

ns-2 Tutorial (2)

Multimedia Networking Group,
The Department of Computer Science, UVA
Jianping Wang

Contents:

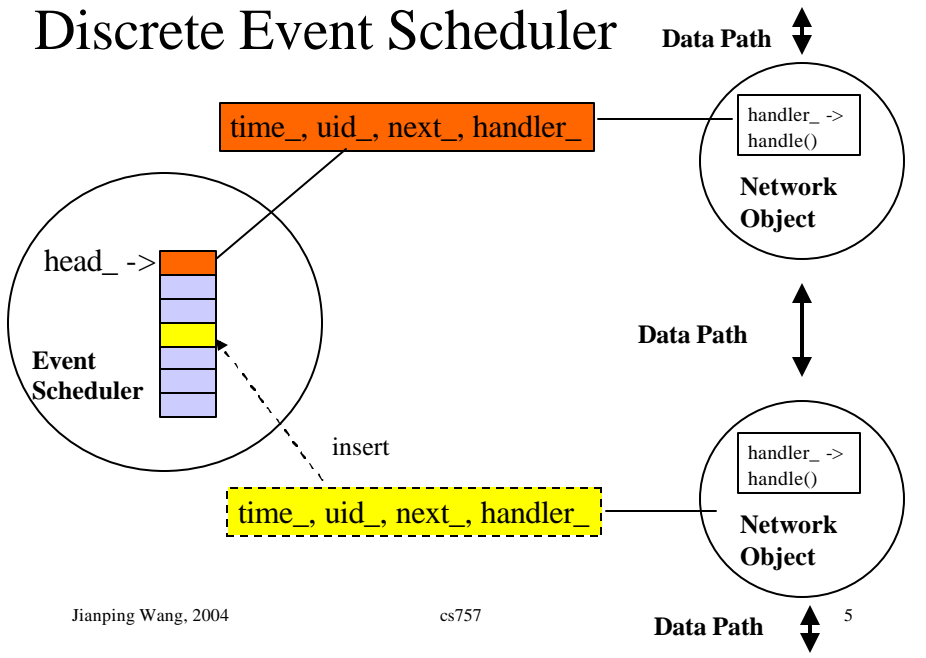
- Objectives of this week
 - What is ns-2?
 - Working with ns-2
 - Tutorial exercise
 - ns-2 internals
 - Extending ns-2
- } Today

ns-2 Internals

Internals

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

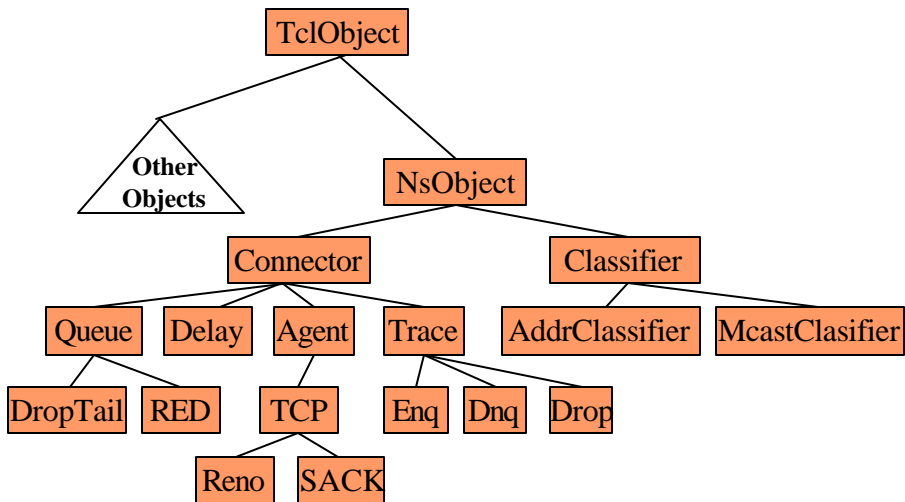
Discrete Event Scheduler



Jianping Wang, 2004

cs757

Class Hierarchy (Partial)

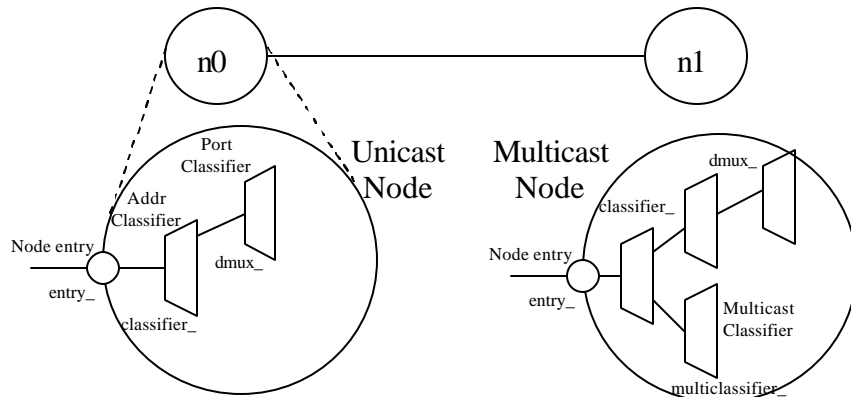


Jianping Wang, 2004

cs757

6

Network Topology - Node

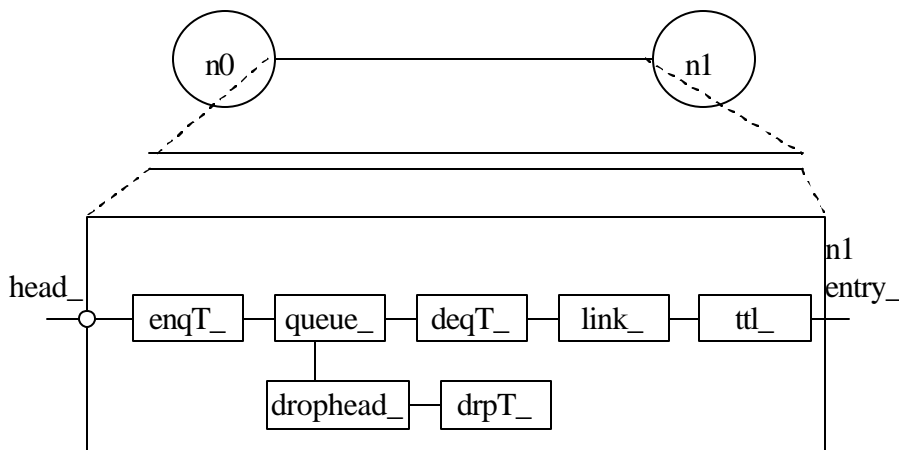


Jianping Wang, 2004

cs757

7

Network Topology - Link

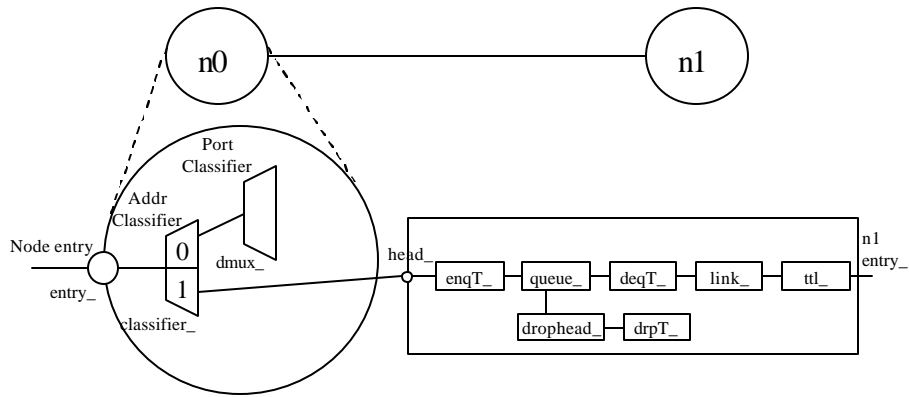


Jianping Wang, 2004

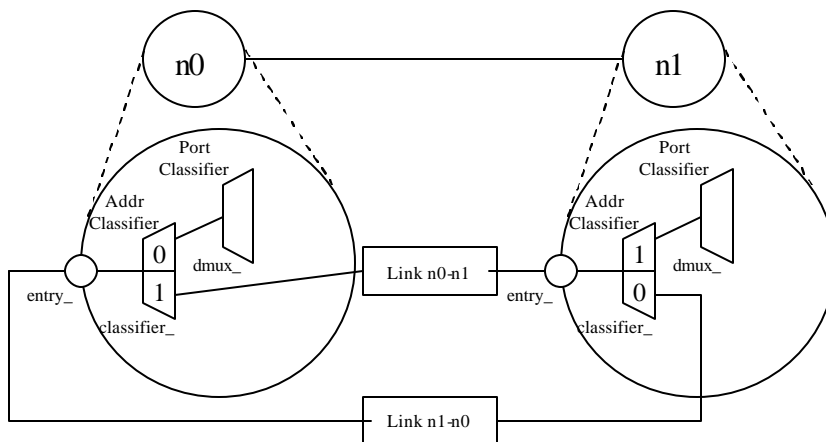
cs757

8

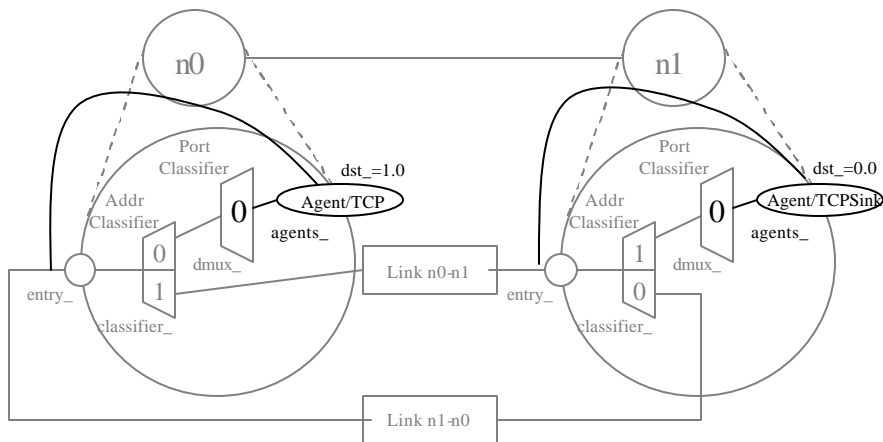
Routing



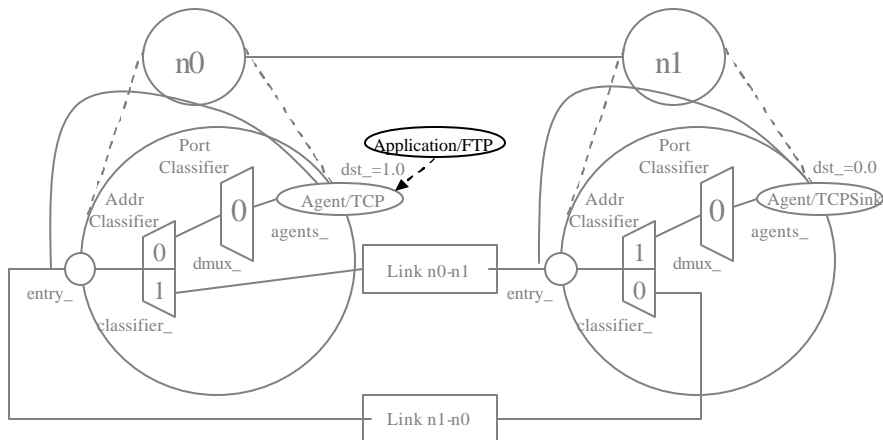
Routing (cont.)



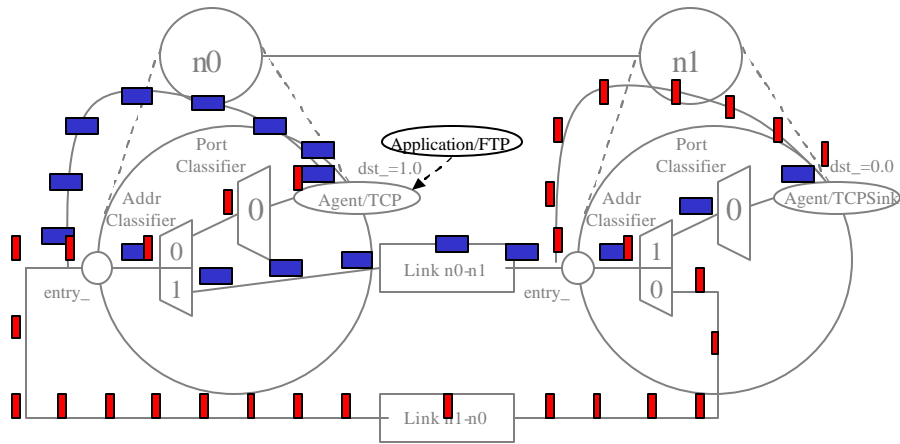
Transport



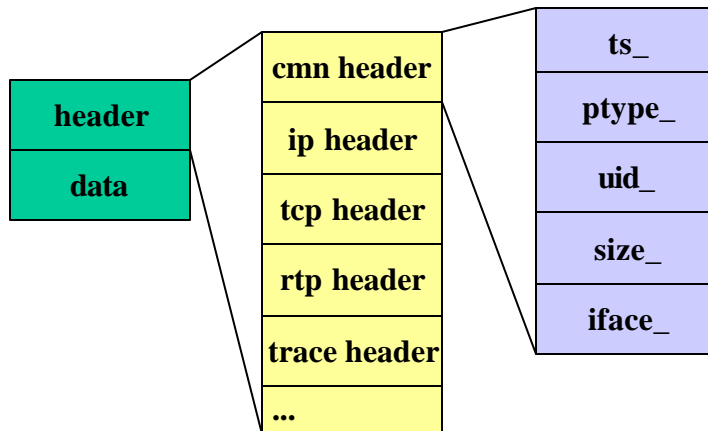
Application



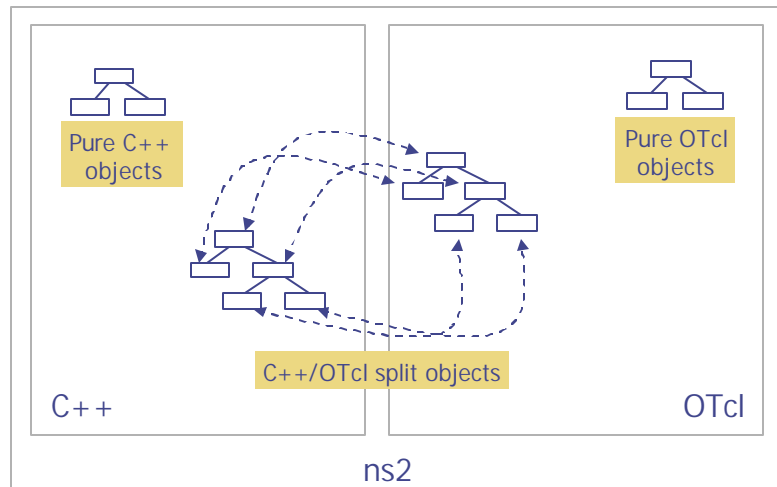
Packet Flow



Packet Format



OTcl and C++: The Duality



Jianping Wang, 2004

cs757

15

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 an OTcl object, and create a linkage between the OTcl object and C++ Object
Tcl	C++ methods to access Tcl interpreter
TclCommand	Standalone global commands
EmbeddedTcl	ns script initialization

Jianping Wang, 2004

cs757

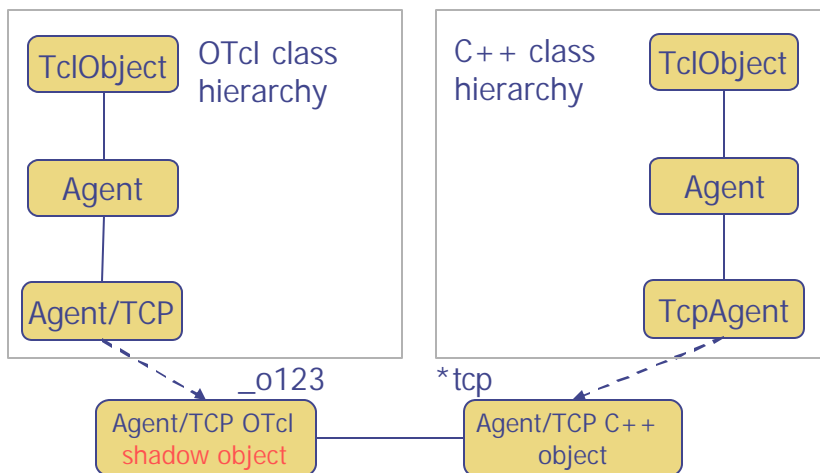
16

TclObject

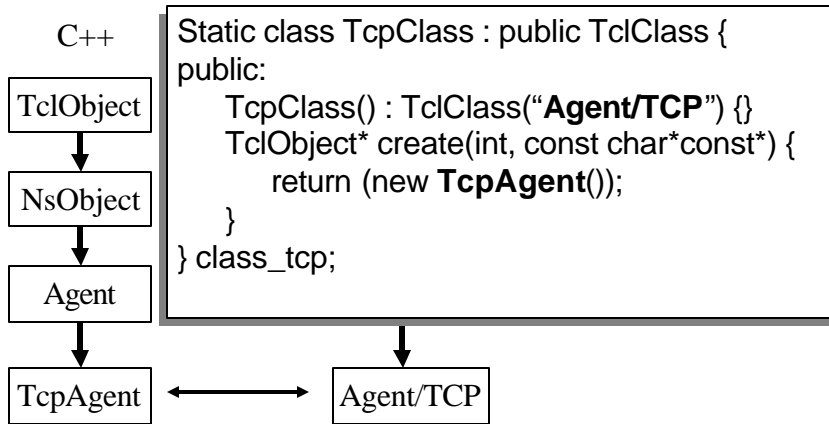
- Basic hierarchy in ns for split objects
- Mirrored in both C++ and OTcl
- Example

```
set tcp [new Agent/TCP]
```

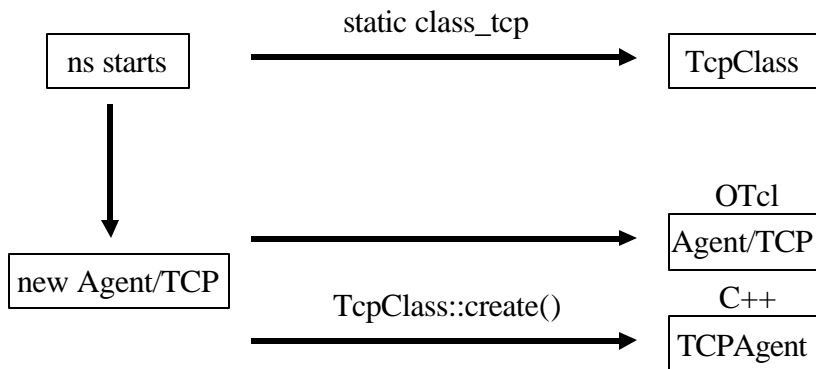
TclObject: Hierarchy and Shadowing



TclClass



The Creation of New Object



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

```
~/ns-2.1b8/tcl/lib/ns-default.tcl
```

– Otherwise a warning will be issued when the shadow object is created

TclObject::command()

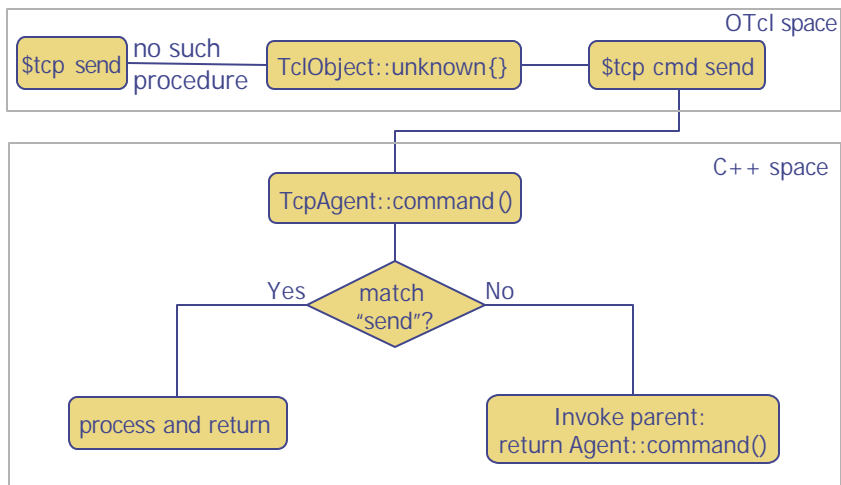
- OTcl

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

- C++

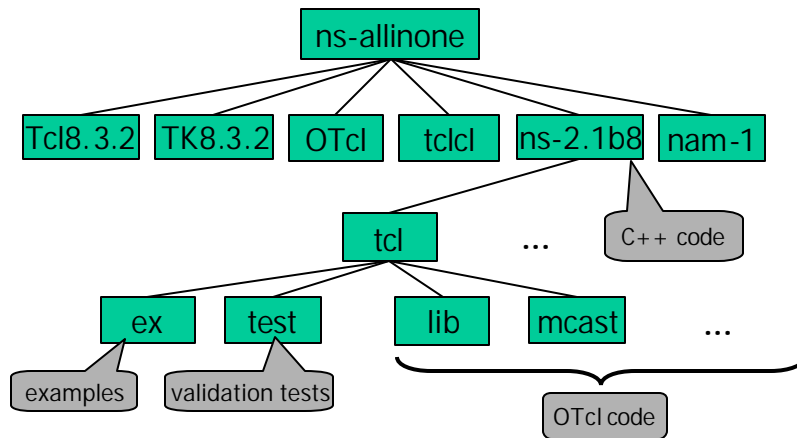
```
int TcpAgent::command(int argc,
                      const char*const* argv) {
    if (argc == 3) {
        if (strcmp(argv[1], "advance") == 0) {
            int newseq = atoi(argv[2]);
            .....
            return(TCL_OK);
        }
    }
    return (Agent::command(argc, argv);
}
}
}
```

TclObject::command()



Extending ns-2

ns Directory Structure

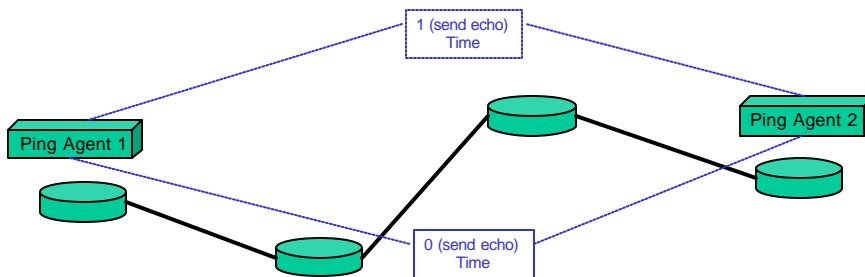


Two ways to Extend ns-2

OTcl or C++?

- OTcl
 - Simple Configuration, Setup, Scenario
 - If it's something that can be done without modifying existing Tcl module.
- C++
 - Anything that requires processing each packet
 - Needs to change behavior of existing module

Ping Protocol



What to extend?

- Create a new header object to store
 - Type of message (request or echo)
 - Timestamp
- Extend the Agent object
 - Implement methods
 - command()
 - recv()
- Create the Otcl object
 - Implement the recv{ } procedure

The new header for ping

- A Header is required by the new protocol, since new information needs to travel.

- Information to store

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

- Header object

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

Register Ping Header

```
enum packet_t {
    PT_TCP,
    .....,
    PT_PING
    PT_NTTYPE // this must be the last one
};

class p_info {
    p_info() {
        name_[PT_TCP] = "tcp";
        .....,
        name_[PT_PING] = "ping";
        name_[PT_NTTYPE] = "undefined";
    }
};
```

Add to ns-allinone~/ns2-2.1b8/packet.h

```
foreach prot {
    AODV
    .....,
    Ping
}{
    add-packet-header $prot
}
```

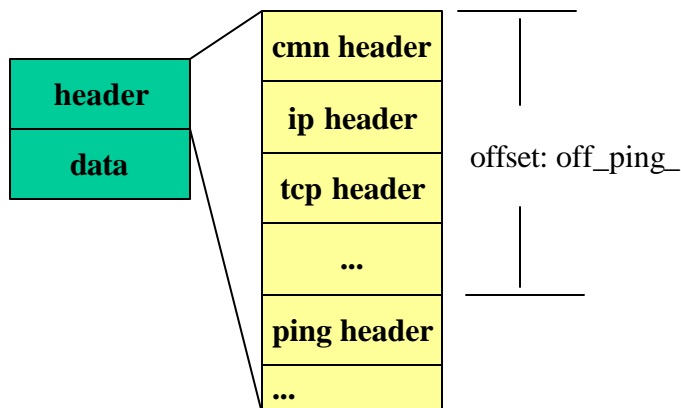
Add to ns-allinone~/ns2-2.1b8/tcl/lib/ns-packet.tcl

Jianping Wang, 2004

cs757

31

Ping Header in PingAgent

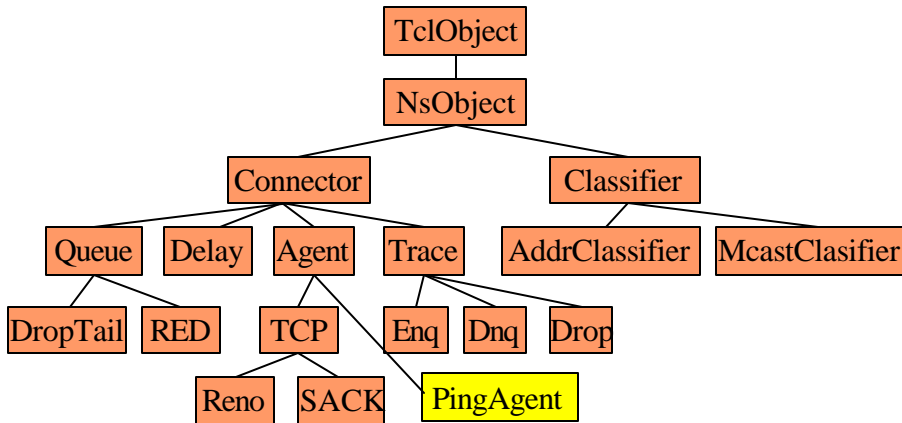


Jianping Wang, 2004

cs757

32

Class Hierarchy



Jianping Wang, 2004

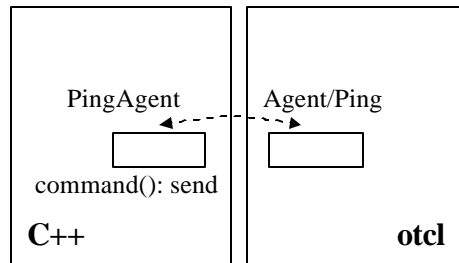
cs757

33

Ping Agent - overview

- class PingAgent

- PingAgent()
- recv()
- send()
- command()
 - send {send() }



- TclClass("Agent/Ping")

```

set msg [new Agent/Ping]
$msg send
  
```

Jianping Wang, 2004

cs757

34

Agent Ping (1/4)

- The Class

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

- Constructor for the class 'PingAgent' binds the variables which have to be accessed both in Tcl and C++**

```
PingAgent::PingAgent() : Agent(PT_PING)
{
    bind("packetSize_", &size_);
    bind("off_ping_", &off_ping_);
}
Jianping Wang, 2004 cs757
```

35

Agent Ping (2/4)

- Implementing the command() method

```
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));
}
Jianping Wang, 2004 cs757
```

36

Agent Ping (3/4)

- **Receiving a Request Packet**

```
void PingAgent::recv(Packet* pkt, Handler*)
{
    // Access the IP header for the received packet:
    hdr_ip* hdr_ip = (hdr_ip*)pkt->access(off_ip_);
    // Access the Ping header for the received packet:
    hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
    // Is the 'ret' field = 0 (i.e. the receiving node is being pinged)?
    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_ping* hdrret = (hdr_ping*)pktret->access(off_ping_);
        // 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);
    } ... continues
}
Jianping Wang, 2004
```

cs757

37

Agent Ping (4/4)

- **Receiving a Confirmation Packet**

```
else {
    // A packet was received. Use tcl.eval to call the Tcl
    // interpreter with the ping results.
    // Note: In the Tcl code, a procedure 'Agent/Ping recv {from rtt}'
    // has to be defined which allows the user to react to the ping
    // result.
    char out[100];
    // Prepare the output to the Tcl interpreter. Calculate the round
    // trip time
    sprintf(out, "%s recv %d %3.1f", name(),
            hdr_ip->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);
}
}
```

Jianping Wang, 2004

cs757

38

Create the Otcl object

- Add this to your main tcl simulation script.

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

The End