Project 2

ECE544 Communication Networks II

Francesco Bronzino

Includes teaching material from Bart Braem and Michael Voorhaen

RUTGERS

WINLAB

Project Goals

- Write custom elements
- Design and implement basic network protocols
- Get familiar with the framework used in the final project

Writing Custom Elements: Element Header

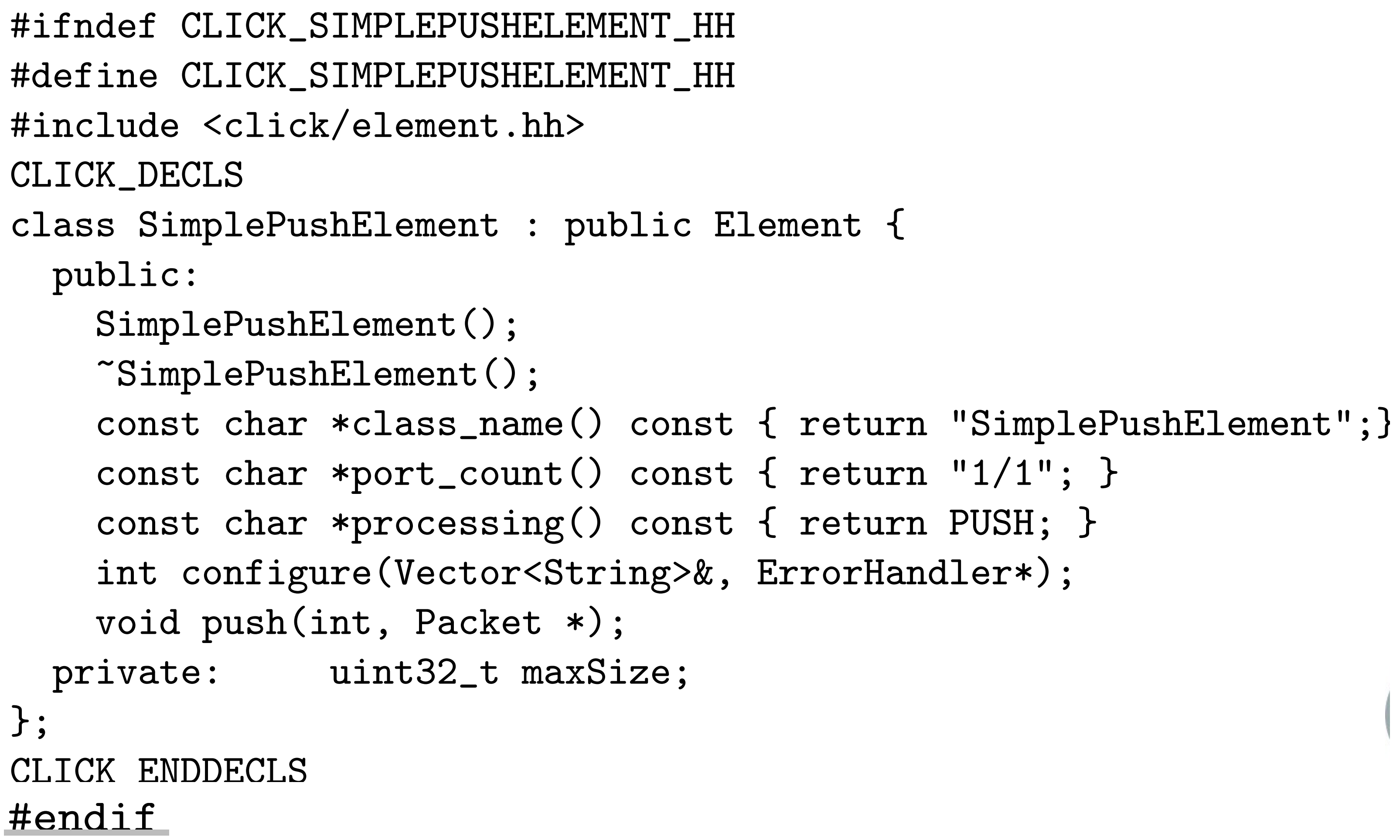
- Necessary in the header:
  - Include-guard macros
  - Click element macros
  - Include click/element.hh
  - The class declaration containing 3 special methods:

```
const char *class_name() const
const char *port_count() const
const char *processing() const
```

Writing Custom Elements: Element Header

- Necessary in the source file:
  - Include click/config.hh first!
  - CLICK_DECLS macro
  - CLICK_ENDDECLS macro
  - EXPORT_ELEMENT macro
  - Implementations of the methods

Writing Custom Elements: SimplePushElement.hh



```
#ifndef CLICK_SIMPLEPUSHELEMENT_HH
#define CLICK_SIMPLEPUSHELEMENT_HH
#include <click/element.hh>
CLICK_DECLS
class SimplePushElement : public Element {
  public:
    SimplePushElement();
    ~SimplePushElement();
    const char *class_name() const { return "SimplePushElement";}
    const char *port_count() const { return "1/1"; }
    const char *processing() const { return PUSH; }
    int configure(Vector<String>&, ErrorHandler*);
    void push(int, Packet *);
  private:     uint32_t maxSize;
};
CLICK ENDDECLS
#endif
```

Writing Custom Elements: SimplePushElement.cc

```
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include "simplepushelement.hh"
CLICK_DECLS
SimplePushElement::SimplePushElement(){}
SimplePushElement::~SimplePushElement(){}

int SimplePushElement::configure(Vector<String> &conf, ErrorHandler
        *errh) {
  if (cp_va_kparse(conf, this, errh, "MAXPACKETSIZE", cpkM, cpInteger,
        &maxSize, cpEnd) < 0) return -1;
  if (maxSize <= 0) return errh->error("maxsize should be larger than 0");
  return 0;
}
```

Writing Custom Elements: SimplePushElement.cc

```
void SimplePushElement::push(int, Packet *p){
  click_chatter("Got a packet of size %d",p->length());
  if (p->length() > maxSize)  p->kill();
  else output(0).push(p);
}
CLICK_ENDDECLS
EXPORT_ELEMENT(SimplePushElement)
```

Writing Custom Elements

- Similarly you can define pull (needs to implement pull operation) and agnostic elements (needs to implement both push and pull operations)
- const char *port_count() const has to return the number of ports your element will have (it can be a flexible number, see examples)

Compile the New Elements

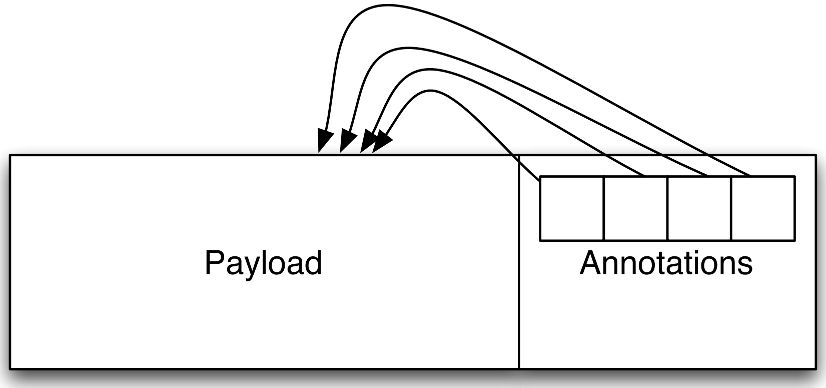
- All elements are stored in /elements/ directory
  - Yours should be put in elements/local
  - Put the .hh and .cc files there
- Go to the base click folder
- To make those elements available:
  - sudo ./configure --enable-local --disable-linuxmodule
    - Required only the first time (or any time after cleaning the folder)
  - make elemlist
  - make
- Notice new elements being compiled, solve any compilation problems and use your elements

Writing Custom Elements

- We are just scratching the surface…
- For more information:
  - Go through the following coding tutorial: http://www.pats.ua.ac.be/software/click/click-2.0/coding.pdf
  - Classes available for element creation: read.cs.ucla.edu/click/doxygen/classes.html
  - Dr Kohler thesis: http://www.read.cs.ucla.edu/click/

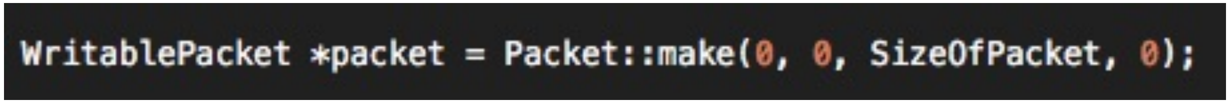
Packets

- Packet consists of payload and annotations, payload:
  - raw bytes (char*)
  - Access with struct*
- Annotations: metadata to simplify processing, "post-its"
  - E.g. start of IP header or TCP header
  - Paint annotations
  - User defined annotations



Packets

- Packets are created using a specific method part of the class *Packet*
  - *make*
- Creating a new packet within a custom element:



```
WritablePacket *packet = Packet::make(0, 0, SizeOfPacket, 0);
```

- More details at: read.cs.ucla.edu/click/doxygen/class_packet.html

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, very large overhead:
  - In terms of memory and computation (allocate objects, create and delete objects)
  - In terms of code base
- Use the plain structs
  - Requires getting used to
  - Straightforward: most packet manipulation is low-level anyway

Packet Formats Example

- Define the packet header

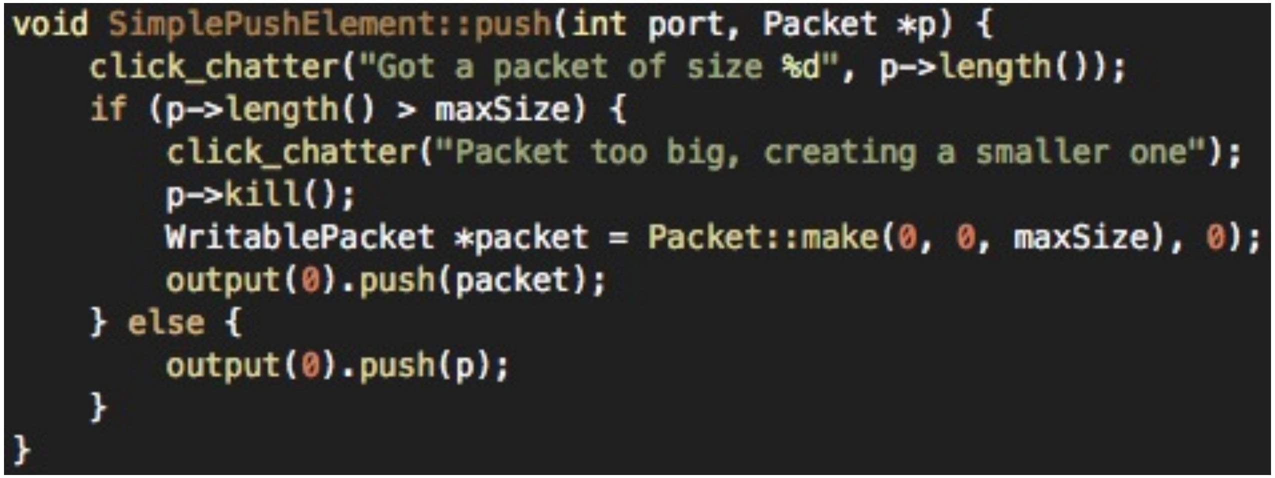
```
struct MyPacketFormat{
  uint8_t type; // 8 bit = 1 byte
  uint32_t lifetime; // 32 bit = 4 bytes
  in_addr destination; // IP address
};
```

- Cast a packet to access the header

```
MyPacketFormat* format=(MyPacketFormat*)packet->data();
format->type = 0;
format->lifetime = htonl(counter);
format->destination = ip.in_addr();
```

When to Create Packets?

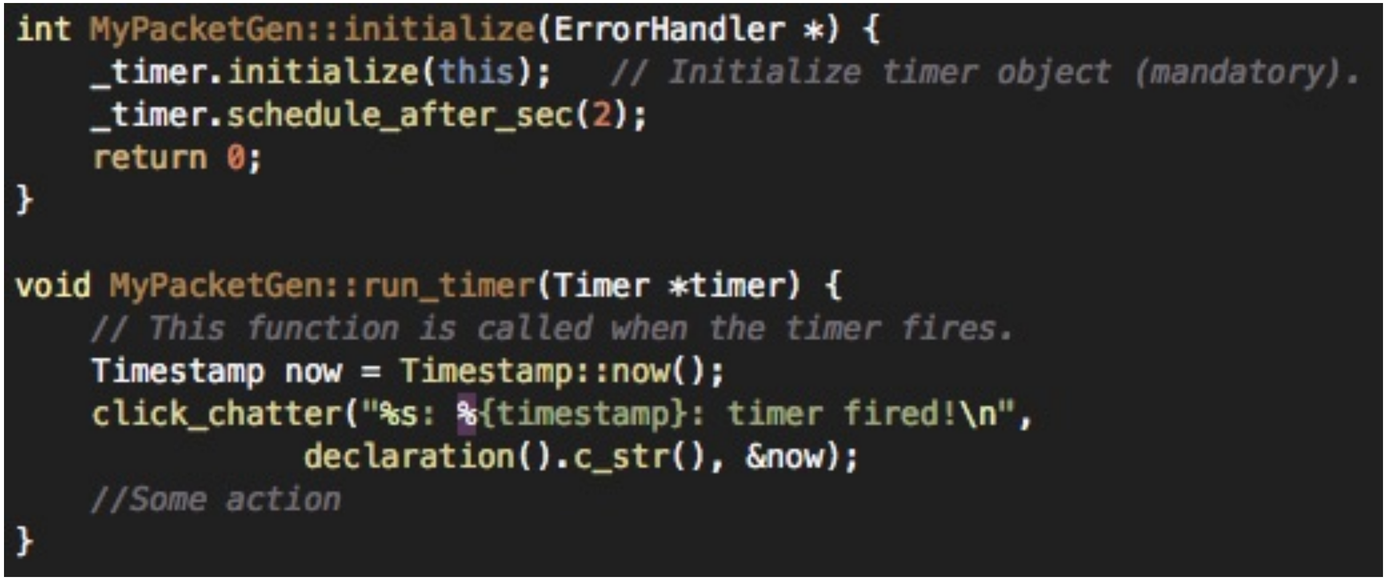
- Until now you only created packets either using a *Source* element or by reading them from a *FromDevice* element
- Option 1: creating them in the middle of the processing pipeline



```
void SimplePushElement::push(int port, Packet *p) {
    click_chatter("Got a packet of size %d", p->length());
    if (p->length() > maxSize) {
        click_chatter("Packet too big, creating a smaller one");
        p->kill();
        WritablePacket *packet = Packet::make(0, 0, maxSize), 0);
        output(0).push(packet);
    } else {
        output(0).push(p);
    }
}
```

When to Create Packets?

- Until now you only created packets either using a *Source* element or by reading them from a *FromDevice* element
- Option 2: timed or periodic creation.



```
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);
    //Some action
}
```

Timers

- Class used to create timed operations.
- Runs the *run_timer* function upon expiry

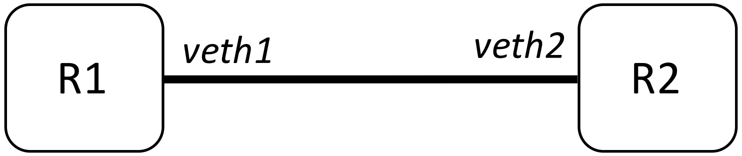
```
class MyElement: public Element {
  public:
    void run_timer(Timer*);
  private:
    Timer timer;
}
MyElement::MyElement(): timer(this){}
int MyElement::configure(Vector<String> &conf, ErrorHandler *errh){
  timer.initialize(this);
  timer.schedule_after_msec(1000);
  return 0;
}
```

Compound Elements

- Group elements in larger elements
- Configuration with variables
- Pass configuration to the internal elements, can be anything (constant, integer, elements, IP address, ...)
- Motivates reuse
- No need to use in these projects, but you will be using one indirectly (more on this later)

Hands On With Our Framework

- To simplify your life, we will provide you with an abstracted concept of router port.
- This will allow you to implement your own protocols on top of the click framework.
- You already got briefly introduced to some of these tools:
  - Remember the createNet1 script?
- This creates a pair of linked interfaces (veth1 and veth2).

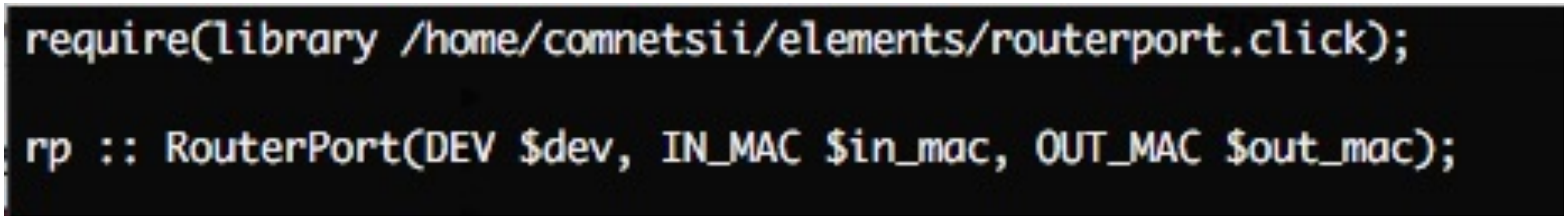


Hands On With Our Framework

- *Port abstraction*: defines one end of a link
- Everything that gets into veth1 arrives unchanged to veth2
- Abstraction obtained through the provided element:
  - elements/routerport.click
- At the beginning of your configuration file:
  - require(library /home/comnetsii/elements/routerport.click);
- *RouterPort* is a push element with one input and one output port

Hands On With Our Framework

- RouterPort takes 3 parameters: device name, local mac, remote mac
- Example:
  - Element that sends every one second a hello message into the port
  - Prints all packets received and discard them

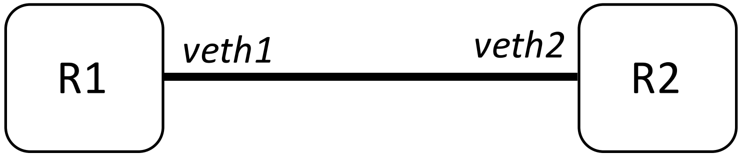


```
require(library /home/comnetsii/elements/routerport.click);

rp :: RouterPort(DEV $dev, IN_MAC $in_mac, OUT_MAC $out_mac);
```

Hands On With Our Framework

- Generate a small network of two routers
  - $ createNet1



- Exchange packets between routers
  - Start two click instances using the example found in:
    - examples/router/printer.click
  - Make sure to set the 3 parameters appropriately given the generated interfaces
  - E.g.:
    - $ sudo ~/click/userlevel/click printer.click dev=veth1 in_mac=08:00:27:9a:04:e5 out_mac=08:00:27:3e:0b:11

Exercise 1

- Generate a small network of two routers
  - $ createNet1
- Router one one side:
  - Define a custom packet with 2 header fields: one of 1 Byte for the type and one of 4 Bytes expressing the length of the payload
  - Write an element every 3 seconds create such packet and assign either 0 or 1 to the type field (payload size can be constant)
  - Use the *RouterPort* elements to push the packet into one device
- Router on other side:
  - Read the packets from a *RouterPort*
  - Print the content of the packet

Exercise 1

- Hints:
  - You will submit 2 click configuration files plus at least one *cc* file and two *hh* files
  - The second router will look almost like the one from project one, but now you are using *RouterPort*

Exercise 2

- Generate a small network of two routers
  - $ createNet1
- Router one one side:
  - Define a custom packet with 2 header fields: one of 1 Byte for the type and one of 4 Bytes expressing the length of the payload
  - Write an element every 3 seconds create such packet and assign either 0 or 1 to the type field (payload size can be constant)
  - Use the *RouterPort* elements to push the packet into one device
  - Use the same *RouterPort* to read packets from the interface
  - Print the content of the packet

Exercise 2

- Router on other side:
  - Read the packets from a *RouterPort*
  - Write a custom element that reads the field of the packet. If the value is 0 push out of port 0, if it is 1 push out of port 1
  - Packets coming out of port 1 are dropped, packets coming out of port 0 are sent back into *RouterPort*
- Hints:
  - You will submit 2 click configuration files plus at least two *cc* file and three *hh* files
  - Most things will look similar to exercise 1

General Info

- Due: March 11th
- Submission instructions:
  - Submit a single archive (zip or tar.gz) to bronzino@winlab.rutgers.edu with subject “ECE544 Project 2”
  - Include in the archive 3 folders named “exercise1”, “exercise2”, “exercise3”. They should contain only files that you implemented (i.e. click configuration files, new elements and applications).
  - **Do not** include the whole click resources or binary files!