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`

– Example Solution –

Problem 3.1 - SDN Hardware

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

Solution:
SOFTWARE: This class includes devices that initially were not designed for OpenFlow. The vendors provide software/firmware updates to several of their devices that enables them to support the configuration/communication using the OpenFlow protocol. The problem about these devices usually is that most OpenFlow features are realized in software only, without hardware support for all operations. This makes using these devices for many extended/advanced OpenFlow scenarios problematic. The packet processing speed can greatly drop if software-supported features are used.

HARDWARE: Devices of this class fully support all operations required by OpenFlow in hardware (ASIC). This usually requires that the device is designed for OpenFlow. Only a few devices in the class exist today. An important reason for this is that developing purpose-built hardware is very costly and takes time.

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.

Solution:
TCAM space is essential for a line-rate packet processing at the switches. Yet, TCAMs are constraint in their size, they are expensive and large hardware elements with a high power consumption (cf. lecture on OpenFlow hardware). As a result, their limited capacity is an important factor to be considered also by the controller. If not considered, full matching tables can result in OF matchers being installed to low performance memory or the refusal of further rule installations. Both cases can result in serious service degradations or outage, situations that

should be avoided. As a potential solution to this problem, *vacancy events* as part of the OpenFlow specification since v1.4.0 were introduced (cf. OpenFlow specification v1.5.0 page 269). They enable the controller to specify a capacity threshold on which the controller is informed. It can take approriate actions, replan the installation of rules to other switches, aggregate existing rules, etc.

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)

Solution:

To maintain a constant-time packet processing, it is essential to realize the matching of packets to installed flow entries in constant time. To achieve this objective it is important to avoid the use of any search algorithms that are dependent on the number of installed flow entries. This is possible using TCAMs as they enable a lookup of flow table entries with a comlexity of O(1). But this implies that not more than the matcher (and the priority) of a flow rule needs to be stored in TCAM. To achieve a constant processing a constant-time lookup is performed on these two components, resulting in a DRAM memory address at which the a data structure with the remaining components of the flow entry is stored. To access the individual components (i.e. counters, instructions, etc.), no search in the memory is required.

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.

Solution:

- Distributed state
- Configuration

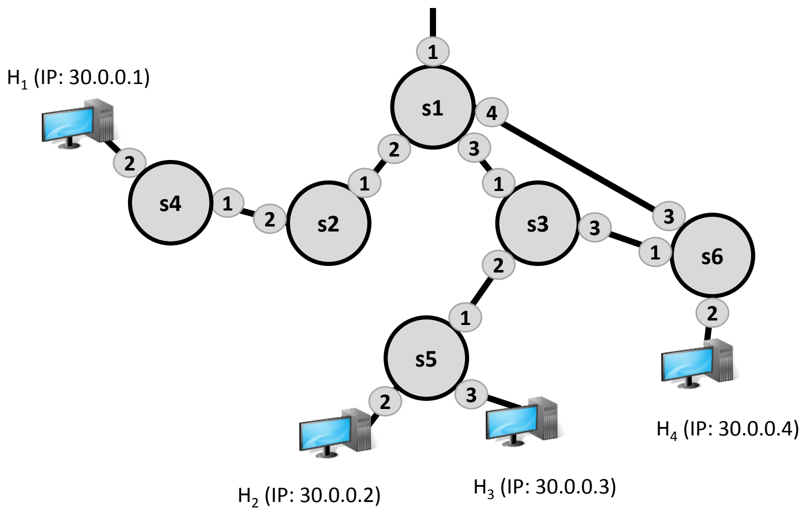
b) What is the major advantage when implementing, for example, a Spanning Tree approach or OSPF in an SDN with NOS.

Solution: The network topology does not need to be discovered (as required for a distributed implementation) before focusing on the actual implementation of the policies. Using SDN with a NOS, the topology information is already available from the NOS and policies can directly be applied based on this information.

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.

```
  if_(match(dstport=80, inport=1),
     (fwd(2)|fwd(3)),
     if_(match(dstport=8080),
        pop(dstip) >> push(dstip=192.168.2.100) >> fwd(1),
        drop))
```

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?



```
from pyretic.lib import *

s1 = if_(match(switch=s1),
        if_(match(dstip='10.0.0.1',dstport=111),
            pop(dstip) >> push(dstip='20.0.0.1') >> pop(dstport) >>
               push(dstport=222) >> (fwd(2)|fwd(3)),
            drop),
        passthrough)

```

```
s2 = if_(match(switch=s2),
          if_(match(dstip='20.0.0.1',dstport=222),
              fwd(2),
              drop),
          passthrough)

s3 = if_(match(switch=s3),
          if_(match(dstip='20.0.0.1',dstport=222),
              flood,
              drop),
          passthrough)

s4 = if_(match(switch=s4),
          if_(match(dstip='20.0.0.1',dstport=222),
              pop(dstip) >> push(dstip='30.0.0.1') >> pop(dstport) >>
                 push(dstport=123) >> fwd(2),
              drop),
          passthrough)

s5 = if_(match(switch=s5),
          if_(match(dstip='20.0.0.1',dstport=222),
              (pop(dstip) >> push(dstip='30.0.0.2') >> pop(dstport) >>
                 push(dstport=234) >> fwd(2) |
               pop(dstip) >> push(dstip='30.0.0.3') >> pop(dstport) >>
                  push(dstport=345) >> fwd(3))
              drop),
          passthrough)

s6 = if_(match(switch=s6),
          if_(match(dstip='20.0.0.1',dstport=222),
              pop(dstip) >> push(dstip='30.0.0.4') >> pop(dstport) >>
                 push(dstport=456) >> fwd(2)
              drop),
          passthrough)

def main():
  return s1 >> s2 >> s3 >> s4 >> s5 >> s6
```

Solution:
(1) The control application performs a mulitcasting of all packets that arrive at s1 with the destination IP-port pair ('10.0.0.1',111) to all four hosts. Other packets are dropped. In fact, this controller only handles the packets involved in the multicasting. No other communication is possible. The multicasting happens only using IP unicast addresses. The destination IP is rewritten for the individual hosts and using host-specific destination ports. These ports could be the specific ports that an application at the hosts waits for incoming packets.
(2) This pair functions as multicast group identifier inside the network for the actual multicast routing (forwarding and duplication) inside the network. This concept is very similar to "Software-Defined Multicast" as presented in lecture 5.

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.

Solution: The standard documents names one example: "This field allow to match all flags, such as SYN, ACK and FIN, and may be used to detect the start and the end of TCP connections." One could also imagine many other applications, such as TCP SYN flood attack detection in combination with METERS. We will extend this list by other examples that you name).

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

Solution: "When processed by a flow table, the packet is matched against the flow entries of the flow table to select a flow entry. If a flow entry is found, the instruction set included in that flow entry is executed. These instructions may explicitly direct the packet to another flow table (using the GotoTable Instruction), where the same process is repeated again." (page 20 of v1.5.0 standard document)