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

Class Hierarchy (Partial)

Jianping Wang, 2004
Network Topology - Node

Network Topology - Link
Routing

Routing (cont.)
Packet Flow

Packet Format

header
data

- cmn header
- ip header
- tcp header
- rtp header
- trace header
- ...
OTcl and C++: The Duality

C++/OTcl Linkage

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TclObject</strong></td>
<td>Root of ns-2 object hierarchy</td>
</tr>
<tr>
<td></td>
<td>bind(): link variable values between C++ and OTcl</td>
</tr>
<tr>
<td></td>
<td>command(): link OTcl methods to C++ implementations</td>
</tr>
<tr>
<td><strong>TclClass</strong></td>
<td>Create an OTcl object, and create a linkage between the OTcl object and C++ Object</td>
</tr>
<tr>
<td><strong>Tcl</strong></td>
<td>C++ methods to access Tcl interpreter</td>
</tr>
<tr>
<td><strong>TclCommand</strong></td>
<td>Standalone global commands</td>
</tr>
<tr>
<td><strong>EmbeddedTcl</strong></td>
<td>ns script initialization</td>
</tr>
</tbody>
</table>
TclObject

- Basic hierarchy in ns for split objects
- Mirrored in both C++ and OTcl
- Example
  ```
  set tcp [new Agent/TCP]
  ```

TclObject: Hierarchy and Shadowing
The Creation of New Object

ns starts -> static class/tcp -> TcpClass

new Agent/TCP -> TcpClass::create() -> C++ TCPAgent

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

- Link C++ member variables to OTcl object variables
- C++
  ```c++
  TcpAgent::TcpAgent() {
    bind("window_", &wnd_);
    ...
  }
  
  - bind_time(), bind_bool(), bind_bw()
- OTcl
  ```tcl
  set tcp [new Agent/TCP]
  $tcp set window_ 200
  ```

Initialization of Bound Variables

- Initialization through OTcl class variables
  ```tcl
  Agent/TCP set window_ 50
  ```
- Do all initialization of bound variables in
  ```bash
  ~/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);
  ```
Extending ns-2

ns Directory Structure

- ns-allinone
  - Tcl8.3.2
  - TK8.3.2
  - OTcl
  - tcl
  - ns-2.1b8
  - nam-1
  - tcl
    - ex
      - examples
    - test
      - validation tests
    - lib
    - mcast
      - C++ code
      - OTcl code
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
  ```c
  struct hdr_ping {
    char ret;
    double send_time;
  };
  ```
- Header object
  ```c
  static class PingHeaderClass : public PacketHeaderClass {
    public:
      PingHeaderClass() : PacketHeaderClass("PacketHeader/Ping",
      sizeof(hdr_ping)) {}
  } class_pinghdr;
  ```
 Register Ping Header

class p_info {
    p_info() {
        name_[PT_TCP] = "tcp";
        name_[PT_PING] = "ping";
        name_[PT_NTYPE] = "undefined";
    }
};

foreach prot {
    AODV
    Ping
} {
    add-packet-header $prot
}

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

Ping Header in PingAgent

header
  cmn header
  ip header
  tcp header
  ping header
    ... offset: off_ping_
Class Hierarchy

```
Class Hierarchy

TclObject
   /\  
NsObject 
   |  |
Connector
   |  |
Classifier
   |  |
Queue
   |  |
Delay
   |  |
Agent
   |  |
Trace
   |  |
AddrClassifier
   |  |
McastClassier
   |  |
DropTail
   |  |
RED
   |  |
TCP
   |  |
Enq
   |  |
Dnq
   |  |
Drop
   |  |
Reno
   |  |
SACK
   |  |
PingAgent

Jianping Wang, 2004
```
Agent Ping (1/4)

• The Class

```cpp
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++

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

Agent Ping (2/4)

• Implementing the command() method

```cpp
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));
}
```
Agent Ping (3/4)

- Receiving a Request Packet

```cpp
void PingAgent::recv(Packet* pkt, Handler*)
{
    // Access the IP header for the received packet:
    hdr_ip* hdrip = (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);
    } else { ... continues }
}
```

Agent Ping (4/4)

- Receiving a Confirmation Packet

```cpp
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(),
            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);
}
```
Create the Otcl object

- Add this to your main tcl simulation script.

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

The End