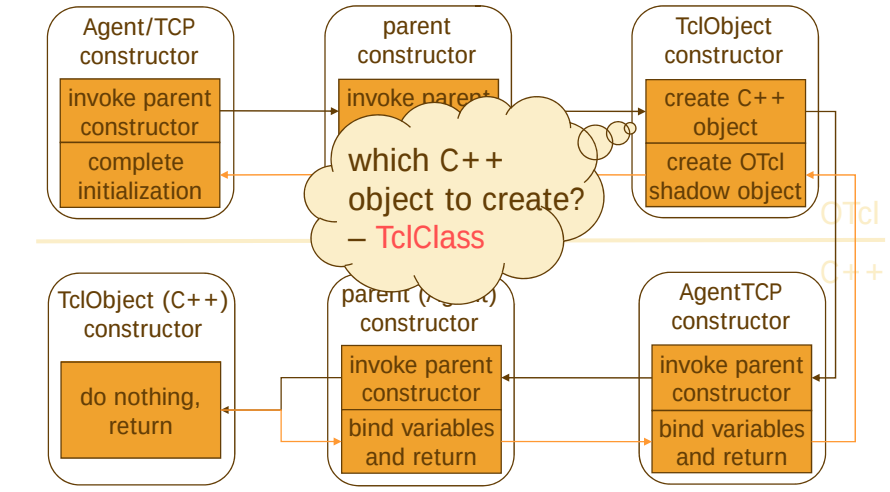


TclObject: Creation and Deletion

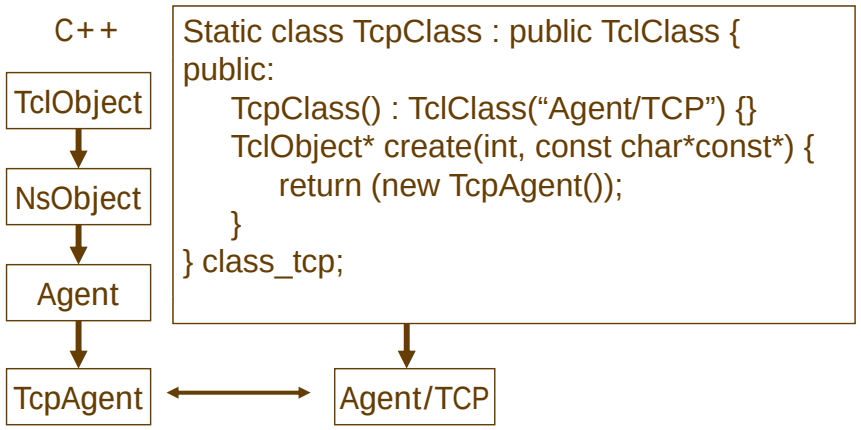
- Global procedures: new{}, delete{}
- Example

```
set tcp [new Agent/TCP]
…
delete $tcp
```

TclObject: Creation and Deletion



TclClass



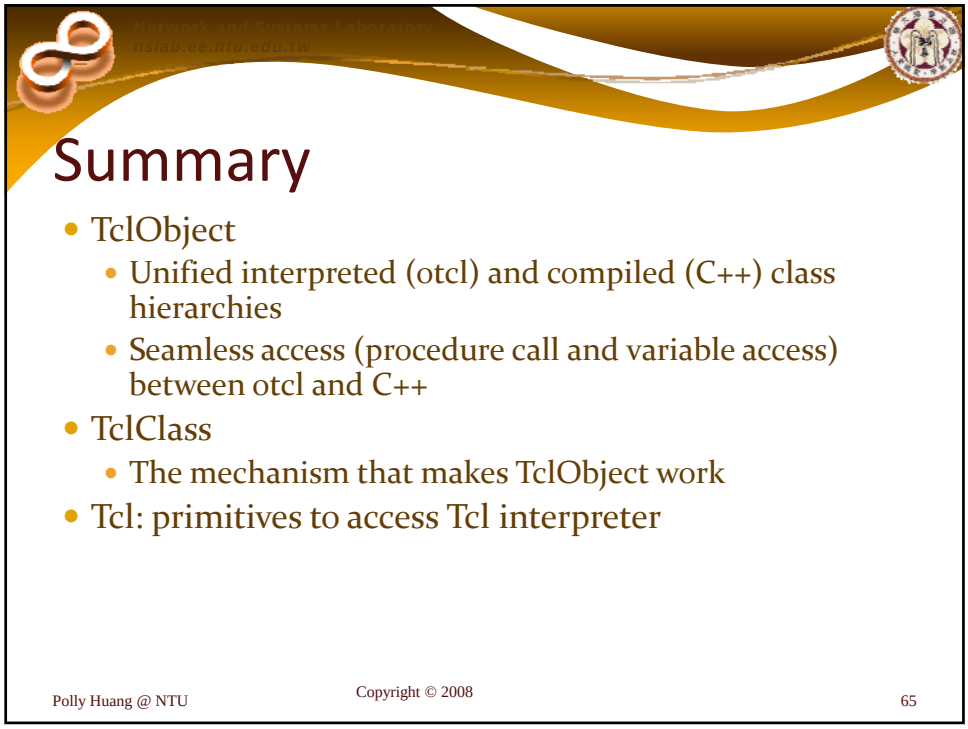

```
Static class TcpClass : public TclClass {
public:
    TcpClass() : TclClass("Agent/TCP") {}
    TclObject* create(int, const char*const*) {
        return (new TcpAgent());
    }
} class_tcp;
```

Class Tcl

- Singleton class with a handle to Tcl interpreter
- Usage
  - Pass a result string to otcl
  - Return success/failure code to otcl
  - Invoke otcl procedure
  - Obtain otcl evaluation results

Class Tcl

```
Tcl& tcl = Tcl::instance();
if (argc == 2) {
    if (strcmp(argv[1], "now") == 0) {
        tcl.resultf("%g", clock());
        return TCL_OK;
    }
    tcl.error("command not found");
    return TCL_ERROR;
 } else if (argc == 3) {
    tcl.eval(argv[2]);
    clock_ = atof(tcl.result());
    return TCL_OK;
}
```



Summary

- TclObject
  - Unified interpreted (otcl) and compiled (C++) class hierarchies
  - Seamless access (procedure call and variable access) between otcl and C++
- TclClass
  - The mechanism that makes TclObject work
- Tcl: primitives to access Tcl interpreter

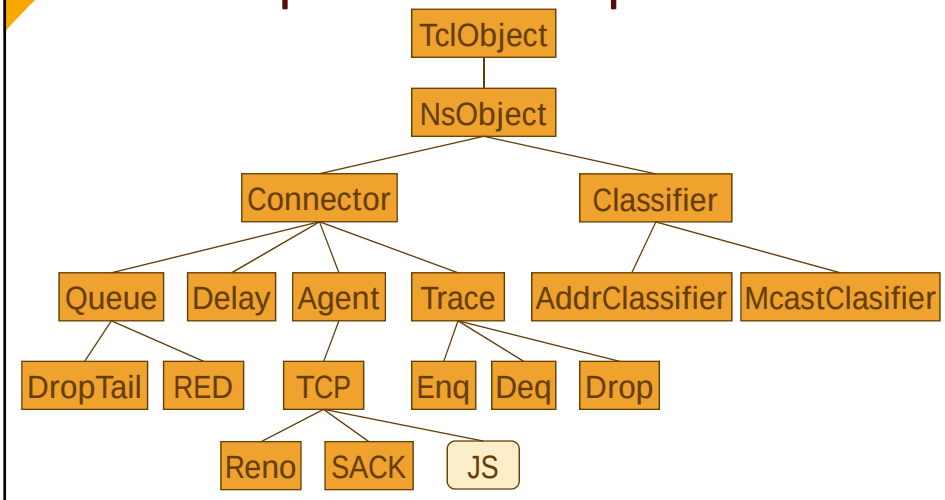
Creating New Components

- new agent, old packet headers
- new agent, new packet header

New Agent, Old Header

- TCP jump start
  - Wide-open transmission window at the beginning
  - From `cwnd_ += 1`
  - To `cwnd_ = MAXWIN_`

TCP Jump Start – Step 1



TCP Jump Start – Step 2

- New file: tcp-js.h

```
class JSTCPAgent : public TcpAgent {
public:
    virtual void set_initial_window() {
        cwnd_ = MAXWIN_;
    }
private:
    int MAXWIN_;
};
```


TCP Jump Start – Step 3

- New file: tcp-js.cc

```
static JSTcpClass : public TclClass {
public:
    JSTcpClass() : TclClass("Agent/TCP/JS") {}
    TclObject* create(int, const char*const*) {
        return (new JSTcpAgent());
    }
};
JSTcpAgent::JSTcpAgent() {
    bind("MAXWIN_", MAXWIN_);
}
```

New Agent, New Header

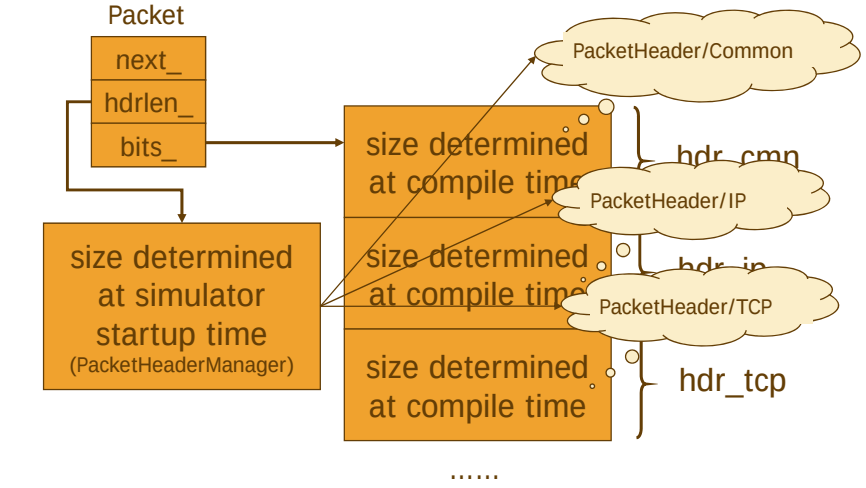
- Example: Agent/Message
  - New packet header for 64-byte message
  - New transport agent to process this new header

New Packet Header

- Create new header structure
- Enable tracing support of new header
- Create static class for otcl linkage (packet.h)
- Enable new header in otcl (tcl/lib/ns-packet.tcl)

- This does not apply when you add a new field into an existing header!

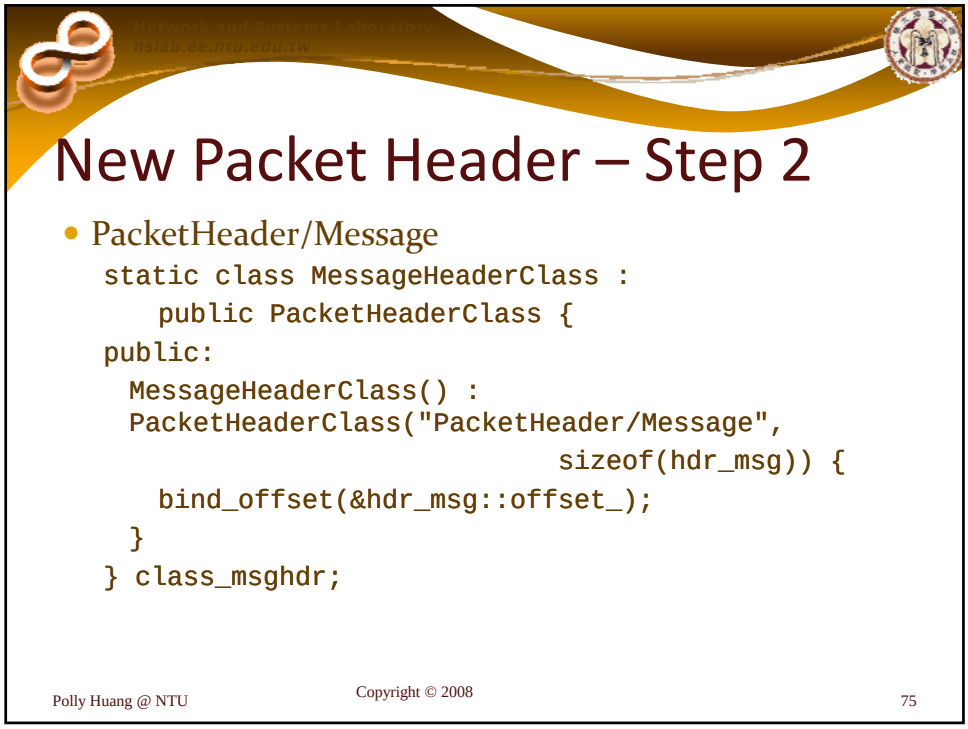
How Packet Header Works



New Packet Header – Step 1

- Create header structure

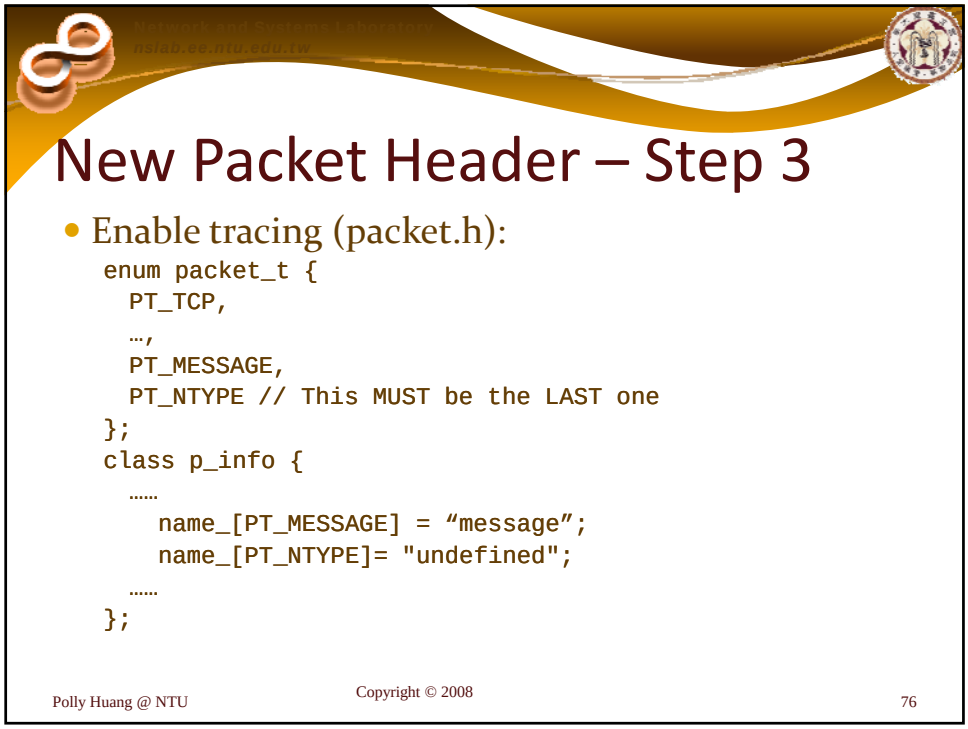
```
struct hdr_msg {
  char msg_[64];
  static int offset_;
  inline static int& offset() { return offset_; }
  inline static hdr_msg* access(Packet* p) {
    return (hdr_msg*) p->access(offset_);
  }
  /* per-field member functions */
  char* msg() { return (msg_); }
  int maxmsg() { return (sizeof(msg_)); }
};
```



New Packet Header – Step 2

- PacketHeader/Message

```
static class MessageHeaderClass :
    public PacketHeaderClass {
public:
  MessageHeaderClass() :
  PacketHeaderClass("PacketHeader/Message",
                                sizeof(hdr_msg)) {
    bind_offset(&hdr_msg::offset_);
  }
} class_msghdr;
```

New Packet Header – Step 3

- Enable tracing (packet.h):

```
enum packet_t {
  PT_TCP,
  …,
  PT_MESSAGE,
  PT_NTYPE // This MUST be the LAST one
};
class p_info {
  ……
    name_[PT_MESSAGE] = “message”;
    name_[PT_NTYPE]= "undefined";
  ……
};
```

New Packet Header – Step 4

- Register new header (tcl/lib/ns-packet.tcl)

```
foreach pair {
  { Common off_cmn_ }
  …
  { Message off_msg_ }
}
```

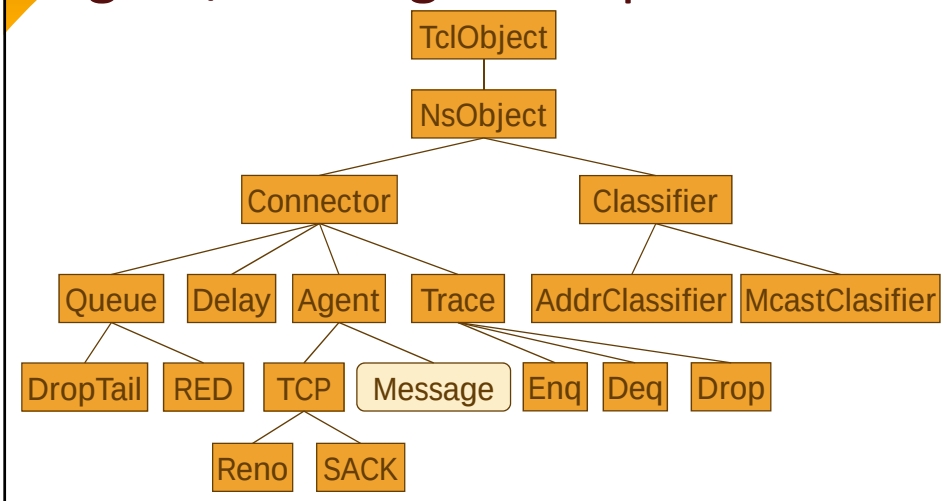

Packet Header: Caution

- Some old code, e.g.:

```
RtpAgent::RtpAgent() {
  ……
  bind("off_rtp_", &off_rtp);
}
……
hdr_rtp* rh = (hdr_rtp*)p->access(off_rtp_);
```

- Don't follow this example!

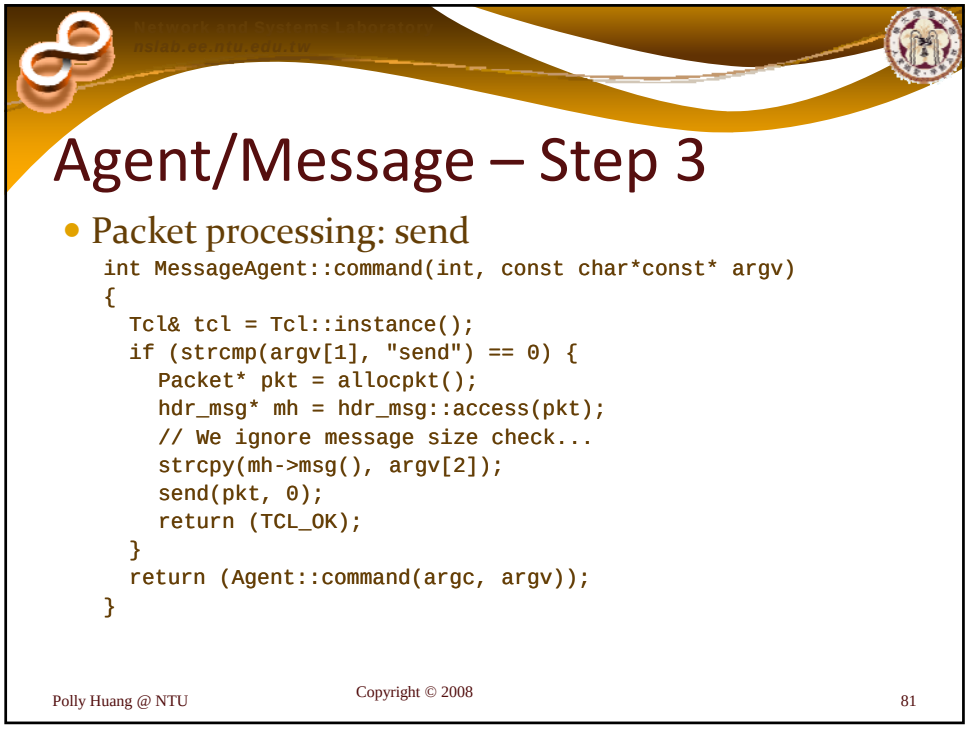
Agent/Message – Step 1



Agent/Message – Step 2

- C++ class definition

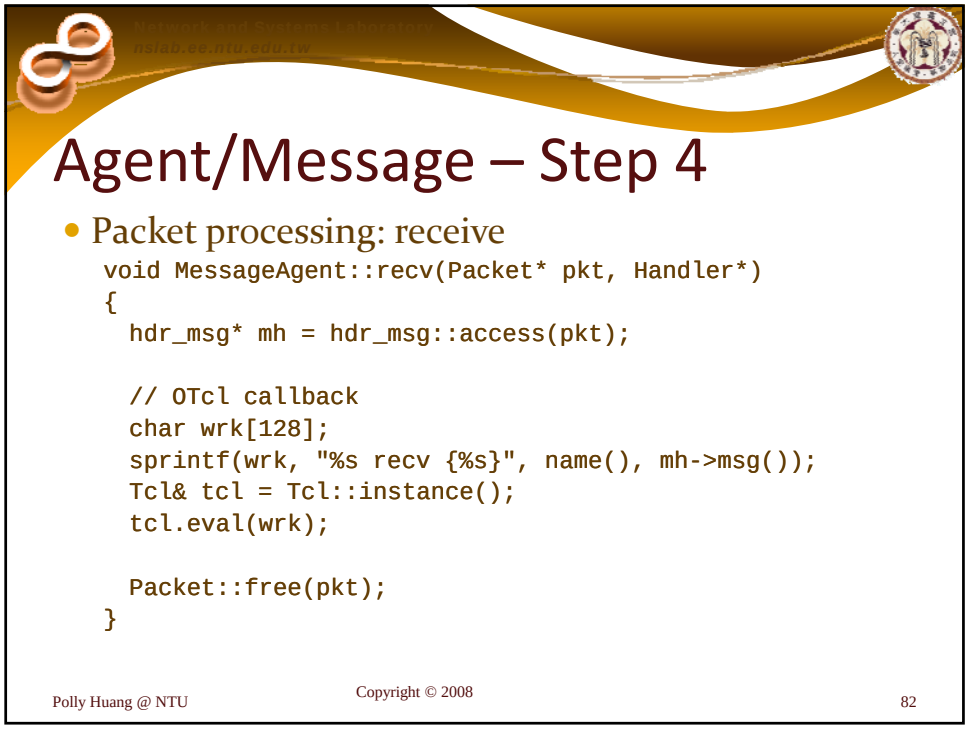
```
// Standard split object declaration
static …

class MessageAgent : public Agent {
public:
  MessageAgent() : Agent(PT_MESSAGE) {}
  virtual int command(int argc, const char*const*
  argv);
  virtual void recv(Packet*, Handler*);
};
```



Agent/Message – Step 3

- Packet processing: send

```
int MessageAgent::command(int, const char*const* argv)
{
  Tcl& tcl = Tcl::instance();
  if (strcmp(argv[1], "send") == 0) {
    Packet* pkt = allocpkt();
    hdr_msg* mh = hdr_msg::access(pkt);
    // We ignore message size check...
    strcpy(mh->msg(), argv[2]);
    send(pkt, 0);
    return (TCL_OK);
  }
  return (Agent::command(argc, argv));
}
```

Agent/Message – Step 4

- Packet processing: receive

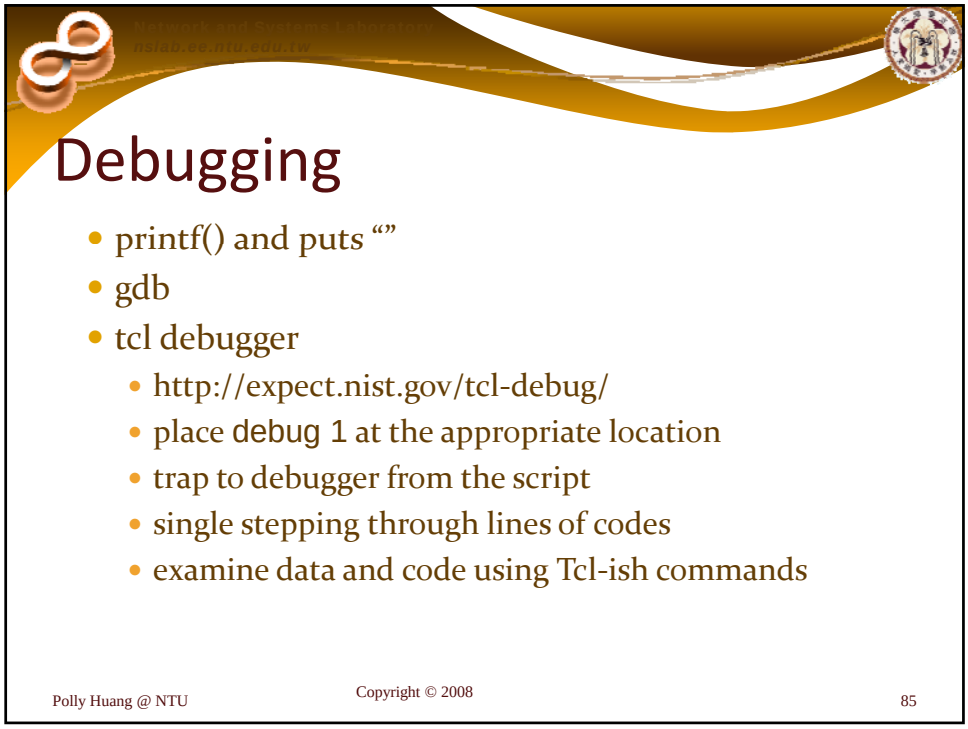
```
void MessageAgent::recv(Packet* pkt, Handler*)
{
  hdr_msg* mh = hdr_msg::access(pkt);

  // OTcl callback
  char wrk[128];
  sprintf(wrk, "%s recv {%s}", name(), mh->msg());
  Tcl& tcl = Tcl::instance();
  tcl.eval(wrk);

  Packet::free(pkt);
}
```

Outline for Today

- ns-2 Internal
- Making changes
- New components
  - in otcl
  - otcl and C++ Linkage
  - in C++
- **debugging**

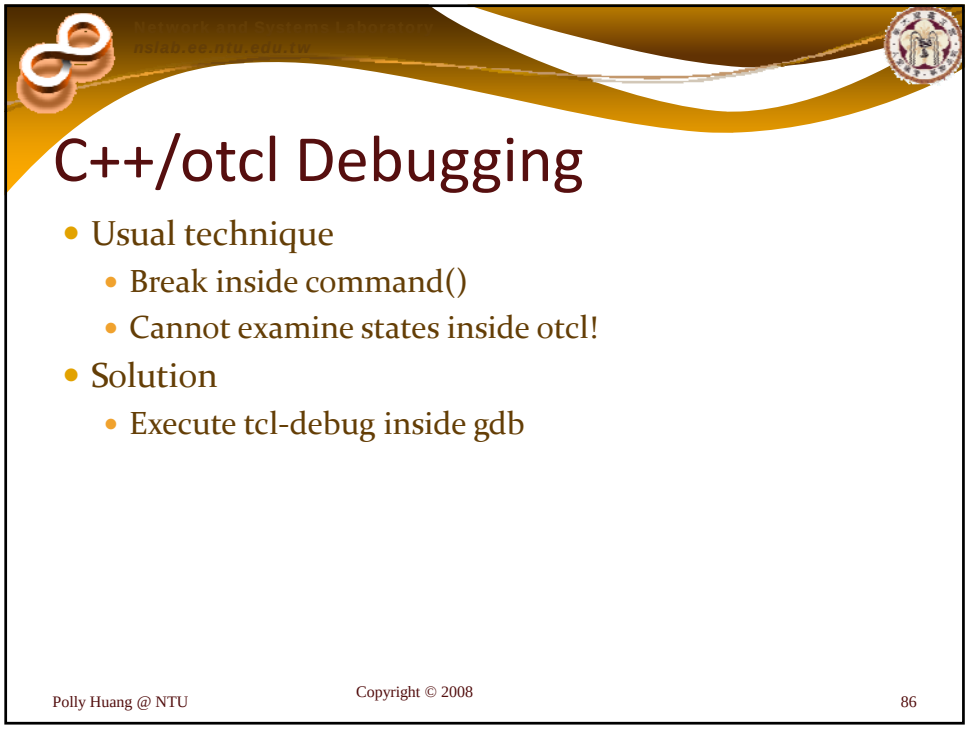
My ns dumps otcl scripts!

- Find the last 10-20 lines of the dump
- Is the error related to "_o4 cmd …" ?
  - Check your command()
- Otherwise, check the otcl script pointed by the error message



Debugging

- printf() and puts “”
- gdb
- tcl debugger
  - http://expect.nist.gov/tcl-debug/
  - place debug 1 at the appropriate location
  - trap to debugger from the script
  - single stepping through lines of codes
  - examine data and code using Tcl-ish commands



C++/otcl Debugging

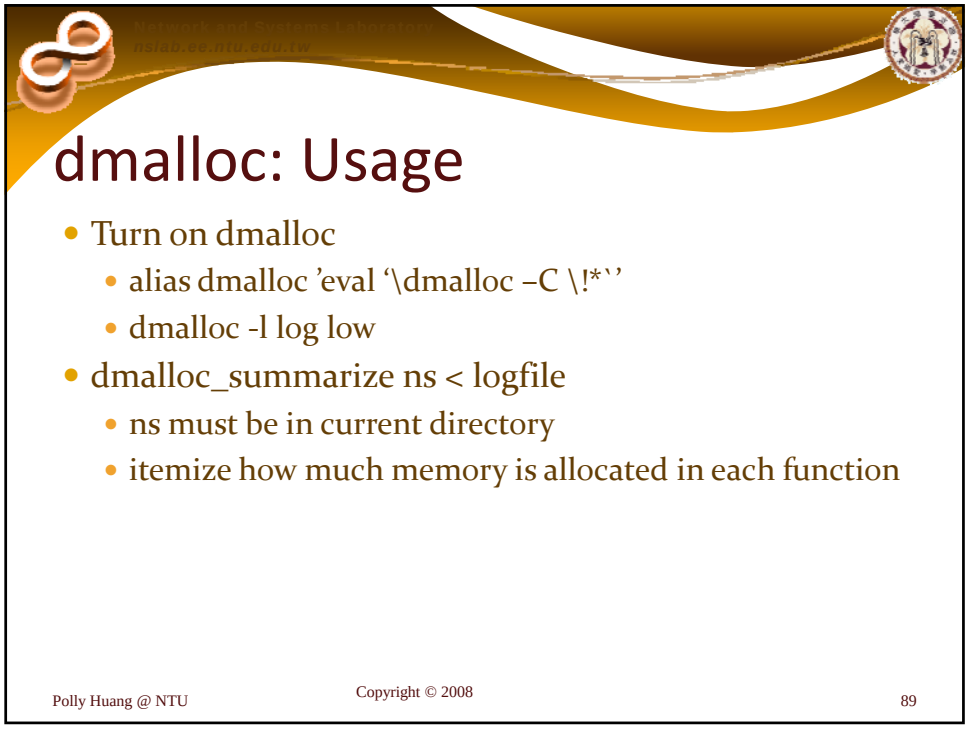
- Usual technique
  - Break inside command()
  - Cannot examine states inside otcl!
- Solution
  - Execute tcl-debug inside gdb

C++/otcl Debugging

```
(gdb) call Tcl::instance().eval("debug 1")
15: lappend auto_path $dbg_library
dbg15.3> w
*0: application
 15: lappend auto_path $dbg_library
dbg15.4> Simulator info instances
_o1
dbg15.5> _o1 now
0
dbg15.6> # and other fun stuff
dbg15.7> c
(gdb) where
#0 0x102218 in write()
......
```

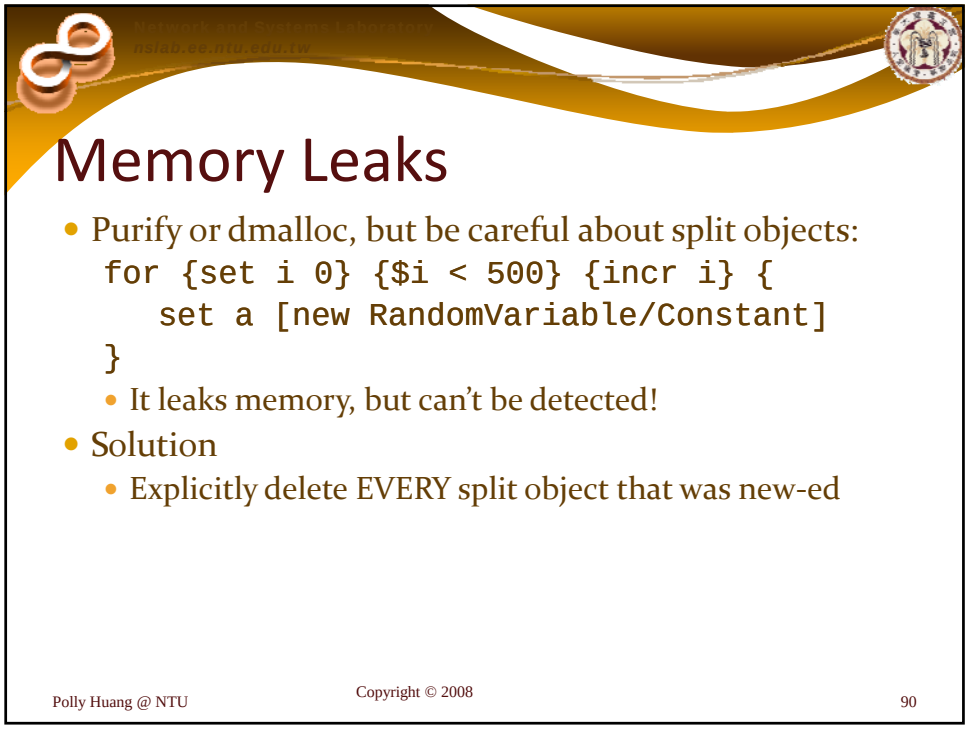

Memory Debugging in ns

- Purify
- Gray Watson's dmalloc library
  - http://www.dmalloc.com
  - make distclean
  - ./configure --with-dmalloc=<dmalloc_path>
  - Analyze results: dmalloc_summarize



dmalloc: Usage

- Turn on dmalloc
  - alias dmalloc 'eval '\dmalloc –C \!*`'
  - dmalloc -l log low
- dmalloc_summarize ns < logfile
  - ns must be in current directory
  - itemize how much memory is allocated in each function



Memory Leaks

- Purify or dmalloc, but be careful about split objects:

```
for {set i 0} {$i < 500} {incr i} {
   set a [new RandomVariable/Constant]
}
```

  - It leaks memory, but can't be detected!
- Solution
  - Explicitly delete EVERY split object that was new-ed

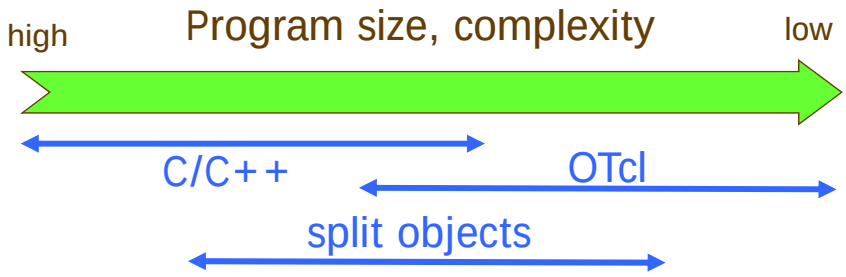
Memory Conservation Tips

- Avoid `trace-all`
- Use arrays for a sequence of variables
  - Instead of `n$i`, say `n($i)`
- Avoid OTcl temporary variables
- Use dynamic binding
  - `delay_bind()` instead of `bind()`
  - See object.{h,cc}

Scalability vs Flexibility

- It's tempting to write all-otcl simulation
  - Benefit: quick prototyping
  - Cost: memory + runtime
- Solution
  - Control the granularity of your split object by migrating methods from otcl to C++

The Merit of OTcl



- Smoothly adjust the granularity of scripting to balance extensibility and performance
- With complete compatibility with existing simulation scripts

Object Granularity Tips

- Functionality
  - Per-packet processing → C++
  - Hooks, frequently changing code → otcl
- Data management
  - Complex/large data structure → C++
  - One-time configuration variables → otcl


Network and Systems Laboratory
nslab.ee.ntu.edu.tw
Questions?
Polly Huang @ NTU
Copyright © 2008
95