Exercise for Lecture Software Defined Networking

Prof. Dr. David Hausheer
Julius Rückert, Leonhard Nobach

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter Term 2015/16
Exercise No. 3
Published at: 17.11.2015
Submission exclusively via Moodle, Deadline: 24.11.2015
Contact: [rueckert|lnobach]@ps.tu-darmstadt.de
Web: http://www.ps.tu-darmstadt.de/teaching/ws1516/sdn/
Submission: https://moodle.tu-darmstadt.de/enrol/index.php?id=6349

Surname (Nachname):	
First name (Vorname):	
ID# (Matrikelnummer):	

Problem 3.1 - SDN Hardware

a) Explain in your own words the main differences between the OpenFlow hardware classes “SOFTWARE” and “HARDWARE”.

b) OpenFlow v1.4.0 introduces the notion of *vacancy events.* Which inherent problem does this feature address? Shortly explain what *vacancy events* are and what would happen if a controller does not make use of them.

c) Explain and briefly discuss which parts of an OpenFlow flow entry should be stored in TCAM and which could be placed in normal DRAM to minimize the required TCAM space and still ensure constant-time packet processing. (Note: You can exclude the priority of a flow entry from the discussion.)

Hint: The answer can be derived from the material presented in the lecture. For a more detailed understanding, it might be helpful to understand how CAMs and TCAMs work on a lower level: `https://www.pagiamtzis.com/cam/camintro/` (this is optional material)

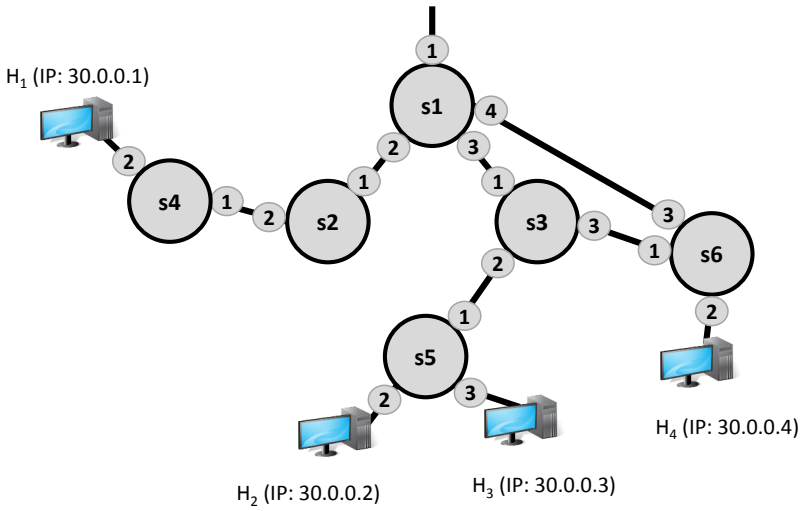
Problem 3.2 - NOS and SDN Languages

a) One problem that should be addressed separately when introducing abstractions in networking (according to Scott Shenker) is the definition of a constrained, yet flexible, forwarding model. Name two additional problems that need to be addressed.

b) What is the major advantage when implementing, for example, a Spanning Tree approach or OSPF in an SDN with NOS.

c) Consider the Pyretic language in its initial version as presented in the lecture. Specify a Pyretic policy that implements the following behavior: (1) Incoming packets are filtered for TCP port 80 and 8080, packets for other ports are dropped. (2) The port 80 packets received on switch port 1 are duplicated and send out to switch port 2 and 3. (3) the destination IP address of the remaining packets is rewritten to '192.168.2.100' and the packets are send out via switch port 1.

d) Assume an OpenFlow-based network that is managed by a controller implemented using Pyretic. A simplified version of the controller implementation is given in the following. The topology of the network is shown in the graphic below, consisting of six switches (s1-s6). The two numbers on the end of a link depict the physical network port numbers that a network link is connected to.
(1) Explain the semantic of the control application on an abstract level.
(2) What is the function/role of the combination of the destination IP-port pair ('20.0.0.1', 222)? Can you draw a connection to a system presented in the lecture?




```
1 from pyretic.lib import *
2
3 s1 = if_(match(switch=s1),
4          if_(match(dstip='10.0.0.1',dstport=111),
5              pop(dstip) >> push(dstip='20.0.0.1') >> pop(dstport) >>
                  push(dstport=222) >> (fwd(2)|fwd(3)),
6              drop),
7          passthrough)
8
```

```
 9 s2 = if_(match(switch=s2),
10          if_(match(dstip='20.0.0.1',dstport=222),
11              fwd(2),
12              drop),
13          passthrough)
14
15 s3 = if_(match(switch=s3),
16          if_(match(dstip='20.0.0.1',dstport=222),
17              flood,
18              drop),
19          passthrough)
20
21 s4 = if_(match(switch=s4),
22          if_(match(dstip='20.0.0.1',dstport=222),
23              pop(dstip) >> push(dstip='30.0.0.1') >> pop(dstport) >>
                    push(dstport=123) >> fwd(2),
24              drop),
25          passthrough)
26
27 s5 = if_(match(switch=s5),
28          if_(match(dstip='20.0.0.1',dstport=222),
29              (pop(dstip) >> push(dstip='30.0.0.2') >> pop(dstport) >>
                    push(dstport=234) >> fwd(2) |
30               pop(dstip) >> push(dstip='30.0.0.3') >> pop(dstport) >>
                    push(dstport=345) >> fwd(3))
31              drop),
32          passthrough)
33
34 s6 = if_(match(switch=s6),
35          if_(match(dstip='20.0.0.1',dstport=222),
36              pop(dstip) >> push(dstip='30.0.0.4') >> pop(dstport) >>
                    push(dstport=456) >> fwd(2)
37              drop),
38          passthrough)
39
40 def main():
41   return s1 >> s2 >> s3 >> s4 >> s5 >> s6
```

Problem 3.3 - The OpenFlow Protocol

For the following questions you have to use the official OpenFlow specifications. You are not intended to read the whole documents! Get familiar with their structure and use them to look up details. Knowing how to navigate and find details within the specifications is essential for several later tasks and the lab. If we do not specify the version to be used, we assume version v1.5.0 in the following.
Relevant OpenFlow Specifications for this task:

- v1.5.0: `https://goo.gl/VDxZN2`

Besides, we recommend the following webpage that can help to investigate differences between versions of the OpenFlow protocol etc.: `http://flowgrammable.org/sdn/openflow/`

a) OpenFlow v1.5.0 introduces the possibility of TCP flag matching. Name some examples what this new feature could be used for and sketch how the solution would work and what other OpenFlow features it might require/use.

b) Shortly explain the different steps of packet processing inside an OpenFlow Switch with multiple flow tables and in the context of *Pipeline Processing*.