Extending ns

Padma Haldar
USC/ISI

Outline

- Extending ns
  - In OTcl
  - In C++
  - Debugging

ns Directory Structure

Extending ns in OTcl

- If you don’t want to compile
  - source your changes in your sim scripts
- Otherwise
  - Modifying code; recompile
  - Adding new files
    - Change Makefile (NS_TCL_LIB), 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 agent_addr
    $self send "$agent_addr send $seq"
    incr seq
    $ns at [expr $ns now+0.1] "$self send [lindex $msg 2]
      $server send [lindex $msg 2] $seq"
  }

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 seq $seq from $sdr"
    $self send "$agent_addr ack $seq"
  }

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"
  }

Agent/Message: Step 4

- Scheduler and tracing
  # Create scheduler
  set ns [new Simulator]
  # Turn on Tracing
  set ffd [new "$message.tr" w]
  $ns trace-all $ffd

Agent/Message: Step 5

- Topology
  for {set i 0} {$i < 6} {incr i} {
    $ns duplex-link $n($i) $n($i+1) 128kb 50ms DropTail
    $ns duplex-link $n(0) $n(1) 10Mb 1ms DropTail
    $ns queue-limit $n(0) $n(1) 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
  ```tcl
  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
  ```tcl
  set sdr [new Sender] $sdr set seq_ 0
  set packetSize_ 1000
  $sdr attach-agent $n(3) $sdr
  set rcvr [new Receiver] $rcvr set packetSize_ 40
  $rcvr attach-agent $n(5) $rcvr
  $ns at 1.1 "$sdr send-next"
  ```

Agent/Message: Step 9

- End-of-simulation wrapper (as usual)
  ```tcl
  $ns at 2.0 finish
  proc finish {} {
    global ns fd
    $ns flush-trace
    close $fd
    exit 0
  }
  $ns run
  ```

Agent/Message: Result

- Example output
  ```
  Receiver gets seq 0 from 3
  Sender gets ack 0 from 5
  Receiver gets seq 1 from 3
  Sender gets ack 1 from 5
  Receiver gets seq 2 from 3
  Sender gets ack 2 from 5
  Receiver gets seq 3 from 3
  ...  
  ```

Add Your Changes into ns

- `tcl/lib/ns-lib.tcl`
  ```
  Class Simulator
  ...
  source ../mysrc/msg.tcl
  ```
- `Makefile`
  ```
  NS_TCL_LIB = \
  tcl/mysrc/msg.tcl \n  ```
  Or change `Makefile` in `ns-allinone` directory, then run `./configure --enable-debug`, `make depend` and `make`
Outline

- Extending ns
  - In OTcl
  - In C++
    - New components

Extending ns in C++

- Modifying code
  - make depend
  - Recompile
- Adding code in new files
  - Change Makefile
  - make depend
  - recompile

Creating New Components

- Guidelines
- Two styles
  - New agent based on existing packet headers
  - Add new packet header

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)

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

Packet Format

- Packet Format Diagram

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

- How Packet Header Works Diagram

Example: Agent/Message

- New packet header for 64-byte message
- New transport agent to process this new header
New Packet Header – Step 1

- Create header structure

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

New Packet Header – Step 2

- PacketHeader/Message

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

New Packet Header – Step 3

- Enable tracing (packet.h):

```c
enum packet_t {
    PT_TCP,
    …
    PT_MESSAGE, // 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)

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

Packet Header: Caution

- Some old code, e.g.:

```c
RtpAgent::RtpAgent() { 
    …
    bind("off_rtp", &off_rtp);
}
```

- Don’t follow this example!
Agent/Message – Step 2

- C++ class definition

```cpp
// Standard split object declaration
class MessageAgent : public Agent {
public:
    MessageAgent() : Agent::Agent(Agent::PT_MESSAGE) {}

    virtual int command(int argc, const char* argv[argc]);

    virtual void recv(Packet*, Handler*);
};
```

Agent/Message – Step 3

- Packet processing: send

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

Agent/Message – Step 4

- Packet processing: receive

```cpp
void MessageAgent::recv(Packet* pkt, Handler*)
{
    hdr_msg* msg = hdr_msg::access(pkt);
    // OTcl callback
    char wrk[128];
    sprintf(wrk, "%s recv %s", name(), msg);
    Tcl tcl = Tcl::instance();
    tcl.eval(wrk);
    Packet::free(pkt);
}
```

Outline

- Extending ns
  - In OTcl
  - In C++
- Debugging: OTcl/C++, memory
- Pitfalls

Debugging C++ in ns

- C++/OTcl debugging
  - Memory debugging
    - purify
    - dmalloc

C++/OTcl Debugging

- Usual technique
  - Break inside command()
  - Cannot examine states inside OTcl!
- Solution
  - Execute tcl-debug inside gdb
C++/OTcl Debugging

 działania:

- (gdbg) call Tcl::instance()->eval("debug 1")
- lappend auto_path $dbg_library
- 0: application
- lappend auto_path $dbg_library
dbg 1.4> Simulator info instances
- _cl
dbg 1.5> _cl now
- 0
dbg 1.6> # and other fun stuff
dbg 1.7> <
- (gdbg) where
- #0 0x102218 in write()

Memory Debugging in ns

- Purify
  - Set PURIFY macro in ns Makefile
  - Usually, put -collocter=<ld_path>
- 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 \"l\"`'
  - dmalloc -l log low
- dmalloc_summarize ns < logfile
  - ns must be in current directory
  - Itemize how much memory is allocated in each function

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

Memory Conservation Tips

- Remove unused packet headers
- Avoid trace-all
- Use arrays for a sequence of variables
  - Instead of new, say n($i)
- Avoid OTcl temporary variables
- Use dynamic binding
  - `delay_bind()` instead of `bind()`
- See object [h,cc]
- See tips for running large sim in ns at [www.isi.edu/ns/nsnam/ns-largesim.html](http://www.isi.edu/ns/nsnam/ns-largesim.html)

Memory Leaks

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

Final Word

- My extended ns dumps OTcl scripts!
  - Find the last 10-20 lines of the dump
  - Is the error related to "_o*** cmd ...
  - Check your command()
  - Otherwise, check the otcl script pointed by the error message