Introduction to Wireless and Mobile Networking

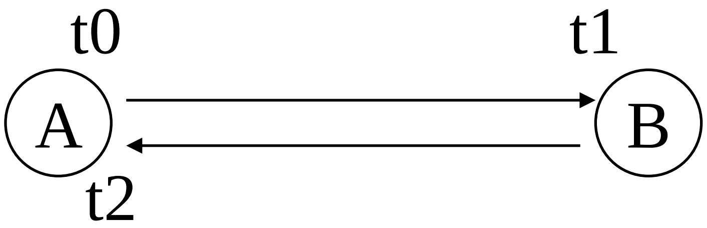
NS-2 Tutorial-4

Hung-Yu Wei
National Taiwan University

Speaker: Chih-Yu Wang

Creating A New Protocol

- NS-2 tutorial: Section VII
  - http://www.isi.edu/nsnam/ns/tutorial/index.html
- Actually, you should go through the whole tutorial
- We start from a simple protocol: ping



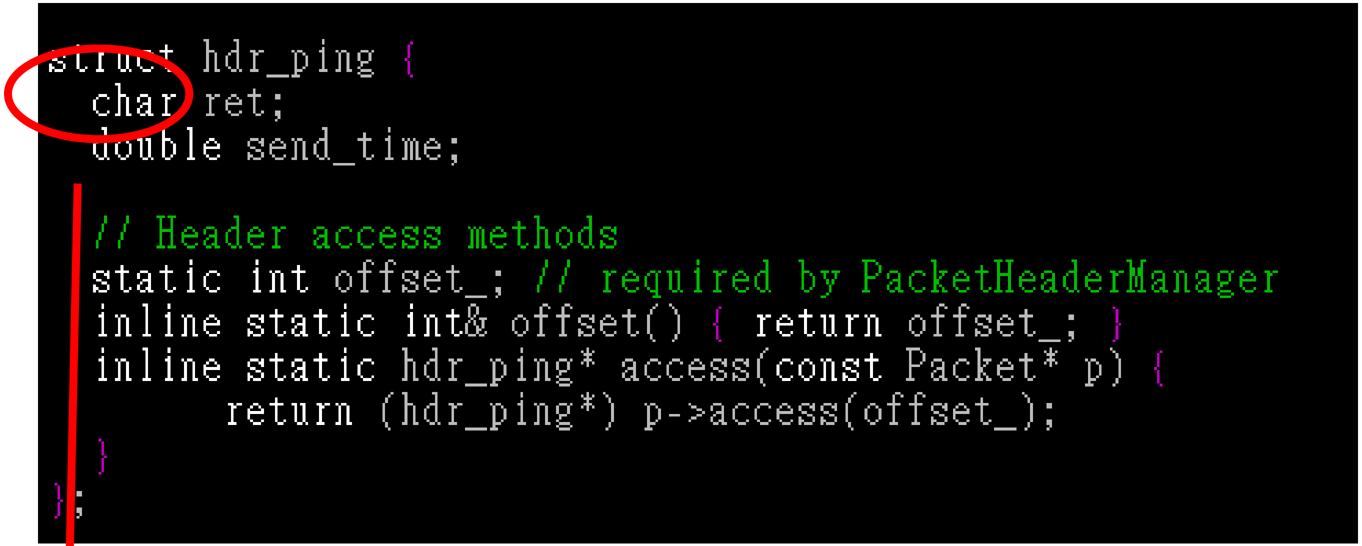
Ping computes (t2-t0)

What we should implement

- The structure of Ping Packet
  - Send_time: the time this PING transmit
  - RET: how many times this PING delivered
- The Ping protocol
  - send (called from script)
  - recv (triggered by NS-2)
- The parameters
  - Packet size
  - Header offset

Ping.h

Header of Ping Packets

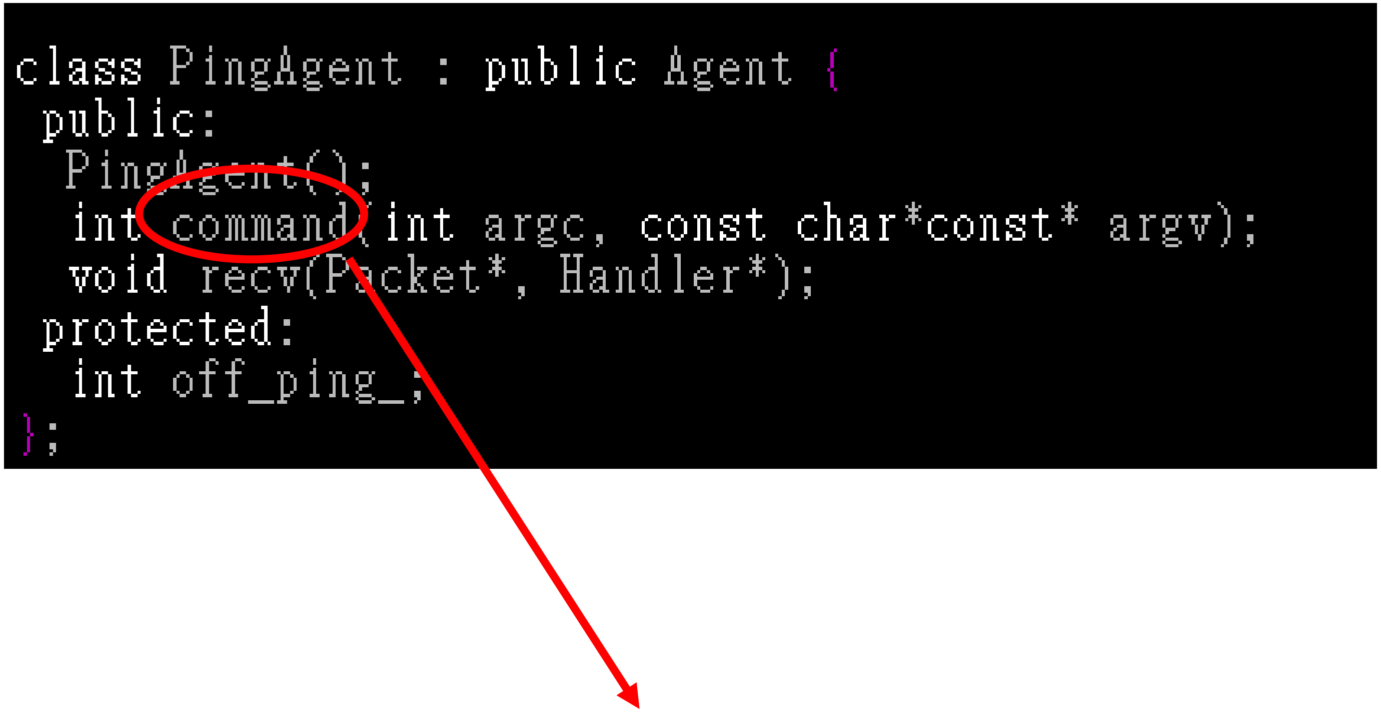


```
struct hdr_ping {
  char ret;
  double send_time;

  // Header access methods
  static int offset_; // required by PacketHeaderManager
  inline static int& offset() { return offset_; }
  inline static hdr_ping* access(const Packet* p) {
        return (hdr_ping*) p->access(offset_);
  }
};
```

char?

Ping.h

- Ping Agent
  - C++ definition

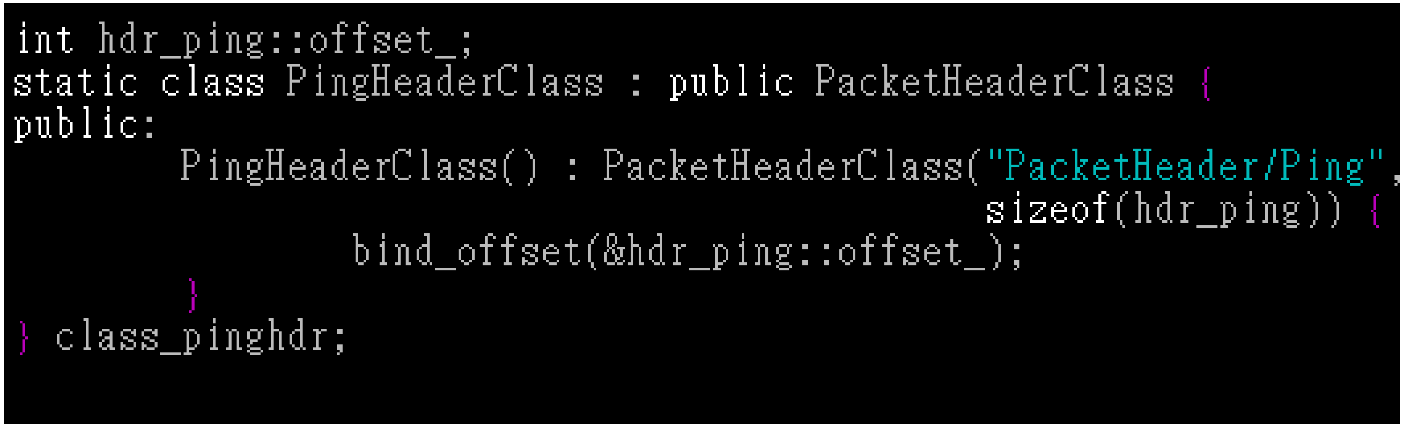


```
class PingAgent : public Agent {
 public:
  PingAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 protected:
  int off_ping_;
};
```

TCL command from ns-2

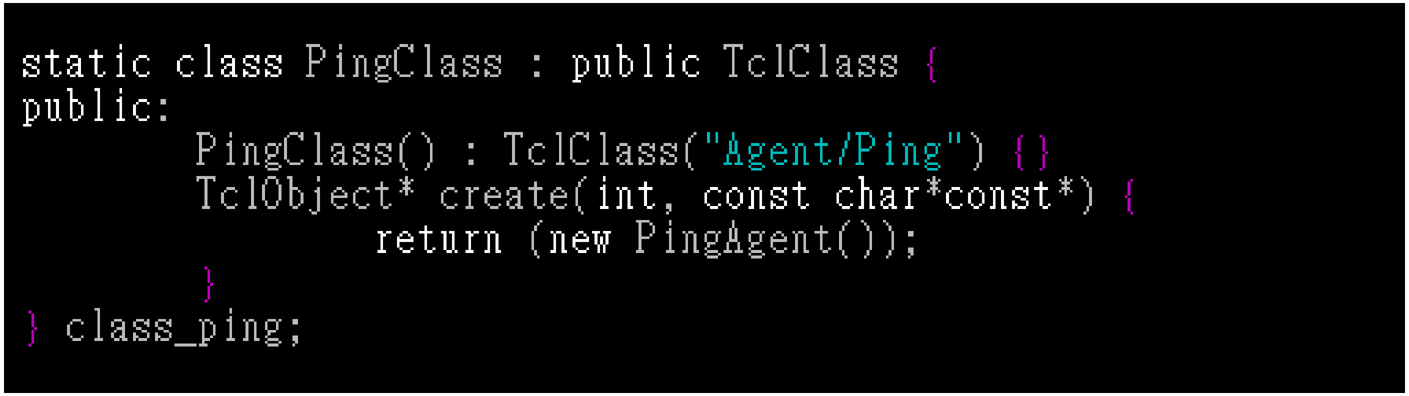
Ping.cc

[Class]Ping Packet Header



```
int hdr_ping::offset_;
static class PingHeaderClass : public PacketHeaderClass {
public:
        PingHeaderClass() : PacketHeaderClass("PacketHeader/Ping",
                                              sizeof(hdr_ping)) {
                bind_offset(&hdr_ping::offset_);
        }
} class_pinghdr;
```

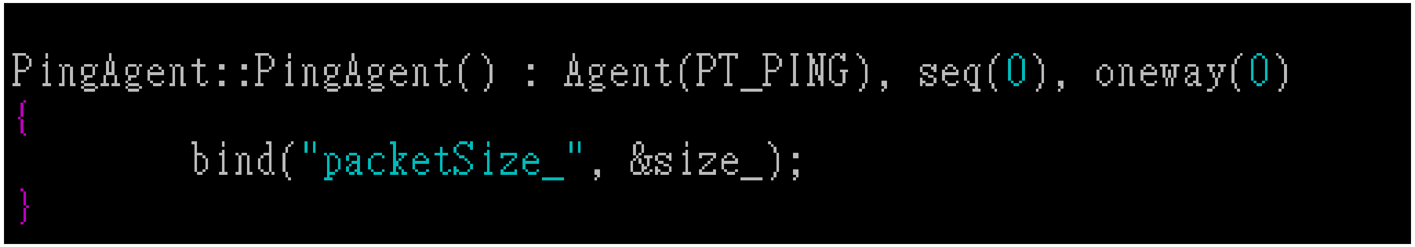
[Class]Ping Agent



```
static class PingClass : public TclClass {
public:
        PingClass() : TclClass("Agent/Ping") {}
        TclObject* create(int, const char*const*) {
                return (new PingAgent());
        }
} class_ping;
```

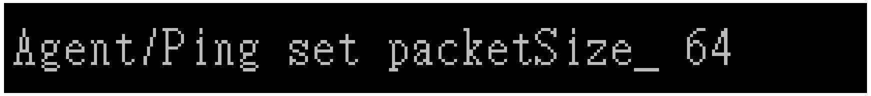
Binding the C++ and OTcl objects/variables

- Ping.cc



```
PingAgent::PingAgent() : Agent(PT_PING), seq(0), oneway(0)
{
        bind("packetSize_", &size_);
}
```

- tcl/lib/ns-default.tcl(or your tcl script)



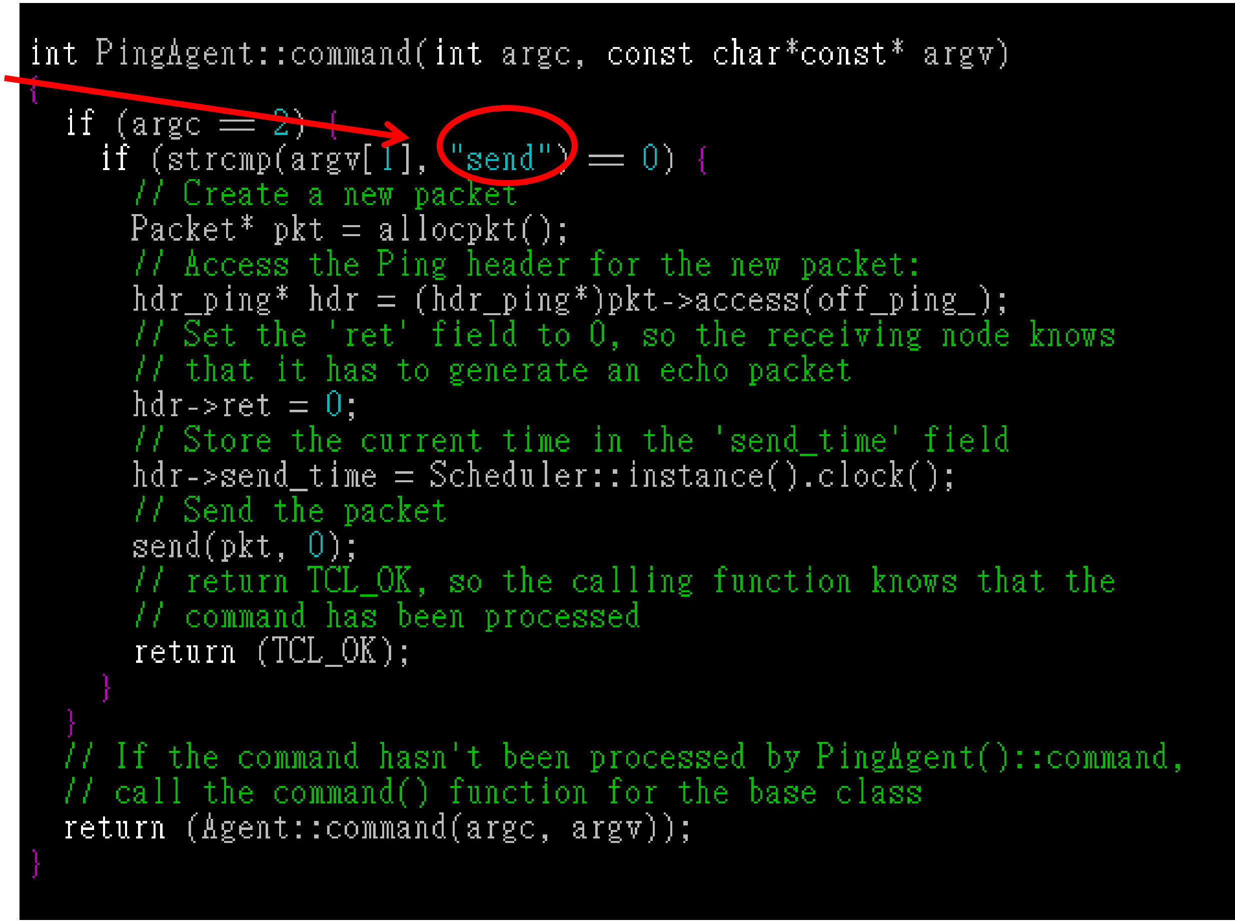
```
Agent/Ping set packetSize_ 64
```

Command Methods: sending packet

In Tcl:

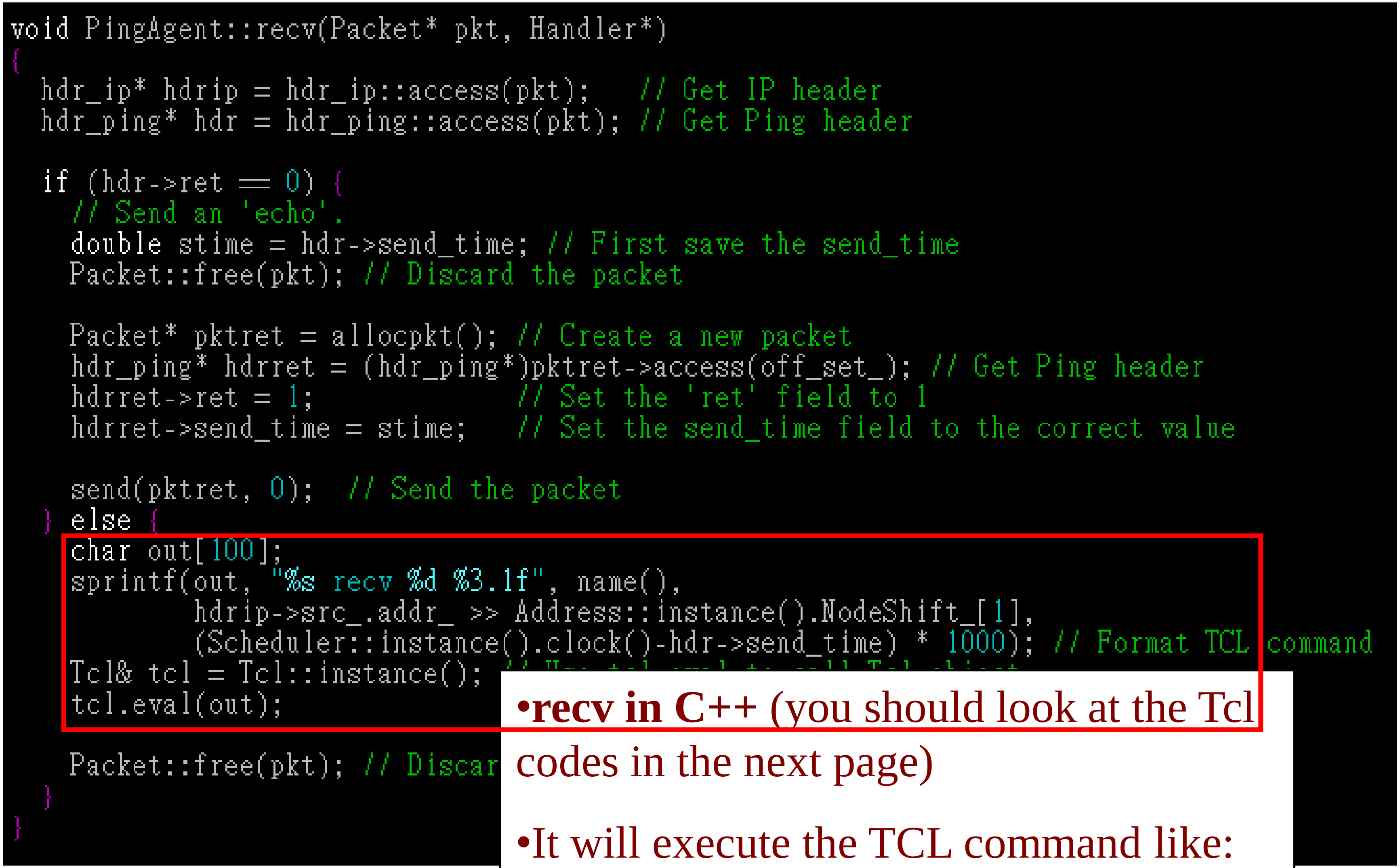
$ns at 0.2 "$p0 send”
$ns at 0.4 "$p1 send”
$ns at 0.6 "$p0 send”
$ns at 0.6 "$p1 send”

- Reference:
  NS-2 manual
  Section 3.4.4



```
int PingAgent::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_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
      // 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));
}
```

Receiving Packets (ping.cc)



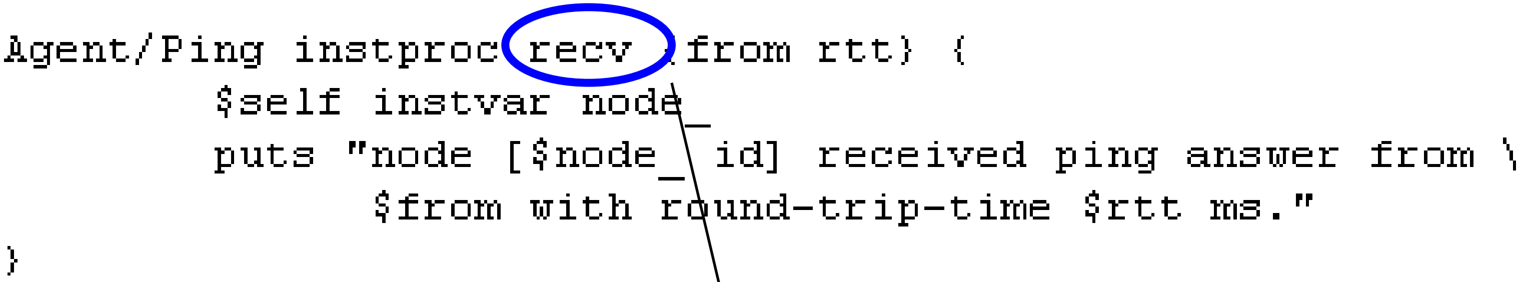
```
void PingAgent::recv(Packet* pkt, Handler*)
{
  hdr_ip* hdrip = hdr_ip::access(pkt);   // Get IP header
  hdr_ping* hdr = hdr_ping::access(pkt); // Get Ping header

  if (hdr->ret == 0) {
    // Send an 'echo'.
    double stime = hdr->send_time; // First save the send_time
    Packet::free(pkt); // Discard the packet

    Packet* pktret = allocpkt(); // Create a new packet
    hdr_ping* hdrret = (hdr_ping*)pktret->access(off_set_); // Get Ping header
    hdrret->ret = 1;                // Set the 'ret' field to 1
    hdrret->send_time = stime;      // Set the send_time field to the correct value

    send(pktret, 0);  // Send the packet
  } else {
    char out[100];
    sprintf(out, "%s recv %d %3.1f", name(),
            hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
            (Scheduler::instance().clock()-hdr->send_time) * 1000); // Format TCL command
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);

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

- **recv in C++** (you should look at the Tcl codes in the next page)
- It will execute the TCL command like:
- node_(0) recv node_(1) 5.00

tcl: Agent/Ping class

- The tutorial put the simulation script and tcl function in the same file
  - Usually, they are different files
- instproc
  - Function in Tcl

* **recv in Tcl** (you should look at the C++ codes in the previous page)



```
Agent/Ping instproc recv {from rtt} {
        $self instvar node_
        puts "node [$node_ id] received ping answer from \
              $from with round-trip-time $rtt ms."
}
```

In C++ codes:

```
void recv(Packet*, Handler*){
        sprintf(out ,"%s recv %d %3.1f"…)
};
```

OTcl Linkage

- Invoking Tcl object
  - Tcl& tcl = Tcl::instance();
  - tcl.evalc(char *)
  - tcl.eval(const char *)
  - tcl.evalf("%d %f…",int,double,…)
- Passing results
  - tcl.result(const char *)
  - tcl.resultf("%d %f…",int,double,…)
- Error handling
  - tcl.error()

tcl: simulation

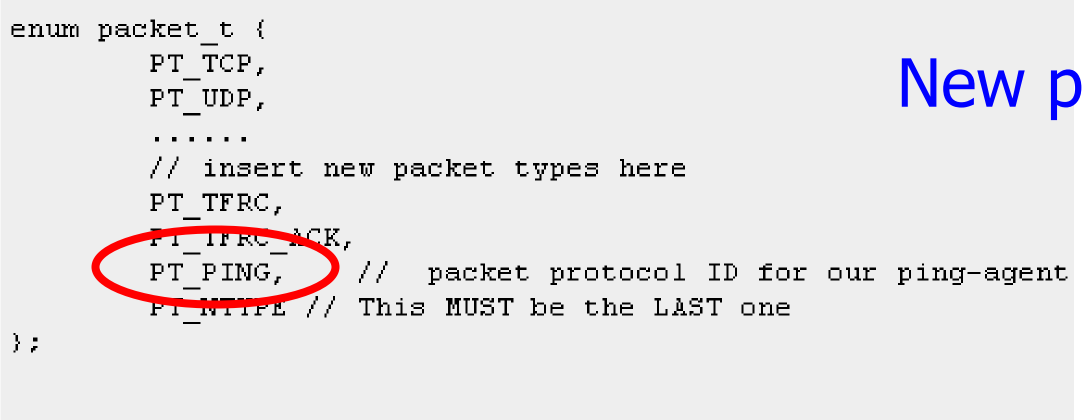
- Simulation Script

```
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
set p1 [new Agent/Ping]
$ns attach-agent $n2 $p1
$ns connect $p0 $p1
$ns at 0.2 "$p0 send”
$ns at 0.4 “$p1 send”
```

Other Modifications

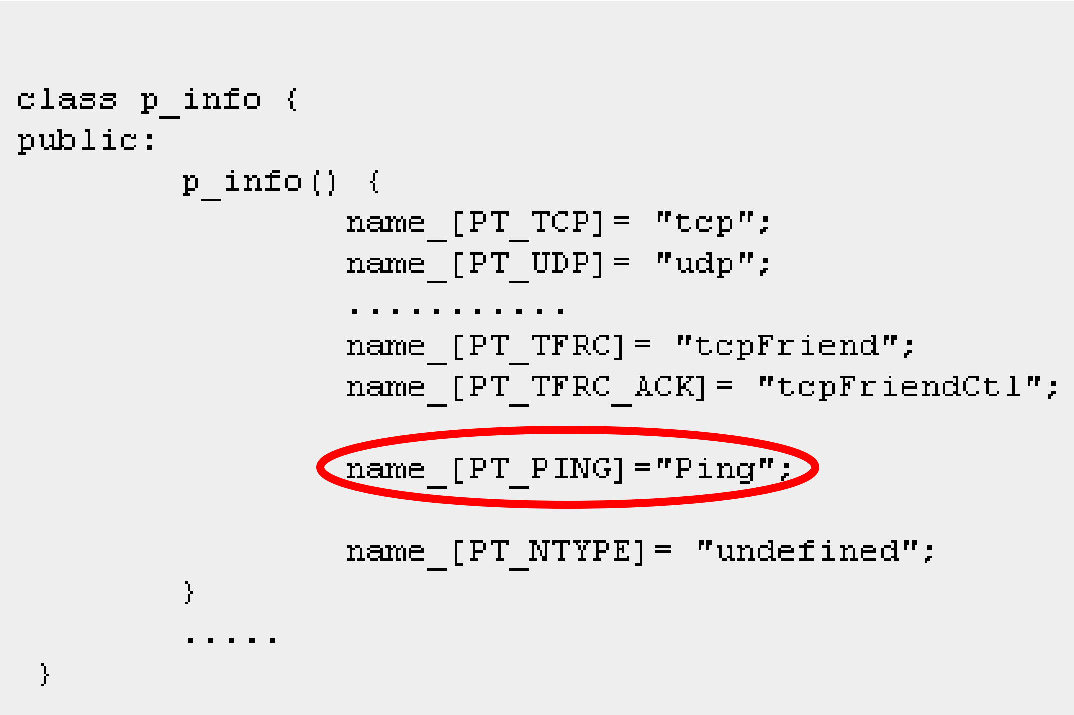
- Makefile
  - You should learn how to use ***make*** for Unix/Linux programming
- common/packet.h
  - Add new packet type
- tcl/lib/ns-packet.tcl
  - Packet header option
- tcl/lib/ns-default.tcl
  - Default Tcl values

/common/packet.h



```
enum packet_t {
        PT_TCP,
        PT_UDP,
        ......
        // insert new packet types here
        PT_TFRC,
        PT_TFRC_ACK,
        PT_PING,    //  packet protocol ID for our ping-agent
        PT_NTYPE // This MUST be the LAST one
};
```

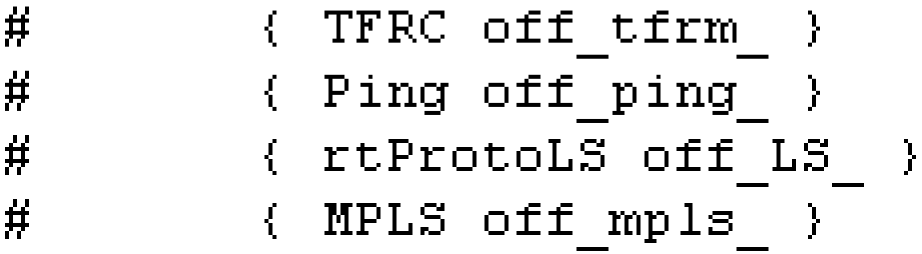
New packet type



```
class p_info {
public:
        p_info() {
                name_[PT_TCP]= "tcp";
                name_[PT_UDP]= "udp";
                ...........
                name_[PT_TFRC]= "tcpFriend";
                name_[PT_TFRC_ACK]= "tcpFriendCtl";

                name_[PT_PING]="Ping";

                name_[PT_NTYPE]= "undefined";
        }
        .....
 }
```

tcl/lib/ns-packet.tcl

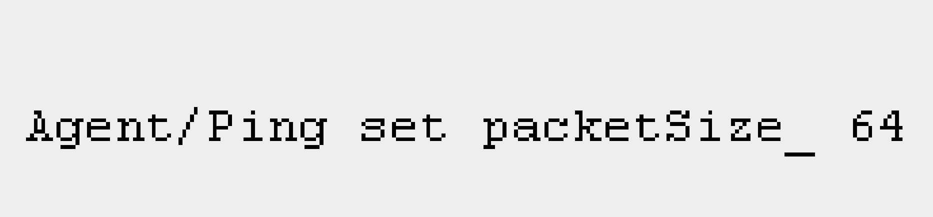
To save some memory, you can disable unneeded packet headers



```
#         { TFRC off_tfrm_ }
#         { Ping off_ping_ }
#         { rtProtoLS off_LS_ }
#         { MPLS off_mpls_ }
```

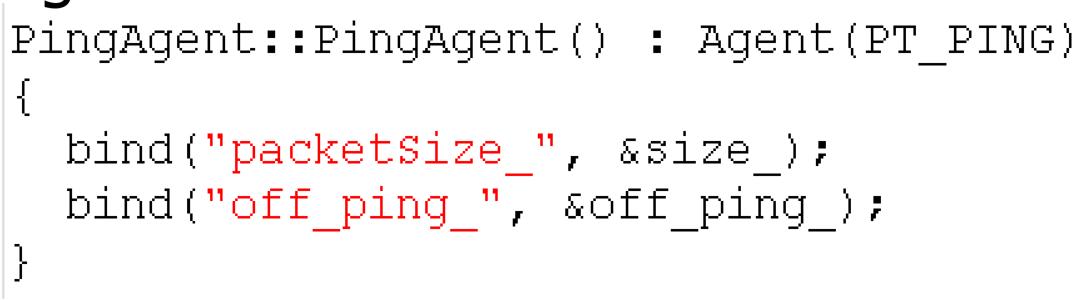
tcl/lib/ns-default.tcl

- Define all the Tcl default values



```
Agent/Ping set packetSize_ 64
```

In ping.cc



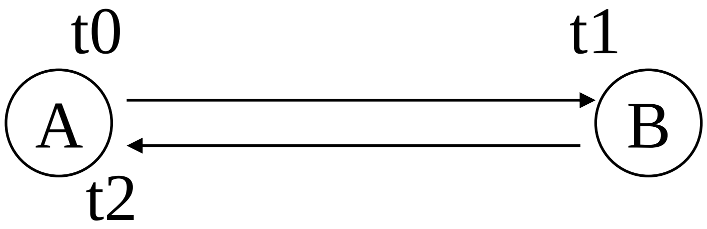
```
PingAgent::PingAgent() : Agent(PT_PING)
{
  bind("packetSize_", &size_);
  bind("off_ping_", &off_ping_);
}
```

Source Codes

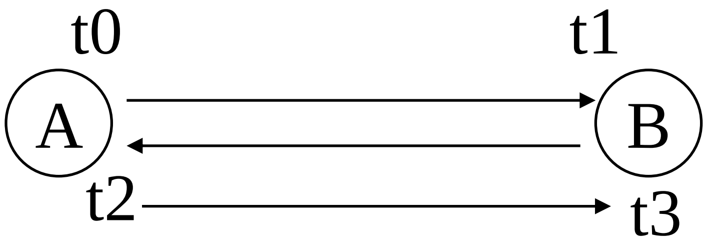
- Ping in NS-2
  - ns-allinone-2.*\ns-2.*\apps\ping.*
  - A complete version ping
- Ping in Tutorial
  - A simplified ping version for teaching purpose
- You could learn from both

Homework #3

- Coming after homework #2 is due
- The most difficult and important one
  - Design a Pong protocol
  - 3-way ping protocol



Ping computes (t2-t0)



Pong computes (t3-t0)

18

Some tips for simulation

- What will experts/researchers do?
  - 20% in implementation, 30% in simulation, 50% in analysis and report
- What will beginners/students do?
  - 90% in implementation, 9% in simulation & analysis, 1% in report
- Analysis > simulation >> implementation
  - The main contribution of your work is not what you have done, but is what you have found (or proved)

Thank you