NS2
How to implement a new protocol?

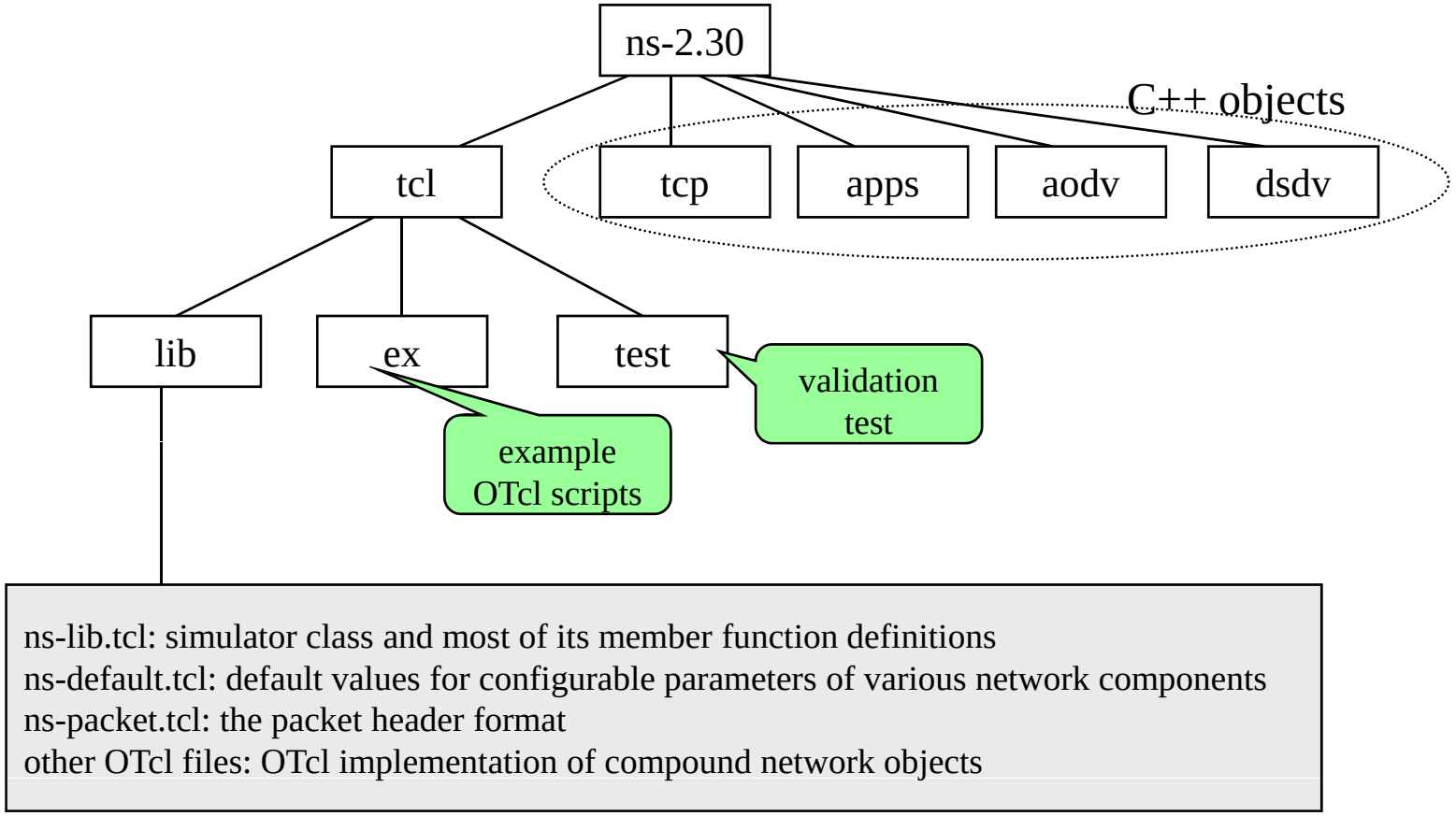
Chang-Gun Lee (cglee@snu.ac.kr)

Assistant Professor

The School of Computer Science and Engineering

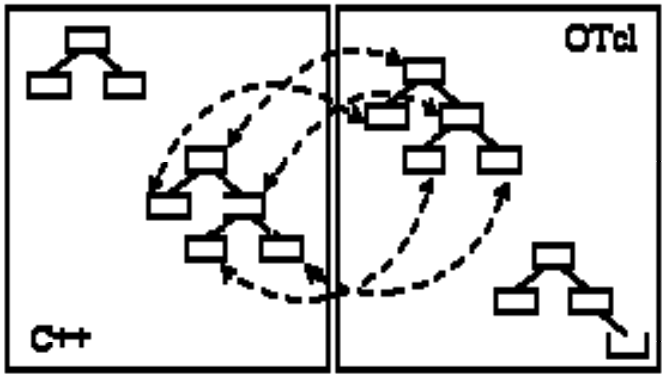
Seoul National University

NS directory structure



What to do to make a new object

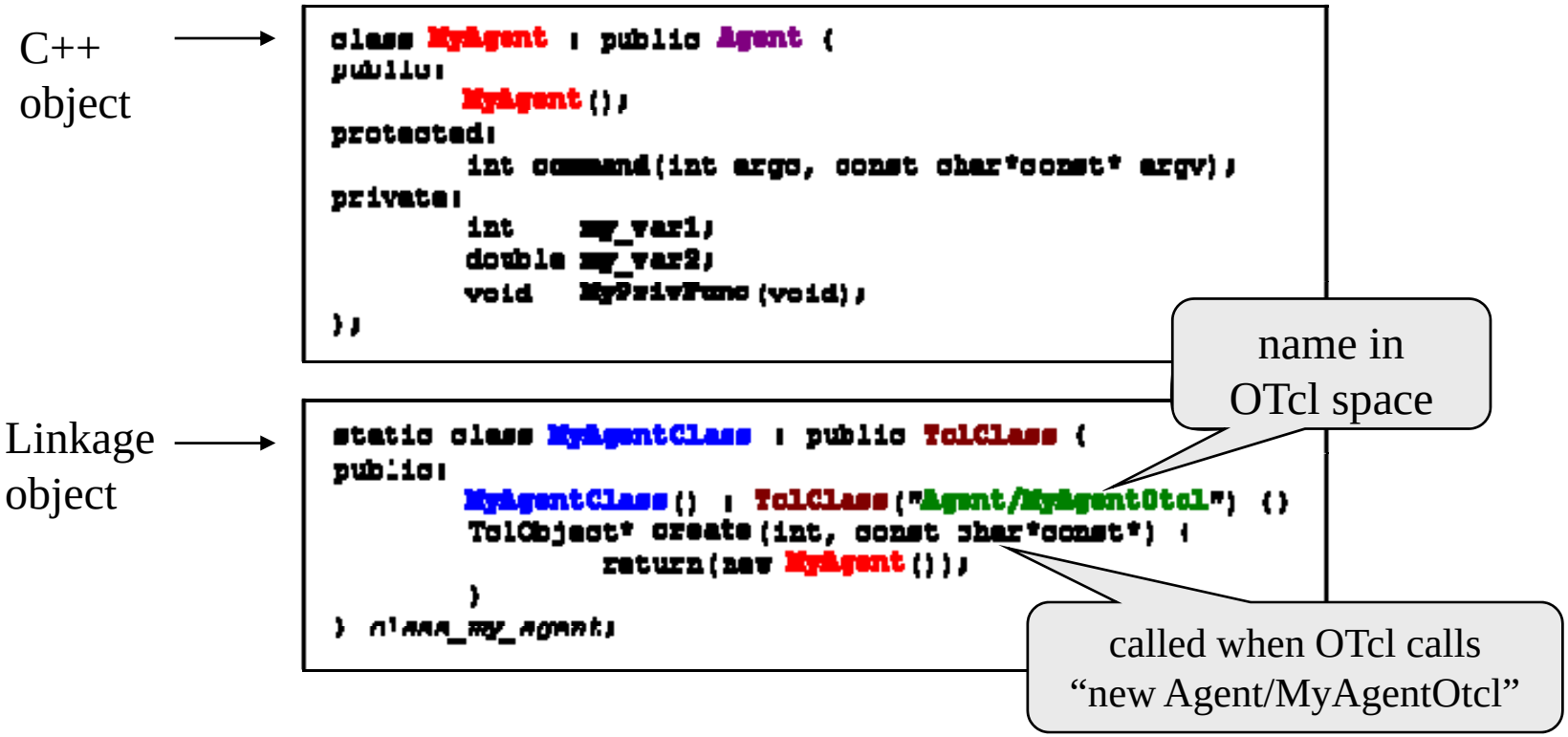
- Define a new object in C++ source
- Define its corresponding OTcl linkage object in C++ source
  - Why?
  - To make it possible to create an instance of this object in OTcl
  - To make it possible for OTcl to access C++ object variables and member functions



- Change Makefile so that it compiles the new C++ source
- Make
- Now you can create and use the new object in OTcl script

Let's make a simple dummy object MyAgent

- Define a new object in C++ source
- Define its corresponding OTcl linkage object in C++ source



How to make OTcl access C++?

- Export C++ object member variables to OTcl
  - Use “bind” in Object constructor function
  - Set default values for the exported variables in “ns-2.30/tcl/lib/ns-lib.tcl” (ns-default.tcl???)

```
MyAgent::MyAgent() : Agent(PT_UDP) {
        bind("my_var1_otcl", &my_var1);
        bind("my_var2_otcl", &my_var2);
}
```

name in OTcl space

Now, OTcl can
“$myagent set my_var2_otcl 3.14”

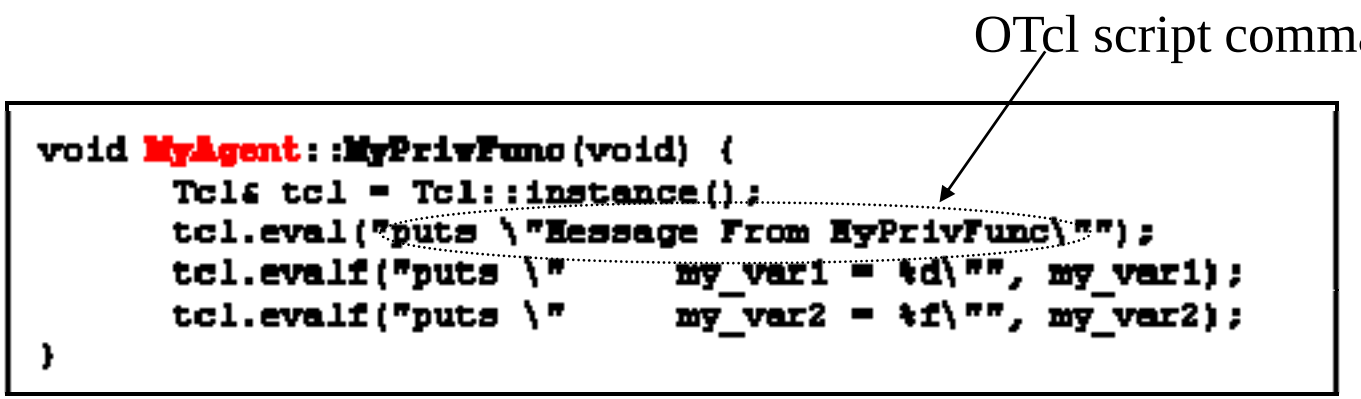
- Export C++ object member functions to OTcl
  - Define a “command” member function of your C++ object (MyAgent) that maps OTcl commands to C++ member functions

```
int MyAgent::command(int argc, const char*const* argv) {
      if(argc == 2) {
            if(strcmp(argv[1], "call-my-priv-func") == 0) {
                  MyPrivFunc();
                  return(TCL_OK);
            }
      }
      return(Agent::command(argc, argv));
}
```

Now, OTcl can
“$myagent call-my-priv-func”

How to execute an OTcl command from C++?

- get a reference to “Tcl::instance()”
- use “eval” (or something like that) to execute OTcl commands



Complete C++ code

```
#include <stdio.h>
#include <string.h>
#include "agent.h"

class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int    my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};


static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl")
{}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

```
MyAgent::MyAgent() : Agent(PT_UDP) {
       bind("my_var1_otcl", &my_var1);
       bind("my_var2_otcl", &my_var2);
}


int MyAgent::command(int argc, const char*const*
argv) {
      if(argc == 2) {
          if(strcmp(argv[1], "call-my-priv-func") == 0) {
                MyPrivFunc();
                return(TCL_OK);
          }
      }
      return(Agent::command(argc, argv));
}


void MyAgent::MyPrivFunc(void) {
      Tcl& tcl = Tcl::instance();
      tcl.eval("puts \"Message From MyPrivFunc\"");
      tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
      tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

Rebuild NS

- Open “Makefile”, add “yourObject.o” at the end of object file list
- Recompile NS using the “make” command
  - make depend
  - make
  - cp ns /usr/local/bin

OTcl script using the new object

ex-linkage.tcl

```
# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_otcl and my_var2_otcl)
set myagent [new Agent/MyAgentOtcl]


# Set configurable parameters of MyAgent
$myagent set my_var1_otcl 2
$myagent set my_var2_otcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

Run “ns ex-linkage.tcl”

result

```
warning: no class variable Agent/MyAgentOtcl::my_var1_otcl
        see tcl-object.tcl in tclcl for info about this warning.
warning: no class variable Agent/MyAgentOtcl::my_var2_otcl

Message from MyPrivFunc
     my_var1 = 2
     my_var2 = 3.140000
```

More Realistic Object (new protocol “MyPing”)

- An agent object that sends a ping packet, receives a reply from the peer, and calculate the round-trip delay.

Step 1: C++ programing

- Write myping.h and myping.cc files

```
struct hdr_myping {
  char ret;
  double send_time;

  // packet header access function
  static int offset_; // required by PacketHeaderManager
  inline static int& offset() {return offset_;}
  inline static hdr_myping* access(const Packet* p){
                return (hdr_myping*) p->access(offset_);
  }
};

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

new proto packet header structure

necessary for packet header access

protocol agent object in C++ space

```
int hdr_myping::offset_;
static class MyPingHeaderClass : public PacketHeaderClass {
public:
  MyPingHeaderClass() :
                PacketHeaderClass("PacketHeader/Ping",
                                  sizeof(hdr_myping)) {
                bind_offset(&hdr_myping::offset_);
  }
} class_mypinghdr;

static class MyPingClass : public TclClass {
public:
  MyPingClass() : TclClass("Agent/MyPing") {}
  TclObject* create(int, const char*const*) {
    return (new MyPingAgent());
  }
} class_myping;
```

instantiation of OTcl linkage object for packet header structure

instantiation of OTcl linkage object for the agent object

```
MyPingAgent::MyPingAgent() : Agent(PT_MyPing)
{
  bind("packetSize_", &size_);
}

int MyPingAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the Ping header for the new packet:
      hdr_myping* hdr = hdr_myping::access(pkt);
      // Set the 'ret' field to 0, so the receiving node knows
      // that it has to generate an echo packet
      hdr->ret = 0;
      // Store the current time in the 'send_time' field
      hdr->send_time = Scheduler::instance().clock();
      // Send the packet
      send(pkt, 0);
      // return TCL_OK, so the calling function knows that the
      // command has been processed
      return (TCL_OK);
    }
  }
  // If the command hasn't been processed by
PingAgent()::command,
  // call the command() function for the base class
  return (Agent::command(argc, argv));
}
```

export C++ object variable to OTcl

export the C++ object control to OTcl

```
void MyPingAgent::recv(Packet* pkt, Handler*)
{
  // Access the IP header for the received packet:
  hdr_ip* hdrip = hdr_ip::access(pkt);
  // Access the Ping header for the received packet:
  hdr_myping* hdr = hdr_myping::access(pkt);
  // Is the 'ret' field = 0 (i.e. the receiving node is be
  if (hdr->ret == 0) {
    // Send an 'echo'. First save the old packet's send_time
    double stime = hdr->send_time;
    // Discard the packet
    Packet::free(pkt);
    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the Ping header for the new packet:
    hdr_myping* hdrret = hdr_myping::access(pktret);
    // Set the 'ret' field to 1, so the receiver won't send another echo
    hdrret->ret = 1;
    // Set the send_time field to the correct value
    hdrret->send_time = stime;
    // Send the packet
    send(pktret, 0);
  } else {
    // A packet was received. Use tcl.eval to call the Tcl
    // interpreter with the ping results.
    // Note: In the Tcl code, a
    // has to be defined which allow
    // result.
    char out[100];
    // Prepare the output to the Tcl inte
    // trip time
    sprintf(out, "%s recv %d %3.1f", name(),
            hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
                    (Scheduler::instance().clock() - hdr->send_time) * 1000);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    // Discard the packet
    Packet::free(pkt);
  }
}
```

"recv" member: packet handler

access packet header

receive message with ret=0: "reply"

receive message with ret=1: "calculate and print RoundTrip Time"

OTcl script command

Step 2: Register the new protocol

- Modify packet.h, ns-packet.tcl, ns-default.tcl files of NS

packet.h

```
enum packet_t {
        PT_TCP,
        PT_UDP,
        …
        PT_MyPing,
        PT_NTYPE // this must be the last one
};

class p_info {
public:
        p_info(){
            name_[PT_TCP] = “tcp”;
            name_[PT_UDP] = “udp”;
            …
            name_[PT_MyPing] = “MyPing”;
            name_[PT_NTYPE] =“undefined”;
        }
        …
};
```

ns-packet.tcl

```
foreach prot {
        AODV
        …
        MyPing
} {
        add-packet-header $prot
}
```

ns-default.tcl

```
…
…
Agent/MyPing set packetSize_ 64
…
```

set default vaules of the protocol parameters

Step 3: Rebuild NS

- Modify Makefile
  - add MyPing.o to the object file list
- Renew dependency
  - make depend
- Re-Make NS
  - make
- Install NS
  - cp ns /usr/local/bin

Step 4: Writing and running OTcl script

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

#Connect the nodes with two links
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail

#Define a 'recv' function for the class 'Agent/MyPing'
Agent/MyPing instproc recv {from rtt} {
        $self instvar node_
        puts "node [$node_ id] received ping answer
from \
        $from with round-trip-time $rtt ms."
}

#Create two ping agents and attach them to the nodes n0
and n2
set p0 [new Agent/MyPing]
$ns attach-agent $n0 $p0

set p1 [new Agent/MyPing]
$ns attach-agent $n2 $p1
```

```
#Connect the two agents
$ns connect $p0 $p1

#Schedule events
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.6 "$p0 send"
$ns at 0.6 "$p1 send"
$ns at 1.0 "finish"

#Run the simulation
$ns run
```

called by C++ when an echo is received

Homework 8

- Make a newPing
  - once started, it sends probing packets 5 times periodically at every 1 sec.
- Using the same network of Homework 7, measure the round-trip delays of each way between node 1 and 5
  - first newPing start at 1.0 sec
  - second newPing start at 11.0 sec