Compound Elements and Project II

ECE544 Communication Networks II

Sumit Maheshwari

02/15/2019

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

- Click router is assembled from packet processing modules called elements
  - Individual elements implement simple router functions such as encapsulation, annotation, queueing etc.
- Click configuration is a directed graph of elements where packets follow the edges of the graph
- Configurations are written in a declarative language that supports user-defined abstractions
- Click configurations can be compounded and elements can added (modular)
Project Goals

• Understanding compound elements
• Passing multiple arguments to configuration files
  • Configuration using files
• Writing custom elements
Compound Elements

• Group elements in larger element
• Configuration with variables
• Pass configuration to the internal elements
  • can be anything (constant, integer, elements, IP address, …)
• Motivates reuse (same as modules in a project)
Hands on with our framework

• To simplify your life, we will provide you 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 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 `routerport.click`
  - `$ 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
RouterPort.click

RouterPort takes up to 7 parameters: device name, local ip, remote ip, local port, remote port, local mac, remote mac

```c
/* Compound class that provide a port abstraction for a router */

elementclass RouterPort { DEV $dev, IN_MAC $in_mac, OUT_MAC $out_mac ;

//Add filtering based on ips and macs
out_dev :: ToDevice($dev)
in_dev :: FromDevice($dev)

//in and out queues
in_queue::Queue();
out_queue_data::Queue();

input -> EtherEncap(0x0800, $in_mac, $out_mac)
    -> out_queue_data
    -> out_dev

//missing the arp part
in_dev -> in_queue
    -> Unqueue
    -> HostEtherFilter($in_mac, DROP OWN true) // check that the mac address
is proper
    -> Strip(14)
    -> output
};
```
Hands on with our framework

• Example (/home/comnetsii/examples/router/printer.click):
  • Element that sends every one second a hello message into the port
  • Prints all packets received and discard them

```javascript
require(library /home/comnetsii/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;
```

• Note: all our scripts generate pair of interfaces belonging to the same subnet
• Connect only interfaces on the same subnet
• Feel free to look into examples and elements directory.
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:

```cpp
const char *class_name() const { return "MyClassifier"; }
const char *port_count() const { return "1/3"; }
const char *processing() const { return PUSH; }
```
Writing custom elements: Element Source

• Necessary in the source file:
  • Include click/config.hh
  • CLICK_DECLS macro
  • CLICK_END_DECLS macro
  • EXPORT_ELEMENT macro
  • Implementation of the methods

• Recommended:
  • Understand the element MyClassifier header and source file at (.hh and .cc):
    comnetsii@comnetsII:~/click/elements/local$
Writing custom elements: SimplePushElement

```cpp
#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: unit32_t maxSize; }

CLICK_END_DECLS
#endif
```

/home/comnetsii/click/elements/local/simplepushelem.hh
Writing custom elements: SimplePushElement

```c
#include <click/config.h>
#include <click/confparse.hh>
#include <click/error.hh>
#include "SimplePushElement.hh"
CLICK_DECLS

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;
}
```
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)
Compile the New Elements

- All elements are stored in $click/elements/ directory
  - Yours should be kept in $click/elements/local/
  - Put the .hh and .cc files there
- Go to the base click folder $click
- To make those elements available:
  - $make elemlist
  - $make
- Notice new elements being compiled
- Solve any compilation problems
- Your elements are ready to be used in the configuration files!

Find these files at: $click/elements/local
simplepushelem.hh
simplepushelem.cc

Try in sample configuration file:
RatedSource("Hello")
->SimplePushElement(MAXPACKETSIZE 4)
->Print
->Discard;

Try changing the arguments!
Playground 1(A): The Port Abstraction

- Generate a small network of two routers:
  
  `comnetsii@comnetsII:~/tools$ sudo createNet1`

- Exchange packets between routers:
  
  - Start two click instances (two terminals) using the example found in:
    
    `examples/router/printer.click`
  
  - Make sure to set the parameters (arguments to configuration file) appropriately.
    
    Example – for first click:
    
    `comnetsii@comnetsII:~/examples/router$ sudo ~/click/userlevel/click printer.click dev=veth2 in_addr=192.168.1.1 out_addr=192.168.1.2 in_port=10001 out_port=10002 in_mac=fe:c6:3b:4e:ce:90 out_mac=da:0f:42:8a:6e:8c`
  
  - Similarly modify the command for the second click instance.
Playground 1(A): The Port Abstraction

• At this point, you will get an error message for inability of RouterPort to handle the arguments passed.
• Modify routerport.click without modifying printer.click (no make or compile required)
• Run both the click instances and you should see packets being received at both the terminals.
Playground 1(B): Packet Size Matters

- Currently, SimplePushElement kills a packet which is more than `maxSize`

- Modify simplepushelem .hh and .cc files at `$/home/comnetsii/click/elements/local/` to also kill a packet which is less than `minSize`

- Compile the modified elements (`$make elemlist` and `$make`)

- Write the following configuration files to test the added function:
  - `min.click`: Input packet is “Hello”, `minSize` is 8 Bytes and `maxSize` is 20 Bytes
  - `ok.click`: Input packet is “Hello there”, `minSize` is 8 Bytes and `maxSize` is 20 Bytes
  - `max.click`: Input packet is “Hello, How are you doing?”, `minSize` is 8 Bytes and `maxSize` is 20 Bytes
Submission Info

- Due:
  - Playground1: Feb 22\(^{th}\) \textit{(before class)}
- Technical questions: use the mailing list. It is better for all of us.
  - comnet2@winlab.rutgers.edu
- Submission instructions:
  - Submit a single archive (zip or tar.gz) to sumitm@winlab.rutgers.edu with subject “ECE544 P2PG1”
  - Include in the archive 1 folder named “PG1”.
  - Include in the folder the modified routerport.click file, modified element’s header and source files, and configuration files
  - If you have written additional test configuration file(s), feel free to send
  - Make sure your code is readable – use commenting!
  - Write a README file if there is anything you think we should know
  - Do not submit the old and unmodified files
  - For the coming exercises, explore source code of elements in the directory: $click/elements/