Project 2

ECE544 Communication Networks II
Francesco Bronzino

Includes teaching material from Bart Braem and Michael Voorhaen
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:
Writing Custom Elements: Element Header

• Necessary in the source file:
  • Include click/config.hh first!
  • CLICK_DECLS macro
  • CLICK_END_DECLS macro
  • EXPORT_ELEMENT macro
  • Implementations of the methods
Writing Custom Elements: SimplePushElement.hh

```c++
#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_END_DECLS
#endif
```

Bart Braem, Michael Voorhaen
Click Coding
6/57
Writing Custom Elements:
SimplePushElement.cc

```c++
#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;
}
```
```cpp
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_END_DECLS
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)
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
  • Dr Kohler thesis: http://www.read.cs.ucla.edu/click/
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

```c
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

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

• Notice new elements being compiled, solve any compilation problems and use your elements
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 5 parameters: device name, local ip, remote ip, 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/comnets2/elements/routerport.click);

rp :: RouterPort(DEV $dev, IN_ADDR $in_addr, OUT_ADDR $out_addr, IN_PORT $in_port, OUT_PORT $out_port, IN_MAC $in_mac, OUT_MAC $out_mac);

RatedSource(DATA "hello", RATE 1) -> rp;

rp -> Print(Received, MAXLENGTH -1) -> Discard;```
Hands On With Our Framework

• Important note: all our scripts, generate pair of interfaces belonging to the same subnet
• Connect only interfaces on the same subnet
• Feel free to look at what is used in the compound element
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 5 parameters appropriately given the generated interfaces
  - E.g.:
    - `sudo ~/click/userlevel/click printer.click dev=veth1 in_addr=192.168.1.1 out_addr=192.168.1.2 in_port=10000 out_port=10001 in_mac=08:00:27:9a:04:e5 out_mac=08:00:27:3e:0b:11`
Hands On With Our Framework

- Writing end-host applications.
- Download package:  
  [link](http://www.winlab.rutgers.edu/comnet2/Projects/project2.tar)
- The package provides a similar port concept for developing at the application layer
- Examples on how to use it:  
  - [link](http://www.winlab.rutgers.edu/comnet2/Projects/project_index.html)
Exercise 1

- Write an element that change the content of every packet into another
- The new content should be configurable from the configuration script
- Use the RouterPort elements to build your network
Exercise 1

• Use provided script to create 4 virtual interfaces
  • Run: $ createNet2

• Obtained topology:

```
R1  veth1  veth2  veth3  veth4  R3
    ^       ^       ^       ^
    |       |       |       |
    veth7  veth8  veth9  veth10
```

• Tips:
  • You can reuse previous exercises to implement R1 and R3
  • You **have** to implement your own element to change the content in R2
Exercise 2

- Design a protocol to transport a file from one end of a communication link to the other over an unreliable link.
- Requires fragmentation and reassembly.
- Choose the ARQ protocol you prefer (stop&wait, go-back N, selective ACK, etc.) or design your own. The design with better performance will receive extra credits.
- Use the program diff to check
Exercise 2

• Use the application layer package to implement the end host applications.
• Implement one sender and one receiver
Exercise 2

• Use provided script to create 4 virtual interfaces
  • Run: $ createNet2

• Obtained topology:

![Network Diagram]

• Tips:
  • R only needs to forward packets. (i.e. you only need 2 elements, the 2 RouterPorts)
Exercise 3

• Replace router R from the previous exercise with a new implementation that uses the LossyRouterPort element

• Tips:
  • You can find the new element in the same folder as the previous one.
Exercise 3

• Write a small README file that describes the your implemented protocol and its performance.
• It should include (but not limited to):
  • Your designed protocol
  • Packet and signaling formats
  • Purpose and functions of each new class/element you implemented
  • Performance results (no need for complex graphs, just an analysis on the performance based on protocol characterization and experiments)
General Info

• Due: March 27\textsuperscript{th}

• 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).
  • \textbf{Do not} include the whole click resources or binary files!