Software Defined Networking

Lab Work 3 Introduction

Jeremias Blendin, Leonhard Nobach, Christian Koch,
Julius Rückert, Matthias Wichtlhuber

PS - Peer-to-Peer Systems Engineering Lab
Dept. of Electrical Engineering and Information Technology
Technische Universität Darmstadt
Rundeturmstr. 12, D-64283 Darmstadt, Germany
http://www.ps.tu-darmstadt.de/

24. October 2014

Lab 2 Task 1 Solution (1)

- Some confusion/uncertainty regarding the task
  - We will stick to this line in the description: “A ping from h1 to h2 should result in a ping from h1 to h2 and h3. As a result, h1 receives more packets than it has sent.”

```
mininet> h1 ping h2
64 bytes from 10.0.0.2: icmp_seq=343 ttl=64 time=0.076 ms
64 bytes from 10.0.0.2: icmp_seq=343 ttl=64 time=0.079 ms (DUP!)
```

  - Verify using tcpdump

```
:~$ sudo tcpdump -ei s1-eth1
52 00:00:00:00:00:01 > 00:00:00:00:00:02, ethertype IPv4 (0x0800), length 98: 10.0.0.1 > 10.0.0.2:
ICMP echo request, id 5168, seq 48, length 64
92 00:00:00:00:00:02 > 00:00:00:00:00:01, ethertype IPv4 (0x0800), length 98: 10.0.0.2 > 10.0.0.1:
ICMP echo reply, id 5168, seq 48, length 64
95 00:00:00:00:00:03 > 00:00:00:00:00:01, ethertype IPv4 (0x0800), length 98: 10.0.0.3 > 10.0.0.1:
ICMP echo reply, id 5168, seq 48, length 64
```

Lab 2 Task 1 Solution (2)

- We will accept all solutions that achieve that
  - Ideal solution: install OpenFlow rules
  - Other approaches
    - Duplicate packets on controller

```
def _duplicate_pingreq_to_host(pkg, target):
       […]
       new_packet.add_protocol(ethernet.ethernet(ethertype=eth.ethertype,
                                                 dst=target[1],
                                                 src=eth.src))
       new_packet.add_protocol(ipv4.ipv4(dst=target[0],
                                         src=ipv4_frame.src,
                                         proto=ipv4_frame.proto))
       new_packet.add_protocol(icmp.icmp(type_=icmp.ICMP_ECHO_REQUEST,
                                         code=icmp_frame.code,
                                         csum=icmp_frame.csum,
                                         data=icmp_frame.data))
       return new_packet
```

  - Port mirroring, but no duplicate ping

```
[Somewhere inside add_flow]
# dl_type required in order to use l3 protocol matching
        match_icmp = datapath.ofproto_parser.OFPMatch(
            in_port=in_port, dl_type=0x0800, nw_proto=1)

        actions_icmp = [datapath.ofproto_parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
```

Lab 2 Task 1 Sample Solution (1)

- See the file „simple_switch_duplicate.py“ for a complete example solution
- The network is static and known: Add a host list

```
# Code by Patrick Welzel and Mahshid Okhovatzadeh
       self.hosts_for_omniping = [
                                   {'port': 1,
                                    'ip': '10.0.0.1',
                                    'mac': '00:00:00:00:00:1'},
                                   {'port': 2,
                                    'ip': '10.0.0.2',
                                    'mac': '00:00:00:00:00:2'},
                                   {'port': 3,
                                    'ip': '10.0.0.3',
                                    'mac': '00:00:00:00:00:3'},
                                  ]
```

Lab 2 Task 1 Sample Solution (2)

- Add additional output rules

```
Adarsh Chikkaballapur Umashankar
Bhargava Narasipura
  actions=[]
  # install a flow to avoid packet_in next time
  if out_port != ofproto.OFPP_FLOOD:
    for host in self.hosts_for_omniping:
      if msg.in_port != host['port']:
        if out_port != host['port']:
          actions.append(datapath.ofproto_parser.OFPActionSetDlDst(haddr_to_bin(host['mac'])))
          actions.append(datapath.ofproto_parser.OFPActionSetNwDst(ofctl.ipv4_to_int(host['ip'])))
        actions.append(datapath.ofproto_parser.OFPActionOutput(host['port']))
      self.logger.info("actions is %s ", actions)
      self.add_flow(datapath, msg.in_port, dst, actions)
```

- OpenFlow rules

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=41.308s, table=0, n_packets=32, n_bytes=3136, idle_timeout=20, idle_age=9,
icmp,in_port=2 actions=output:1,mod_dl_dst:00:00:00:00:00:03,mod_nw_dst:10.0.0.3,output:3
 cookie=0x0, duration=41.311s, table=0, n_packets=32, n_bytes=3136, idle_timeout=20, idle_age=9,
icmp,in_port=1 actions=output:2,mod_dl_dst:00:00:00:00:00:03,mod_nw_dst:10.0.0.3,output:3
 cookie=0x0, duration=41.308s, table=0, n_packets=32, n_bytes=3136, idle_timeout=20, idle_age=9,
icmp,in_port=3 actions=output:1,mod_dl_dst:00:00:00:00:00:02,mod_nw_dst:10.0.0.2,output:2
```

Lab Work 3

Exploring OpenFlow 1.3

Upgrading Mininet

- The Mininet version on your VM does not support OpenFlow 1.3
- We have to upgrade to the latest version
- Reboot your existing Mininet VM and enter:

```
~$ cd mininet
~/mininet$ git pull
~/mininet$ git checkout 2.2.0b1
~/mininet$ sudo ./util/install.sh -n
```

- Verify your installation

```
~/mininet$ mn --version
2.2.0b1
~/mininet$
```

Run a simple_switch with OpenFlow 1.0

- Test with OpenFlow 1.0
  - First terminal: ssh session 1:

```
~$ ryu-manager ryu.app.simple_switch
```

  - Second terminal: ssh session 2

```
~$ mn --topo single,3 --mac --arp --switch ovsk \
    --controller=remote,ip=127.0.0.1
mininet> h1 ping h2
```

  - Third terminal: ssh session 3
    - Note the new tool: ovs-ofctl (google „man ovs-ofctl“ for details)

```
~$ sudo ovs-ofctl dump-flows s1
```

- Note the debug output in session 1
  - Investigate the source code
  - Find out the meaning of the log messages

Run a simple_switch with OpenFlow 1.3

- Test with OpenFlow 1.3
  - First terminal: ssh session 1:

```
~$ ryu-manager ryu.app.simple_switch_13
```

  - Second terminal: ssh session 2

```
~$ mn --topo single,3 --mac --arp \
         --switch ovsk,protocols=OpenFlow13 \
         --controller=remote,ip=127.0.0.1
mininet> h1 ping h2
```

  - Third terminal: ssh session 3

```
~$ sudo ovs-ofctl dump-flows s1
2014-11-24T11:47:01Z|00001|vconn|WARN|unix:/var/run/openvswitch/s1.mgmt: version negotiation
failed (we support version 0x01, peer supports version 0x04)
ovs-ofctl: s1: failed to connect to socket (Broken pipe)
```

Run a simple_switch with OpenFlow 1.3

- Third terminal: ssh session 3

```
~$ sudo ovs-ofctl dump-flows s1
2014-11-24T11:47:01Z|00001|vconn|WARN|unix:/var/run/openvswitch/s1.mgmt: version negotiation failed (we support version 0x01, peer
supports version 0x04)
ovs-ofctl: s1: failed to connect to socket (Broken pipe)
```

- Version negotiation failed
  - We (ovs-ofctl) support version 0x01 (OpenFlow 1.0)
  - Peer (Open vSwitch) supports version 0x04 (OpenFlow 1.3)

  ➡ note the version notation

```
~$ sudo ovs-ofctl dump-flows -O Openflow13 s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=13.922s, table=0, n_packets=13, n_bytes=1274, priority=1,in_port=2,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=12.924s, table=0, n_packets=12, n_bytes=1176, priority=1,in_port=1,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=14.541s, table=0, n_packets=6, n_bytes=588, priority=0 actions=CONTROLLER:65535
```

- Specify the OpenFlow version "`-O OpenFlow1X`"

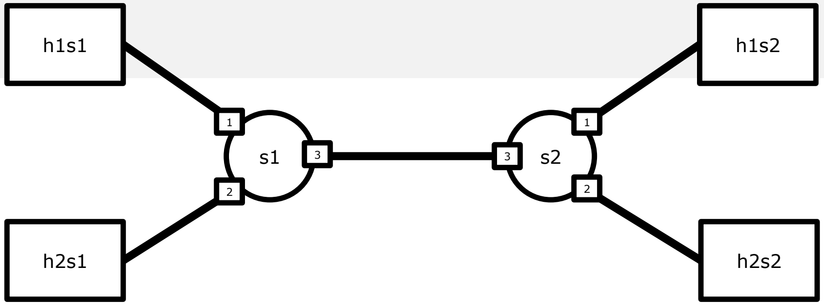
Task 1

- Look at the source code of the Ryu modules simple_switch.py and simple_switch_13.py.
  - Describe and explain the differences between the OpenFlow 1.0 and OpenFlow 1.3 version of simple_switch.

Task 2 (1)

1. Have a look at the OpenFlow 1.0-based filtering switch simple_switch_filter.py.
   1. The basic forwarding method is again layer 2 switching.
   2. We increase security by allowing only one host per port
   3. This rule does not apply to inter-switch links
   4. Run the topology:

```
sudo mn --topo linear,n=2,k=2 --mac --arp --switch ovsk \
        --controller=remote,ip=127.0.0.1
mininet> h1s1 ping h2s2
```



2. Verify
   1. Install and run MAC spoofing tool

```
sudo apt-get update && sudo apt-get install nmap
mininet> h1s1 nmap --spoof-mac 00:00:00:00:00:33 10.0.0.2
```

   2. You should see the following message in your controller output:

```
dropping spoofed packet on s1 src=00:00:00:00:00:33 dst=ff:ff:ff:ff:ff:ff in_port=1
```

Task 2 (2)

3. Have a look at the OpenFlow 1.0 based program that implements the filtering switch described in the last slide: simple_switch_filter.py.

4. Create an OpenFlow 1.3 version called simple_switch_filter_13.py.
   1. Base your implementation on simple_switch_13.py and simple_switch_filter.py
   2. Use two flow tables for the OpenFlow 1.3 version
      1. Use the first flow table for matching input ports and source MAC address
      2. Use the second flow table for sending the packets out the correct port.

5. Describe and discuss the differences regarding the number of flow rules used
   1. Consider different topologies and number of hosts for the sake of discussion even though running them with the simple_switch_filter.py is not possible
   2. Mathematically describe an upper bound for the number of flow rules
      1. For the OpenFlow 1.0 version
      2. For the OpenFlow 1.3 version

Task 2

Old version for refernce only

See updated version of task 2 on last slide

1. Have a look at the OpenFlow 1.0 based sample solution for lab 2 simple_switch_duplicate.py.

2. Create an OpenFlow 1.3 version of simple_switch_duplicate.py based on simple_switch_13.py.
    1. Use one flow table for matching of ports
    2. Use a second flow table for duplication and setting destination MAC and IP address

3. Describe and discuss the differences regarding the number of flow rules used
    1. Assume a large number of connected hosts for the sake of discussion

Tools

- Looking for a Python IDE?
  - Eclipse with PyDev plugin
    - http://pydev.org/
  - PyCharm (free for Student for research purposes)
    https://www.jetbrains.com/student/

- Things to be aware of
  - Ryu relies on a concurrent networking library called Eventlet (http://eventlet.net)
  - Debugging Eventlet requires special support and does currently neither work with PyDev nor with PyCharm out of the box