Custom Packets and Project II

ECE544 Communication Networks II

Sumit Maheshwari

02/22/2019

Includes teaching material from Bart Braem, Michael Voorhaen and previous TA Francesco Bronzino
Packet Formats

• Packet formats == structs
  • structs are a typical C concept, very low level
  • tempting to improve this by wrapping the packets in objects
  • attractive to create packet factories

• Do not do this (large overhead):
  • In terms of memory and computation (allocate objects, create and delete objects)

• Use the plain structs
  • Requires getting used to
  • Straightforward: most packet manipulation is low-level anyway
Packet Formats Example

• Define the packet header
  ```c
  struct MyPacketFormat{
    unit8_t type; // 8 bit = 1 byte
    unit32_t lifetime; // 32 bit = 4 bytes
    in_addr destination; // IP address
  };
  ```

• Cast a packet to access the header
  ```c
  MyPacketFormat* format = (MyPacketFormat*)packet->data();
  format->type = 0;
  format->lifetime = htonl(counter);
  format->destination = ip.in_addr();
  ```

More at: /home/comnetsii/click/elements/local
Timer Based Actions

class MyPacketGen : public Element {
public:
    MyPacketGen();
    ~MyPacketGen();

    const char *class_name() const { return "MyPacketGen"; }
    const char *port_count() const { return "0/1"; }
    const char *processing() const { return PUSH; }

    int configure(Vector<String> &, ErrorHandler *);
    void push(int, Packet *);
    int initialize(ErrorHandler *errh);
    void run_timer(Timer *timer);
    void sendRequest();
    void sendResponse();
    void sendUpdate();

private:
    Timer _timer;
    int current_loop;
};

More at:
/home/comnetsii/click/elements/local
Run Timer

```cpp
int MyPacketGen::initialize(ErrorHandler *) {
    timer.initialize(this); // Initialize timer object (mandatory).
    timer.schedule_after_sec(2);
    return 0;
}

void MyPacketGen::run_timer(Timer *timer) {
    // This function is called when the timer fires.
    Timestamp now = Timestamp::now();
    click_chatter("%s: %{timestamp}: timer fired!\n", 
                   declaration().c_str(), &now);
    if(current_loop == 0) {
        this->sendRequest();
    } else if (current_loop == 1) {
        this->sendResponse();
    } else {
        this->sendUpdate();
    }
    current_loop++;
    if(current_loop < 3 ){
        timer.reschedule_after_sec(2);
    }
}
```

More at: /home/comnetsii/click/elements/local
Playground 2(A): The Middlebox

- Write an element that changes the content of every packet into another (original content is lost)

- The new content should be configurable from the configuration script using your element e.g. ContentChanger(“Middlebox changed this content!”);

- Feel free to add more functionalities to your elements like minLength, maxLength etc.

Hints:
- Think of push/pull port requirements. Can only push or only pull work?
- Refer to existing elements’ implementations in click/elements/ directory
Playground 2(A): The Middlebox

- Use provided script to create 4 virtual interfaces
  
  ```bash
  comnetsii@comnetsII:~/tools$ sudo createNet2
  ```

- At R1, generate packets with the payload: "Hello"
- At R2, use your implemented element to change the payload to any custom string you like.
- At R3, print the new payload and discard the packet
- Use the RouterPort (modified) abstraction to send to your network

- Will you use encapsulations from Project1?
- How many RouterPorts are needed?
Playground 2(B): Customized Packet

- Write an element which continuously generates a **12 Bytes, 4 fields custom packet every second**:
  - Type: 1 Byte, Value: 0
  - Sequence 1 Byte, Value: *current packet sequence number*
  - Length: 2 Bytes, Value: 12
  - Payload: 8 Bytes, Value: ‘olleh’

- Send this packet from R1 as in PG1
- R2 should just forward it to the next hop without changing the content
- R3 should receive the packet and print the content

Will the packet sent out from R1 be of 12 Bytes?
Submission Info

• Due:
  • Playground2: March 1st (before class)

• Submission instructions:
  • Submit a single archive (zip or tar.gz) to sumitm@winlab.rutgers.edu with subject “ECE544 P2PG2”
  • Include in the archive 1 folder named “PG2”.
  • Include in the folder the modified all the configuration files and the implemented elements
  • Write a README file for your design if necessary
  • If you have written additional test configuration file(s), feel free to send
  • Make sure your code is readable – use commenting!

• Technical questions: use the mailing list. It is better for all of us.
  • comnet2@winlab.rutgers.edu