The Click Modular Router
Intro

- The “old way”
- Routers
- Large companies dominated the networking market
- Routers sold as entire box
- Software is proprietary
  - Few standards between companies
- Need a new feature?
Motivation

- The “old way”
- Network innovation is slow
  - But tech is moving fast
- Expect more from the network
- Simple elements are now complex
  - e.g. NAT, vlan, firewall, etc.
Click Modular Router

- **Flexible**
  - Powerful enough to solve existing demands
  - Easy to add new features
- **Modular**
  - Easy to reuse existing tools/code
  - Simply compose existing tools
- **Open**
  - Allow for researchers to extend/experiment

---

The Click Abstraction

1. Elements
2. Configuration
3. Method Interfaces
4. Ports
The Click Abstraction - Elements

- Elements are the building blocks
- Provide the computing power
- Details and complexity are here
- Code that gets run for each packet
- Think of these as objects in OOD

Examples: IPClassifier, CheckTCPHeader, Counter, Tee...
Reference: [http://read.cs.ucla.edu/click/elements](http://read.cs.ucla.edu/click/elements)
The Click Abstraction

1. Elements
2. Configuration
3. Method Interfaces
4. Ports
The Click Abstraction - Config

- Need to instantiate the elements
- Provide arguments for each element
  - Allows for more flexibility and configuration of each element
- Fine-tunes the behavior of the element at a per instance basis

Example: IPClassifier( tcp, udp, icmp, - )
The Click Abstraction

1. Elements
2. Configuration
3. Method Interfaces
4. Ports
The Click Abstraction - Method Intf.

- Allows for communication between elements
- Think of these as methods in OOD

Example: Queue.count
The Click Abstraction

1. Elements
2. Configuration
3. Method Interfaces
4. Ports
The Click Abstraction - Ports

- Need to compose elements
- Packet handoffs
  - Packet leaves one element and proceeds to next element
  - Inspired by traditional routers
- Packets enter and leave through ports
- Push vs. Pull ports
The Click Abstraction - Ports

Push Port
- Initiated by source end
- Most connections
- Represented by black ports

Pull Port
- Initiated by destination end
- Useful to control timing of packet
- Represented by white ports

Cannot combine push part and pull port!
Putting it all together

Element

Push Port

Configuration String

Agnostic Ports

FromDevice(eth0) → Counter → Discard
Setting up Click VM

1. Install VM software (VirtualBox, VMWare)
2. Download and import appliance:
   http://www.cs.duke.edu/courses/fall15/compsci514/ClickClassVM.ova
   Note: ~2.5 GB so it may take a few minutes to download
3. Verify your set-up:
   In a terminal run “~/bin/click ~/click/conf/test.click”
4. Download packet analysis tool - wireshark
   “sudo apt-get install wireshark”

If you have any questions, see me after class/send me an email
Click Configurations

- Configuration describe network box
- Combine the elements we’ve learned about

http://www.read.cs.ucla.edu/click/tutorial1
Click Configurations

- Wireshark can capture and view packet traces
- Reading in packets, and sending them out
  - `FromDump('<trace file>'), FromDevice('<interface>')`
  - `ToDump('<trace file>'), ToDevice('<interface>')`

Examples:
- `FromDump(packets.dump) -> ...`
- `FromDevice(eth0) -> ...`
Click Configurations

- Chaining elements elements
  - FromDump(...) -> IPClassifier(...)  
  - Implicit output port 0 to input port 0
Click Configurations

- Creating variable
  - `ipc::IPClassifier(tcp, udp, -)`
  - Can now refer to this element as `ip`
  - Useful when there are multiple input/output ports, or for clarity
  - e.g. `ipc[1]` refers to output 1 (udp) of the element

Port 0 = `ipc[0]`, all tcp traffic
Port 1 = `ipc[1]`, all udp traffic
Port 2 = `ipc[2]`, all other traffic
Click Configurations

- **Problem:**
  - Input: f1a.dump
  - Output: see table

- **Tools:**
  - RadixIPLookup
  - IPClассifier (Be careful with order. Filters are checked in order)

<table>
<thead>
<tr>
<th>Destination</th>
<th>File Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>131.0.0.0/8</td>
<td>f2b.dump</td>
</tr>
<tr>
<td>131.179.0.0/16</td>
<td>f2c.dump</td>
</tr>
<tr>
<td>18.0.0.0/8</td>
<td>f2d.dump</td>
</tr>
<tr>
<td>Anything else (-)</td>
<td>f2e.dump</td>
</tr>
</tbody>
</table>
Solutions

FromDump(f2a.dump, STOP true)
    -> r :: RadixIPLookup(131.0.0.0/8 0, 131.179.0.0/16 1, 18.0.0.0/8 2, 0/0 3)
    r[0] -> ToDump(f2b.dump, ENCAP IP)
    r[1] -> ToDump(f2c.dump, ENCAP IP)
    r[2] -> ToDump(f2d.dump, ENCAP IP)
    r[3] -> ToDump(f2e.dump, ENCAP IP)

FromDump(f2a.dump, STOP true)
    -> ip :: IPClассifier(dst 131.179.0.0/16, dst 131.0.0.0/8, dst 18.0.0.0/8, -)
        // 131.179.0.0/16 must come first because filters are checked in order
    ip[0] -> ToDump(f2c.dump, ENCAP IP)
    ip[1] -> ToDump(f2b.dump, ENCAP IP)
    ip[2] -> ToDump(f2d.dump, ENCAP IP)
    ip[3] -> ToDump(f2e.dump, ENCAP IP)
Click Configurations

- Creating a compound element
  - Combination of other smaller elements
  - Use this when other elements provide enough functionality, and you want to treat them as a black box
  - Similar to class, superclass in OOD

```cpp
class MyCompoundTCPChecker{
    input->CheckIPHeader()->IPClassifier(tcp,-)->CheckTCPHeader->[0]output
}
```
Click Configuration

Try some of the other tutorial exercises on your own!
Click Elements

- There are many click elements already included
  - [http://read.cs.ucla.edu/click/elements](http://read.cs.ucla.edu/click/elements)
- But you might need to write your own
- Naming standards:
  - Element Name = CamelCase
  - File name = lowercase.cc (and lowercase.hh)
Click Elements

simplepullelement.hh:

class SimplePullElement: public Element {
    public: …
    const char *class_name() const {return "SimplePullElement";}  
    const char* port_count() const{ return "1/1"; } 
    const char* processing() const{ return PULL; } 
    Packet*pull(int i);
}

simplepullelement.cc:

Packet* SimplePullElement::pull(int i){
    Packet* p = input(0).pull();
    if (p == 0) return 0;
    click_chatter("Destination IP address %s", IPAddress(p->ip_header()->ip_dst.s_addr).c_str() );
    click_chatter("Got a packet of size %d",p->length());
    if(p->length() > maxSize){ p->kill(); return 0; }
    else return p;
}

Packet *pull(int i) for pull elements, pull request output port i

void push(int i, Packet *p) for push elements, input port i

Packet *simple_action(Packet *) for agnostic elements
Click Element - Ports

simplepullelement.hh:

class SimplePullElement: public Element {
    public: ...
    const char *class_name() const {return "SimplePullElement";}
    const char* port_count() const{ return"1/1"; }
    const char* processing() const{ return PULL; }
    Packet*pull(int i);
}

- "<input>/<output>" ports
- "1-2/2" - one or two input ports, two output ports
- "1/-4" - up to 4 output ports
- "2/-1" - at least 2 input ports
- "1-/=" - at least 1 input port, same number of output ports
Click Library

- CPP standard libraries cannot be used
- Instead, use the click libraries
- Many CPP STL have click equivalents
- e.g. vector and hashmap
  - #include <click/vector.hh>
  - #include <click/hastable.hh>
Adding/Building your element

- Within ‘~/click’
- Finish your element (.cc file) with an export statement
  - EXPORT_ELEMENT(SimplePullElement)
- Move your element to a folder in ~/click/elements/
- Add that folder to the configure file
  - e.g. if your element is in ‘local’, add “--enable-local”
- Run ‘make elemlist’
- Run ‘make install’
Final Hints

- Use `click_chatter("...")` to print
- Wireshark can save, import packet traces (pcap/dump files)
- See other elements (~/click/elements/) for hints

Resources:

http://www.read.cs.ucla.edu/click/
http://www.pats.ua.ac.be/software/click/