ns-2 Tutorial

Extending the Simulator

Matthias Transier

transier@informatik.uni-mannheim.de

Universität Mannheim

Based on a tutorial by Polly Huang, ETHZ

Overview

- Packets and headers
  - General mode of operation
  - Defining a new header type
- Agents
  - Example implementation
- Debugging

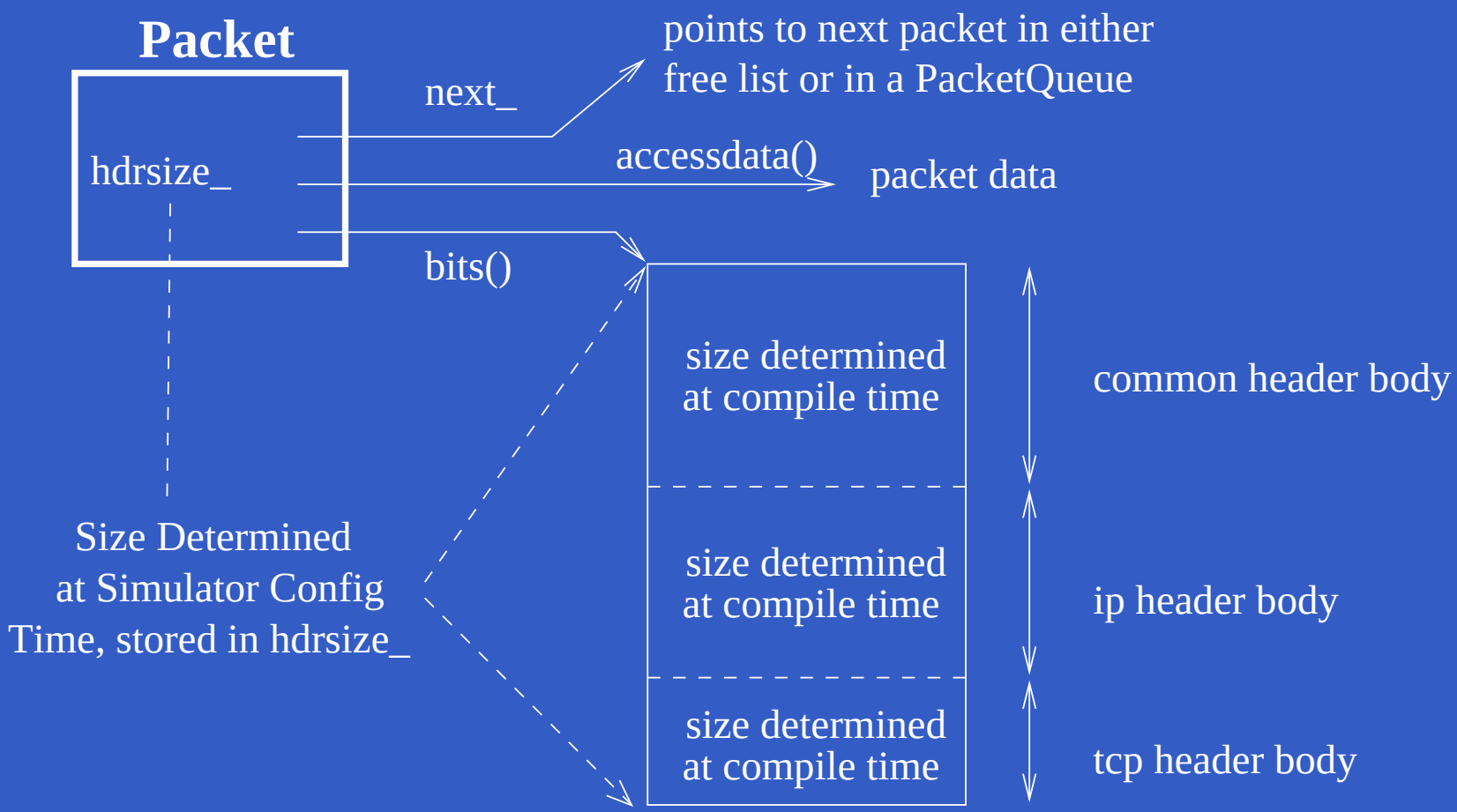
Packets and headers

Packets and headers

- Packets are the basic object for data exchange
- ns-2 has its own memory management for allocating packets
- Each packet contains all enabled packet headers
- Compiled-in packet formats are enabled in OTcl before start of simulation via the PacketHeaderManager:

```
remove-all-packet-headers
add-packet-header IP TCP
...
set ns [new Simulator]
```

Packets


Packet
hdrsize_
next_
points to next packet in either
free list or in a PacketQueue
accessdata()
packet data
bits()
size determined
at compile time
size determined
at compile time
size determined
at compile time
common header body
ip header body
tcp header body
Size Determined
at Simulator Config
Time, stored in hdrsize_

Common header

- Each packet contains a common header
- The common header contains the following information:
  - a timestamp,
  - a numeric packet type,
  - a simulator-wide unique id,
  - the simulated packet size,
  - and a field for a label (used e.g. for wired multicast).

Header classes

How do the header classes work?

- a static class "`MyHeader`" is derived from `PacketHeaderClass`
- `PacketHeaderClass` is a subclass of `TclClass`

Header classes

How do the header classes work?

- a static class "`MyHeader`" is derived from `PacketHeaderClass`
- `PacketHeaderClass` is a subclass of `TclClass`
- its constructor is called with:
  - a name for the Tcl class
  - the size of the header structure
  - an offset of 0

Header classes

How do the header classes work?

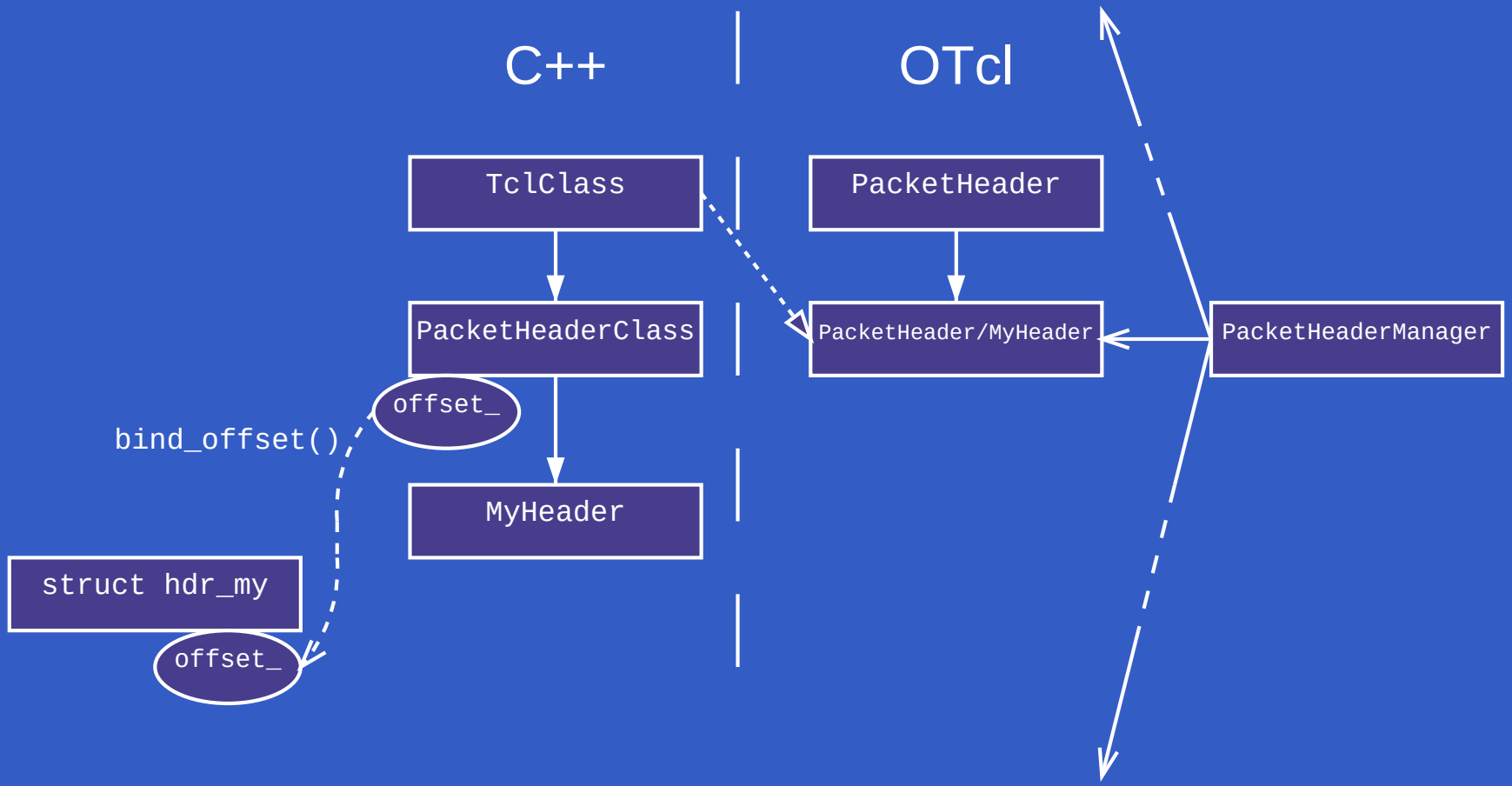
- a static class "`MyHeader`" is derived from `PacketHeaderClass`
- `PacketHeaderClass` is a subclass of `TclClass`
- its constructor is called with:
  - a name for the Tcl class
  - the size of the header structure
  - an offset of 0
- a call to `bind_offset` shares the variable pointer of the struct

Header classes

How do the header classes work?

- a static class "`MyHeader`" is derived from `PacketHeaderClass`
- `PacketHeaderClass` is a subclass of `TclClass`
- its constructor is called with:
  - a name for the Tcl class
  - the size of the header structure
  - an offset of 0
- a call to `bind_offset` shares the variable pointer of the struct
- the PacketHeaderManager sets the correct offset value on start

Header classes II



Adding a new header type I

- First, create the structure:

```
struct hdr_msg {
  char msg_[64];
  static int offset_;
  inline static int& offset() { return offset_; }
  inline static hdr_msg* access(Packet* p) {
    return (hdr_msg*) p->access(offset_);
  }
  char* msg() { return (msg_); }
  int maxmsg() { return (sizeof(msg_)); }
};
```

Adding a new header type II

- Create a static class for OTcl linkage:

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

Adding a new header type III

- Defi ne a numeric packet type in packet.h:

```
enum packet_t {
  PT_TCP,
  ...,
  PT_MESSAGE,
  PT_NTYPE // This MUST be the LAST one
};
class p_info {
  ...
    name_[PT_MESSAGE] = "message";
    name_[PT_NTYPE] = "undefined";
  ...
};
```

Adding a new header type IV

- Add support for packet tracing in cmu-trace.cc:

```
void CMUTrace::format_msg(Packet *p, int offset) {
  struct hdr_msg *mh = hdr_cmn::access(p);
  sprintf(pt_->buffer() + offset, "%s", mh->msg());
}

void CMUTrace::format(Packet* p, const char *why) {
  ...
    case PT_MSG:
      format_msg(p, offset);
    default:
  ...
}
```

Agents

Agents

- Agents are used as traffi c endpoints or at various protocol layers
- Interface to other agents: `send` and `recv` functions
- Agent types are defi ned by static split object classes
- A new instance of an agent is created via OTcl:

```
set newtcp [new Agent/TCP]
$newtcp set window_ 20
$newtcp set portID_ 1
```

Creating a new agent I

- MessageAgent should exchange messages of format:

  ```
  Addr Op SeqNo
  ```

- First, create derived C++ class:

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

- Specify static split object (as seen for headers)

Creating a new agent II

- Implement a possibility to send packets, e.g. via OTcl interface:

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

Creating a new agent III

- Defi ne a receive function:

```
void MessageAgent::recv(Packet* pkt, Handler*)
{
  hdr_msg* mh = hdr_msg::access(pkt);

  char wrk[128]; /* Caution! */
  sprintf(wrk, "%s recv {%s}", name(), mh->msg());
  Tcl& tcl = Tcl::instance();
  tcl.eval(wrk);

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

Creating a new agent IV

- Defi ne receive function in OTcl object:

```
Agent/Message instproc recv msg {
  set src [lindex $msg 0]
  set type [lindex $msg 1]
  set seq [lindex $msg 2]
  puts -nonewline [$self set agent_addr_]
  puts " received ’$type ($seq)’ from $src"
  if {$type == "send"} {
    $self send "$addr_ ack $seq"
  }
}
```

Debugging

Debugging ns-2 code

- First choice: printf() and puts“”
- Use of debuggers:
  - C++ parts: gdb
    - Cannot examine states inside OTcl at any time
  - OTcl: tcl-debug
    - add `debug 1` to OTcl source
  - Mixed evaluation of OTcl state from within C++ source
    - execute gdb and invoke tcl-debug