Packet, Packet Header, and Header Format
Outline

• Overview
• Packet Allocation and Deallocation
• Packet Header
• Data Payload
• Customizing Packets
• Summary
Overview: Outline

- Packet Architecture
- Packets As Events, and Delayed Packet Reception
- A Link List of Packets
- A Free Packet List

Overview

• The main objective: Send packets from one node to another.
• In NS2,
  - **NO** packet is actually sent.
  - Tell Node n2 that a packet arrives.
• How does it actually work?

Overview

- A packet consists of
  - Packet Header
  - Data Payload

- In most case, you do not need to send the entire packet.

- You may need to tell NS2
  - Packet length: Tell n2 when the packet (i.e., `hdr_cmn::size_`) is received.
  - Bit error rate (BER): Simulate error and send packet only if it is not in error.

- No need to send entire packet

Packet Architecture

- 5 main parts:
  1. Actual packet
  2. Class Packet
  3. Protocol specific header
  4. Packet header manager
  5. Data payload

Actual Packet

• The actual memory where the packet is stored.

• Two main parts
  - $\text{bits}_\text{in}$: Store headers
  - $\text{data}_\text{in}$: Store payload $\Rightarrow$ Again, not being used in most cases

Class Packet: Variables

- Main packet class

```cpp
//~/ns/common/packet.h
class Packet : public Event {
private:
    unsigned char* bits_;
   AppData* data_; 
   bool fflag_; 
protected:
    static Packet* free_; 
    int ref_count_; 
public:
    Packet* next_; 
    static int hdrlen_; 

    ...
}
```

# Class Packet: Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits_</td>
<td>A bag of bits which contain packet header</td>
</tr>
<tr>
<td>data_</td>
<td>Pointer to an AppData object which contains data payload</td>
</tr>
<tr>
<td>fflag_</td>
<td>True if the packet is in use.</td>
</tr>
<tr>
<td>free_</td>
<td>A pointer to the head of the packet free list</td>
</tr>
<tr>
<td>ref_count_</td>
<td>Number of objects which are using the packet</td>
</tr>
<tr>
<td>next_</td>
<td>A pointer the next packet in the linked list of packets</td>
</tr>
<tr>
<td>hdrlen_</td>
<td>Length of packet header</td>
</tr>
</tbody>
</table>

Class Packet: Functions

//~/ns/common/packet.h
class Packet : public Event {
private:
    static void init(Packet*) {bzero(p->bits_, hdrlen_);}
public:
    //Packet Allocation and Deallocation
    Packet() : bits_(0), data_(0), ref_count_(0), next_(0) { }
    inline unsigned char* const bits() { return (bits_); }
    inline Packet* copy() const;
    inline Packet* refcopy() { ++ref_count_; return this; }
    inline int& ref_count() { return (ref_count_); }
    static inline Packet* alloc();
    static inline Packet* alloc(int);
    inline void allocdata(int);
    static inline void free(Packet*);

    //Packet Access
    inline unsigned char* access(int off){return &bits_[off]);}

# Class Packet: Functions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>init(p)</td>
<td>Clears the packet header bits_ of the input packet p.</td>
</tr>
<tr>
<td>copy()</td>
<td>Returns a pointer to a duplicated packet.</td>
</tr>
<tr>
<td>refcopy()</td>
<td>Increases the number of objects, which refer to the packet, by one.</td>
</tr>
<tr>
<td>alloc()</td>
<td>Creates a new packet and returns a pointer to the created packet.</td>
</tr>
<tr>
<td>alloc(n)</td>
<td>Creates a new packet with “n” bytes of data payload and returns a pointer to the created packet.</td>
</tr>
<tr>
<td>allocdata(n)</td>
<td>Allocates “n” bytes of data payload to the variable data_.</td>
</tr>
<tr>
<td>free(p)</td>
<td>Deallocates packet p.</td>
</tr>
<tr>
<td>access(off)</td>
<td>Retrieves a reference to a certain point of the variable bits_ (i.e., packet header).</td>
</tr>
</tbody>
</table>

Packet Architecture

- 5 main parts:
  1. Actual packet
  2. Class Packet
  3. Protocol specific header
  4. Packet header manager
  5. Data payload

Protocol Specific Header

- Packet header classified based on types.
  - Common packet header
  - IP packet header
  - TCP packet header

- Defined on a portion of packet header or

  \*bits_

The header length is stored in C++

  Packet::hdrlen_,

which is bound to

OTcl:

  PacketHeaderManager::hdrlen

Protocol specific header size is determined during the compilation.

The header length is stored in C++

  Packet::hdrlen_,

which is bound to

OTcl:

  PacketHeaderManager::hdrlen

Packet header size is determined during the construction of the simulator.

Class Packet: Variables

• Main packet class

```cpp
//~/ns/common/packet.h
class Packet : public Event {
private:
    unsigned char* bits_;  
    AppData* data_;        
    bool fflag_;           
protected:
    static Packet* free_;  
    int ref_count_;        
public:
    Packet* next_;         
    static int hdrlen_;    
    ...
};
```

Q: What does it mean?
A: Every packet has the same header length.

Protocol Specific Header

- Protocol specific header consists of 3 classes:
  1. A C++ class: stores packet attributes
  2. An OTcl class: acts as an interface to the OTcl domain
  3. A mapping class: binds the above C++ and OTcl classes

E.G.,

<table>
<thead>
<tr>
<th>C++</th>
<th>Mapping</th>
<th>OTcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdr_cmn</td>
<td>CommonHeaderClass</td>
<td>PacketHeader/Common</td>
</tr>
</tbody>
</table>

Packet Header Manager and Data Payload

• Packet Header Manager
  - Maintain the list of active protocol
  - Create header based on the list

• Data Payload
  - Try not to use it!!
  - Store actual data (i.e., *data_*)

Overview: Outline

- Packet Architecture
- Packets As Events, and Delayed Packet Reception
- A Link List of Packets
- A Free Packet List

Packets as Events

- **Class Packet** is a child class of **Class Event**.

<table>
<thead>
<tr>
<th>Event ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Arrival</td>
<td>Arrival</td>
<td>Arrival</td>
<td>Arrival</td>
<td>Arrival</td>
<td>Arrival</td>
</tr>
<tr>
<td>Time</td>
<td>0.8</td>
<td>1.5</td>
<td>5.2</td>
<td>7.4</td>
<td>9.4</td>
<td>11.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event ID</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Departure</td>
<td>Departure</td>
<td>Departure</td>
</tr>
<tr>
<td>Time</td>
<td>2.4</td>
<td>5.5</td>
<td>10.5</td>
</tr>
</tbody>
</table>

Packets as Events

- Recall:
  Event = AtEvent;  Handler = AtHandler

- For Packets:
  Event = Packet;
  Handler = NsObject

Delayed Packet Reception

• Suppose we have would like to
  - Send a Packet whose pointer is $p$
  - To an NsObject whose pointer is $target_\_$

• Two types of packet forwarding
  1. Immediate: $(target_-, p)$
  2. Delayed by $d$ seconds: $s.schedule(target_-, p, d)$

• What is $s$? How do we obtain it?
  - $s$ is a Scheduler object
    - $Scheduler\& s = Scheduler::instance();$
Overview: Outline

- Packet Architecture
- Packets As Events, and Delayed Packet Reception
- A Link List of Packets
- A Free Packet List
Class Packet: Variables

• Main packet class

```cpp
#include "ns/common/packet.h"
class Packet : public Event {
private:
    unsigned char* bits_;  
    AppData* data_; 
    bool fflag_; 
protected:
    static Packet* free_; 
    int ref_count_; 
public:
    Packet* next_; 
    static int hdrlen_; 

    ...
}
```

A Link List of Packets

- A class pointer variable `next_` provide a support to create link list
- **Class** `PacketQueue` form a link list of packets:
  - `head_`: a pointer to the first packet
  - `tail_`: a pointer to the last packet

A Link List of Packets

- **Two main functions of class** `PacketQueue`
  - `enqueue(p)`: Put a packet *p* in the `PacketQueue`
  - `dequeue()`: Take the "head" (i.e., first) packet of the `PacketQueue`

Overview: Outline

- Packet Architecture
- Packets As Events, and Delayed Packet Reception
- A Link List of Packets
- A Free Packet List
A Free Packet List

• What would NS2 do after a packet is no longer in use (e.g., reaches the destination)?
  - Destroy it!, or
  - Keep it and reuse it!

• NS2 keeps packets not in use in a “free packet list”.

• Advantage: No need to allocate/deallocate memory for several times

• Disadvantage: Waste of memory
A Free Packet List

• When NS2 need a new packet
  1. Take a packet from the free packet list, or (if (1) is not possible)
  2. Create a new packet

• When a packet is no longer needed
  - Put it in a free packet list

• When do we return the memory to the system?
  ➔ When the simulation terminates

Class Packet: Variables

• Main packet class

```cpp
#include //~/ns/common/packet.h

class Packet : public Event {
    private:
        unsigned char* bits_;
        AppData* data_;        // True if the packet is in use:
        bool fflag_;  

    protected:
        static Packet* free_;  // You cannot put the packet on the free packet list
        int ref_count_;  

    public:
        Packet* next_;        // If fflag_ is true.
        static int hdrlen_;  // A pointer to the first packet on the free packet list
        static int
...  
    }
```

A Free Packet List

Q: Why do I write free_ outside the class? Why do all packets instance points to the same free_?

A: free_ is a static variable

- How a free packet list is actually used?
  ➤ Packet allocation/deallocation

Outline

• Overview
• Packet Allocation and Deallocation
• Packet Header
• Data Payload
• Customizing Packets
• Summary

Packet Allocation/Deallocation

- **Allocation**
  - When a packet is needed
  - Take a packet from the free packet list
  - If not possible, create a new one

- **Deallocation**
  - When a packet is no longer needed
  - Put it in the free packet list

Packet Allocation

- **Main C++ function** `alloc()`
  - Two steps: 1) Allocation, and 2) Initialization

```cpp
//~/ns/common/packet.h
inline Packet* Packet::alloc()
{
    Packet* p = free_;  // there is at least one packet in the free packet list
    if (p != 0) {
      assert(p->fflag_ == FALSE);
      free_ = p->next_;  
      assert(p->data_ == 0);
      p->uid_ = 0; p->time_ = 0;
    } else {
      p = new Packet;
      p->bits_ = new unsigned char[hdrlen_];
    }
    init(p); // Clear the space in bits_
    p->next = 0; p->fflag = TRUE;
}
```

Packet Allocation: Other functions

- `alloc(int n)`: creates a packet with payload size $n$
- `allocdata(int n)`: creates a data payload with size $n$

```cpp
//~/ns/common/packet.h
inline Packet* Packet::alloc(int n)
{
  Packet* p = alloc();
  p->allocdata(n);
  return (p);
}
inline void Packet::allocdata(int n)
{
  data_ = new PacketData(n);
}
```

Packet Allocation: Other functions

- copy(): Creates a duplicated packet.
- refcopy(): Increment ref_count_; not actually create a duplicated packet.

//~/ns/common/packet.h
inline Packet* Packet::copy()
{
    Packet* p = alloc();
    memcpy(p->bits(), bits_, hdrlen_);
    if (data_)
        p->data_ = data_->copy();
    return (p);
}

inline Packet* refcopy() { ++ref_count_; return this; }
Packet Allocation: ref_count_

- The variable \texttt{ref\_count}:
  - The number of objects which refers to this packet
  - Several objects may refer to the same object
  - Refer = Have its pointer point to this packet

Class Packet: Variables

• Main packet class

```c++
//~/ns/common/packet.h
class Packet : public Event {
private:
    unsigned char* bits_;  // True if the packet is in use:
    AppData* data_;        // You cannot put the packet
    bool fflag_;           // on the free packet list
protected:
    static Packet* free_;  // If fflag_ is true.
    int ref_count_;        // No. of objects which have its variable
public:
    Packet* next_;        // point to this packet
    static int hdrlen_;    
    ...
};
```

Packet Deallocation

- **Main function free()**

```c
//~/ns/common/packet.h
inline void Packet::free(Packet* p)
{
    if (p->fflag_) {
        if (p->ref_count_ == 0) {
            assert(p->uid_ <= 0);
            if (p->data_ != 0) {
                delete p->data_; p->data_ = 0;
            }
            init(p); p->fflag_ = FALSE;
            p->next_ = free_; free_ = p;
        } else {
            --p->ref_count_;
        }
    }
}
```

- Only free packet which is currently in use
- We do not have to do anything with packets which is not in use

- **ref_count_ > 0**
- means somebody is still using the packet
- Decrease ref_count_

**Cleanup the packet**

**Put the packet at the head of the free packet list**

Packet Deallocation

• **Main function** `free()`

```c
inline void Packet::free(Packet* p)
{
    if (p->fflag_) {
        if (p->ref_count_ == 0) {
            assert(p->uid_ <= 0);
            if (p->data_ != 0) {
                delete p->data_; p->data_ = 0;
            }
        } else {
            --p->ref_count_;
        }
    } else {
        init(p); p->fflag_ = FALSE;
        p->next_ = free_; free_ = p;
    }
}
```

Outline

• Overview
• Packet Allocation and Deallocation
• Packet Header
• Data Payload
• Customizing Packets
• Summary
Packet Header: Outline

- Protocol Specific Header (PSH) Architecture
- Packet Type
- Protocol Specific Header
- Packet Header Access Mechanism
- Packet Header Manager
- Construction of Packet Header

**PSH Architecture**

- Examples of PSH: Common, IP, TCP, etc.
- Stored in variable `bits_` of a Packet object
- **Two-level** packet header architecture
  1. Composition of PSH
  2. `C++ struct` data type containing PSH

Q. The size of the entire packet header is stored in

---

PSH Architecture

- **Variable** `bits_` is a “bag of bits”
- **1st level:**
  - Allocate spaces for PSHs
  - Use an “offset” concept

### PSH Architecture: Offset

- **2\textsuperscript{nd} Level**: An attribute of a PSH
- **Use C++ struct data type**
- **Main field** = \texttt{offset} = No. of bytes between
  - The beginning of \texttt{Packet::bits}, and
  - The beginning of the space allocated to a PSH

---

An Example of PSHs

- A common PSH
- Contain common attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet size</td>
<td>size_</td>
</tr>
<tr>
<td>Packet type</td>
<td>ptype_</td>
</tr>
<tr>
<td>Unique ID</td>
<td>uid_</td>
</tr>
<tr>
<td>Offsetting value</td>
<td>offset_</td>
</tr>
</tbody>
</table>
Common Header: C++ Code

//~/ns/common/packet.h
struct hdr_cmn {
    packet_t ptype_;    // packet type
    int size_;          // simulated packet size
    int uid_;           // unique id
    dir_t  direction_;  // direction: 0=none, 1=up, -1
    static int offset_; // offset for this header

    inline static hdr_cmn* access(const Packet* p) {
        return (hdr_cmn*) p->access(offset_);
    }

    inline static int& offset() { return offset_; }
    inline packet_t& ptype() { return (ptype_); }
    inline int& size() { return (size_); }
    inline int& uid() { return (uid_); }
};

Common Header: Offseting

- **The variable** `offset_` **is static**
  - **Only one** `offset_` **instance**
  - **Every packet has the same value of** `hdr_cmn::offset_`

- **Function** `access(Packet* p)`
  - **Static** ➔ **can be invoked from anywhere**
  - **Return a pointer to common header of the packet** `*p`

- **E.g., To set the size of Packet** `*p` **to be** `my_size`

  ```
  hdr_cmn* chdr = hdr_cmn::access(p);
  chdr->size_ = my_size;
  ```

Common Header: C++ Code

```cpp
#include <ns3/packet.h>

struct hdr_cmn {
  packet_t ptype_;  // packet type
  int size_;       // simulated packet size
  int uid_;        // unique id
  dir_t direction_; // direction: 0=none, 1=up, -1
  static int offset_; // offset for this header

  inline static hdr_cmn* access(const Packet* p) {
    return (hdr_cmn*) p->access(offset_);
  }
  inline static int& offset() { return offset_; }
  inline packet_t& ptype() { return (ptype_); }
  inline int& size() { return (size_); }
  inline int& uid() { return (uid_); }
};
```

The type of `ptype_` is `packet_t`. ➔ What is `packet_t`?
Packet Header: Outline

- Protocol Specific Header Architecture
- Packet Type
- Protocol Specific Header
- Packet Header Access Mechanism
- Packet Header Manager
- Construction of Packet Header

Packet Type

- Defined in Common Header, not in TCP or IP
- Define in `enum packet_t` below:

```c
//~/ns/common/packet.h
enum packet_t {
    PT_TCP,
    PT_UDP,
    PT_CBR,
    PT_AUDIO,
    PT_VIDEO,
    PT_ACK,
    ...
    PT_NTYPE // This MUST be the LAST one
};
```

Q: What is its actual value?
   a. string
   b. integer
   c. floating point
   d. boolean

Packet Type

• How do we show the type of packet?
  ➔ class p_info (i.e., packet info.)

• An external variable packet_info of class p_info

• An array member variable name_
  - Index = an element of packet_t
  - Value = The corresponding string

• A function name(packet_t p)
  - Return a string corresponding to p

Packet Type

class p_info {
public:
    p_info() {
        name_[PT_TCP] = "tcp";
        name_[PT_UDP] = "udp";
        ...
        name_[PT_NTYPE] = "undefined";
    }
    const char* name(packet_t p) const {
        if (p <= PT_NTYPE) return name_[p];
        return 0;
    }
private:
    static char* name_[PT_NTYPE+1];
};
extern p_info packet_info; /* map PT_* to string name */

Packet Header: Outline

- Protocol Specific Header Architecture
- Packet Type
- Protocol Specific Header
- Packet Header Access Mechanism
- Packet Header Manager
- Construction of Packet Header
Protocol Specific Header

- Store attributes specific to a certain protocols.
- 48 built-in protocol specific headers
- Each protocol involves 3 following parts:
  1. C++ class:
     - struct data type
     - The place where packet info is stored
     - Format: hdr_<XXX> (e.g., hdr_cmn)
  2. OTcl class:
     - OTcl interface
     - Format: PacketHeader/<XXX> (e.g., PacketHeader/Common)
  3. Mapping class:
     - Map a C++ class to an OTcl class
     - Format: <XXX>HeaderClass (e.g., CommonHeaderClass)

# Protocol Specific Header

- **Examples:**

<table>
<thead>
<tr>
<th><code>C++ Class</code></th>
<th><code>Mapping class</code></th>
<th><code>OTcl class</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>DropTail</td>
<td>DropTailClass</td>
<td>Agent/TCP</td>
</tr>
<tr>
<td></td>
<td>CommonHeaderClass</td>
<td></td>
</tr>
</tbody>
</table>

**C++ Class: Struct**

```cpp
//~/ns/common/packet.h
struct hdr_cmn {
    packet_t ptype_;    // packet type
    int size_;          // simulated packet size
    int uid_; // unique id
    dir_t direction_; // direction: 0=none, 1=up, -1
    static int offset_; // offset for this header

    inline static hdr_cmn* access(const Packet* p) {
        return (hdr_cmn*) p->access(offset_);
    }
    inline static int& offset() { return offset_; }
    inline packet_t& ptype() { return (ptype_); }
    inline int& size() { return (size_); }
    inline int& uid() { return (uid_); }
};
```

Mapping Class

• The base class is PacketHeaderClass

```cpp
//~/ns/common/packet.h
class PacketHeaderClass : public TclClass {
protected:
    PacketHeaderClass(const char* classname, int hdrlen) :
        TclClass(classname), hdrlen_(hdrlen), offset_(0){};

    inline void bind_offset(int* off) { offset_ = off; };
    inline void offset(int* off) {offset_ = off;};
    int hdrlen_; // # of bytes for this header
    int* offset_; // offset for this header

public:
    TclObject* create(int argc, const char*const* argv){return 0;};
    virtual void bind { ... };
};
```

Q: What does create(...) do?
A: Create a shadow object

Mapping Class

- The constructor of class `PacketHeaderClass`

```
PacketHeaderClass(const char* classname, int hdrlen) :
    TclClass(classname), hdrlen_(hdrlen), offset_(0);{}
```

- Two input arguments:
  - `classname`: Feed it to class `TclClass`
  - `hdrlen`: Set `hdrlen_` to this value
- Also set `offset_` to NULL

Mapping Class

- An example: class CommonHeaderClass

```cpp
//~/ns/common/packet.cc
class CommonHeaderClass : public PacketHeaderClass {
public:
    CommonHeaderClass() : PacketHeaderClass("PacketHeader/Common", sizeof(hdr_cmn)) {
        bind_offset(&hdr_cmn::offset_);
    }
} class_cmnhdr;
```

Q: What does this line do?
- Bind this class to OTcl class "PacketHeader/Common"
- put the size of hdr_cmn in CommonHeaderClass::hdrlen_
Mapping Class

- **Class** CommonHeaderClass:

  ```cpp
  bind_offset(&hdr_cmn::offset_);
  ```

  ```cpp
  class PacketHeaderClass : public TclClass {
  protected:
    inline void bind_offset(int* off) { offset_ = off; }
    ...

  Q: Suppose &hdr_cmn::offset_ = 30. What would be the value of offset_?
  A: 0xd6f9c0

  struct hdr_cmn {
    ...
    static int offset_; // offset for this header
    ...
  };
```
Packet Header: Outline

- Protocol Specific Header Architecture
- Packet Type
- Protocol Specific Header
- Packet Header Access Mechanism
- Packet Header Manager
- Construction of Packet Header

Packet Header Access Mechanism

• PSH consists of three components

\[ \text{C++ Class} \quad \text{Mapping class} \quad \text{OTcl class} \]

\[ \text{hdr_cmn} \quad \text{CommonHeaderClass} \quad \text{PacketHeader/Common} \]

A structure to store packet attributes
Bind C++ and OTcl classes
OTcl Interface

• How do we access packet attributes?
  - Retrieve, and
  - Modify

Packet Header Access Mechanism

• **Two** step process:
  1. Retrieve the ref. to the C++ class
  2. Access attributes using the *structure* of the C++ class

• Let’s have a look at how a packet is create!

• When we create a packet, we need to initialize the packet, e.g., setting packet size.

**Q:** Which NS2 component is responsible to creating packets?

**A:**

**Q:** How does it create a packet?

**A:**

Packet Creation Example

• **Function allocpkt() of class Agent**

```cpp
//~/ns/common/agent.cc
Packet* Agent::allocpkt() {
    Packet* p = Packet::alloc();
    initpkt(p);
    return (p);
}
```

Packet Creation Example

Packet* Agent::initpkt(Packet* p) {
    hdr_cmn* ch = hdr_cmn::access(p);
    ch->uid() = uidcnt_++;
    ch->ptype() = type_;               ch->size() = size_;                ...
    hdr_ip* iph = hdr_ip::access(p); 1. Ref. Retrieval
    iph->saddr() = here_.addr_;        iph->sport() = here_.port_;        iph->daddr() = dst_.addr_;         iph->dport() = dst_.port_;        ...
    ...
    - Step 2 is quite straightforward.
    - Let's have a look at Step 1.
    1. Ref. Retrieval
    2. Attrib. access
    2. Attrib. access

Step 1: Retrieve a Ref. to PSH

```
hdr_cmn* ch = hdr_cmn::access(p);
inline static hdr_cmn* access(const Packet* p) {
    return (hdr_cmn*) p->access(offset_);
}
```

Packet Header: Outline

- Protocol Specific Header Architecture
- Packet Type
- Protocol Specific Header
- Packet Header Access Mechanism
- Packet Header Manager
- Construction of Packet Header
Packet Header Manager

• Responsibilities:
  - Keep the list of active protocols
  - Use at the initialization to create a packet format

• Related classes:

  - **C++ Class**
    - PacketHeaderManager

  - **Mapping class**
    - PacketHeaderManagerClass

  - **OTcl class**
    - PacketHeaderManager

Packet Header Manager

//~/ns/common/packet.cc

class PacketHeaderManager : public TclObject {
public:
    PacketHeaderManager() {bind("hdrlen ",
        &Packet::hdrlen_);}
};

Can you see the difference in this statement when compared to class TcpAgent?

static class PacketHeaderManagerClass : public TclClass {
public:
    PacketHeaderManagerClass() :
        TclClass("PacketHeaderManager") {}
    TclObject* create(int, const char*const*) {
        return (new PacketHeaderManager);
    }
} class_packethdr_mgr;

Packet Header Manager

- **OTcl class has two main variables**
  - `hdrlen_`: The length of the entire packet header
  - `tab_`: An associative array
    - **Index**: PSH
    - **Value**: 1 = active; N/A = inactive
Packet Header: Outline

- Protocol Specific Header Architecture
- Packet Type
- Protocol Specific Header
- Packet Header Access Mechanism
- Packet Header Manager
- Construction of Packet Header

Packet Construction

• **Three step process:**

1. **At the compilation**
   1.1 Construct mapping variables of PSH
   1.2 Construct mapping variables of the packet header manager
   1.3 Invoke `TclClass::bind()`
   1.4 Setting up active protocol list

2. **During the network configuration phase**
   ➔ Create the packet format

3. **During the simulation phase**
   ➔ Create packets

1. At the Compilation

- C++ and OTcl code translation \(\rightarrow\) EXE

**Task 1.1: Construct mapping variables of PSHs**

```cpp
class CommonHeaderClass : public PacketHeaderClass {
public:
    CommonHeaderClass() : PacketHeaderClass(
        "PacketHeader/Common", sizeof(hdr_cmn)) {
        bind_offset(&hdr_cmn::offset_);
    }
} class_cmnhdr;
```

1.1 Construct mapping variables

1. At the Compilation

1. At the Compilation

- **Consequence of Task 1.1**
  - Use example of common packet header
  - The mapping variable `class_cmnhdr` is created.
  - PSH length is stored in `class_cmnhdr::hdrlen`.
  - `class_cmnhdr::offset` points to the variable `hdr_cmn::offset`.

The Consequence of Tasks 1.1

1. At the Compilation

Task 1.2: Construct mapping variables of packet header manager

- The important step is

```cpp
// ~/ns/common/packet.cc
class PacketHeaderManager : public TclObject {
public:
    PacketHeaderManager() {bind("hdrlen_",
                        &Packet::hdrlen_);} }
```

- **Consequence:** the static variable `hdrlen_` is bound to the static variable `hdrlen_` of class `Packet`
The Consequence of Tasks 1.2

1. At the Compilation

Task 1.3: Execution of TclClass::bind()

- At NS2 invocation
- bind() is overridden by class PacketHeaderClass

```c++
//~/ns/common/packet.h
class PacketHeaderClass : public TclClass {
    ...
    virtual void bind()
    {
        TclClass::bind();
        Tcl& tcl = Tcl::instance();
        tcl.evalf("%s set hdrlen_ %d", classname_, hdrlen_);
        add_method("offset");
    }
};
```

1. At the Compilation

• Q: Suppose the length of the common packet header is 104 bytes. What does the following lines do?

```tcl
tcl.evalf("%s set hdrlen_ %d",
            classname_, hdrlen_);
```
The Consequence of Tasks 1.3

1. At the Compilation

Task 1.4: Setting up active protocol list

- **Sourcing file** `~ns/tcl/lib/ns-packet.tcl`

```tcl
PacketHeaderManager set hdrlen_ 0
foreach prot {
    Common
    IP
    ...
} {
    add-packet-header $prot
}
proc add-packet-header args {
    foreach cl $args {
        PacketHeaderManager set tab_(PacketHeader/$cl) 1
    }
}
```

Packet Construction

• Three step process:
1. At the compilation
   1.1 Construct mapping variables of PSH
   1.2 Construct mapping variables of the packet header manager
   1.3 Invoke `TclClass::bind()`
   1.4 Setting up active protocol list
2. During the network configuration phase
   ➔ Create the packet format
3. During the simulation phase
   ➔ Create packets

2. During the Network Configuration Phase

- Prior to "\$ns run"
- Main task: Set the offset for all active PSH.
- Q: How do NS2 know which one is active?
  A:
  - During a creation of the Simulator object
  - Invoke instproc `create_packetformat` of class `Simulator`.

2. During the Network Configuration Phase

```tcl
//~ns/tcl/lib/ns-packet.tcl
Simulator instproc create_packetformat { } { 
    PacketHeaderManager instvar tab_
    set pm [new PacketHeaderManager]
    foreach cl [PacketHeader info subclass] { 
        if [info exists tab_($cl)] { 
            set off [$pm allochdr $cl] 
            $cl offset $off 
        } 
    } 
    $self set packetManager_ $pm
}
```

Q: `tab_` is an instvar of ??
A: PacketHeaderManager
Packet Header Manager

- **OTcl class has two main variables**
- **hdrlen**: The length of the entire packet header
- **tab**: An associative array
  - Index: PSH
  - Value: 1 = active; N/A = inactive

2. During the Network Configuration Phase

```tcl
//~ns/tcl/lib/ns-packet.tcl
Simulator instproc create_packetformat { } {
    PacketHeaderManager instvar tab_
    set pm [new PacketHeaderManager]
    foreach cl [PacketHeader info subclass] {
        if [info exists tab_($cl)] {
            set off [($pm alloc hdr $cl]
            $cl offset $off
        }
    }
    $self set packetManager_ $pm
}
```

2. During the Network Configuration Phase

```
//~ns/tcl/lib/ns-packet.tcl
1 PacketHeaderManager instproc allochdr cl { 
2 set size [cl set hdrlen_]
3 $self instvar hdrlen_ 
4 set NS_ALIGN 8 
5 set incr [expr ($size + ($NS_ALIGN-1)) & ~(NS_ALIGN-1)]
6 set base $hdrlen_ 
7 incr hdrlen_ $incr 
8 return $base
9 }
```

Exam: What does this line do?
2. During the Network Configuration Phase

Packet Construction

• Three step process:
  1. At the compilation
     1.1 Construct mapping variables of PSH
     1.2 Construct mapping variables of the packet header manager
     1.3 Invoke `TclClass::bind()`
     1.4 Setting up active protocol list
  2. During the network configuration phase
     ➔ Create the packet format
  3. During the simulation phase
     ➔ Create packets

3. During the Simulation Phase

- Packet creation
- Use the format defined in the former two steps
- Most common one: class Agent
  - Packet creation: `Packet* p = allocpkt();`
  - Packet initialization: `initpkt(p)`
Outline

• Overview
• Packet Allocation and Deallocation
• Packet Header
• Data Payload
• Customizing Packets
• Summary

Data Payload

- Generally not being used
- Main classes:
  - AppDataType
  - AppData
  - PacketData
- Main files
  - ~ns/common/ns-process.h
  - ~ns/common/packet.h
- See Section 8.4 of the Text

Defining a New Packet Header

- Define data payload → not recommended
- Store necessary information in the packet header.
- How? → Ex: My Header

C++ Class: hdr_my
Mapping class: MyHeaderClass
OTcl class: PacketHeader/My

Defining a New Packet Header

• 4 main steps:

1. Define a PSH $C++$ class `hdr_my`
   - Define the structure to hold packet attributes
   - Declare variable `offset_`
   - Define function `access(p)`
   - Define new packet type (if necessary) in `enum packet_t` and `p_info`

2. Define a PSH $OTcl$ class `PacketHeader/My`

Defining a New Packet Header

• 4 main steps:

3. Define a mapping C++ class \texttt{hdr\_my}
   - Derive class \texttt{MyHeaderClass} from class \texttt{PacketHeaderClass}
   - In the constructor,
     • Feed the following two parameters as the input
       1. OTcl class name: \texttt{PacketHeader/My}
       2. Size of the my header: \texttt{sizeof(hdr\_my)}
     • Execute function \texttt{bind\_offset(&hdr\_my::offset\_)}
   - Instantiate an instance \texttt{class\_my} at the construction
Defining a New Packet Header

• 4 main steps:

4. Activate my header using `PacketHeaderManager`

```plaintext
PacketHeaderManager set hdrLen_ 0
foreach prot {
    Common
    My
    ...
} {
    add-packet-header $prot
}
```

• Put only the suffix `xxx` (in `PacketHeader/xxx`)
Outline

• Overview
• Packet Allocation and Deallocation
• Packet Header
• Data Payload
• Customizing Packets
• Summary

Summary

• In most case, NO packet is actually send.  Put info. in the packet header and send it.

• A packet (class Packet) consists of
  - Packet header (bits_): Info. about packet
  - Data payload (data_): Actual payload

• Packet allocation
  - Take one from a free packet list, or
  - Create a new one

• Packet deallocation
  - Put the packet in the free packet list.

Summary

• Packet header:
  - Variable = *bits_*
  - Contains several Protocol specific header (PSH)
  - How would you access a PSH? ( )

• 3 Main parts of PSH
  - A C++ class: hdr_cmn
  - An OTcl class: PacketHeader/Common
  - A mapping class: CommonHeaderClass

Summary

• Packet creation process
  1. At the compilation:
     • Construction of PSH and packet header mapping variables,
     • invoke TclClass::bind(), and
     • set up active protocol list
  2. During the network configuration phase: Create packet format
  3. During the simulation: Create packets

• Defining a new packet header
  1. Define a PSH C++ class
  2. Define a PSH OTcl class
  3. Define a PSH mapping class
  4. Activate the header