Exercise for Lecture Software Defined Networking

Prof. Dr. David Hausheer
Julius Rückert, Leonhard Nobach

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter Term 2015/16
Exercise No. 5
Published at: 08.12.2015
Submission exclusively via Moodle, Deadline: 12.01.2016
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 5.1 - NEC guest lecture: SDN, NFV, and Service Chaining

a) What does the term "commoditization" refer to?

Solution:

- Bare metal switches move towards whiteboxes based purely on merchant silicon.
- This was first done for server machines and now is done for L2/L3 switching and routing network equipment.

b) Explain why the flow granularity in OpenFlow is said to be not predefined.

Solution:

- At some point, the matching in OpenFlow was understood too narrowly by some experts that then reduced it to be able to only work on 3/5-tuples.
- Indeed and as you already learned, OpenFlow can match on any combination of supported fields.

c) Explain the proposed driver-like abstraction on SDN controllers.

Solution:

- The concept of drives is know from operating systems, where they provide a high-level software interface to hardware components and hide hardware details to applications.
- This idea is mapped to SDN controllers in the following manner: On an abstract level, the SDN controller can be understood to act like a driver to the SDN applications in that it translates high-level calls to low-level network behavior. This way, SDN applications could work on a high-level northbound API. As extreme, this might even hide the specific southbound API used.

d) Explain the concept of "intents" in the context of Northbound Interfaces (NBIs). What are the key benefits for an network architecture of using intents?

Note: As the topic was only very briefly discussed in the lecture, please read the following ONF blog article to deepen your understanding on the topic and to answer the above question: `https://goo.gl/bl0MBK`.

Solution: The concept of *intents* for NBIs:

- The idea of intents in the context of NBIs is to let an SDN application specify *what* it requires from the network, without specifying details on *how* it is to be achieved. The latter is compared to a *prescription* in the blog article.
- The goal of intents thus is to help shifting the focus from networking details when interacting with the controller (e.g. "use MPLS", "apply class-based routing", "use commercial product XYZ") to focus more on the actual high-level requirements of distributed network applications. An intent could, for example, request low delay, high speed, or secure datapaths, without specifying how to achieve this goal.

Benefits for network architecture:

- The idea of Intents allows the SDN application to operate on a higher level by directly specifiying business intents/objectives without mixing them with the networking logic required to achieve these intents.
- This directly translates to avoiding dependency on specific technologies, approaches, protocols, as well as vendor-specific features or technologies.
- This in turn allows network providers to choose a technology that meets their internal requirements the best and achieves the intent.
- A chosen approach can be easily changed without impacting the SDN application as long as the intent is fullfilled. There are actually a lot of analogies to having interfaces in software engineering and all the related principles to achieve separation of concern.

e) New Tables proposed by OpenState

In the lecture, an approach called *OpenState* was presented that extends the matching behavior of OpenFlow. For this, it proposes to expose two new table abstractions to the controller: the *state table* and the *XFSM table*. Explain the purpose of these two tables and how they are used in handling of incoming packets. For this, please have a look at the original paper on OpenState (Hint: Sections 2.1 and 2.2 are relevant for answering the question): `http://openstate-sdn.org/pub/openstate-ccr.pdf`

Solution:

- OpenState introduces the possibility of using a locally managed state in the matching of incoming packets.

- XFSM table: The table is used to encode programmer-defined states, conditions that lead to a state transition, the respective next state for each of the states, and actions to be executed on state transitions of an *eXtended Finite State Machine* (XFSM). An XFSM can be considered a Mearly Machine for the basic examples.
- State table: This table is used to keep the actual state for a flow. It contains a flow key that is based in header field values that identify the flow and an associated state that matches one of the states in the XFSM table.
- The steps on incoming packets are the following: (1) a key is extracted from the incoming packet and matched against the entries in the state table. The resulting state is passed on to the second step together with the header fields of the packet. If there was no explicit state found in the state table, a default state is passed on. (2) The state and the header fields are matched against the state and event conditions in the XFSM table. The result of this second step is a list of actions to be applied to the packet as well as the label of the next state. (3) The packet headers are used to, again, extract the match key of the packet and update the respective state in the state table with the new state label. In parallel, the paket headers and the list of actions is passed on to the normal pipeline processing of OpenFlow.

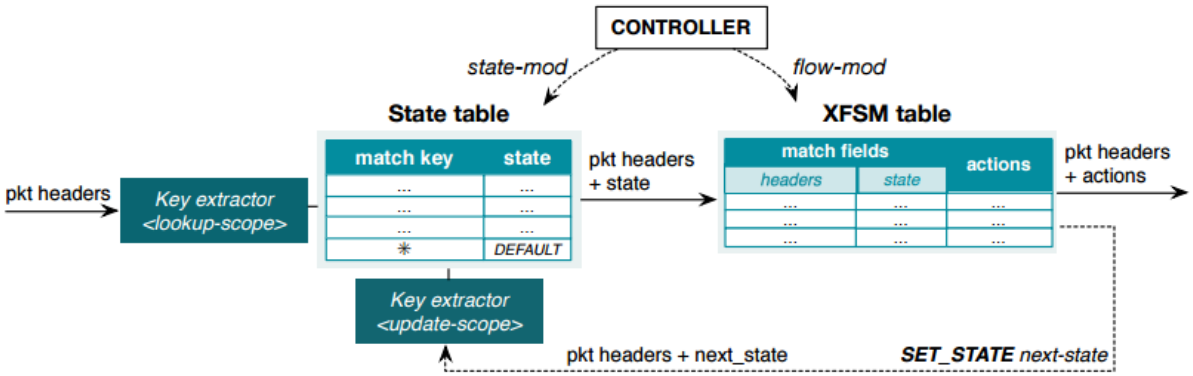

Figure 1: OpenState tables. Source: http://openstate-sdn.org/pub/openstate-ccr.pdf

f) Applicability ot OpenState

Which types of use cases or applications could benefit most from OpenState?

Solution:

- In use cases where rules cannot be proactively installed and such that pose high requirement for a timely processing of new incoming flows could potentially benefit from the approach.
- Besides, the most gain could be achieved if the intended behavior can be mapped to a state machine that is completely local to the switch. The *port knocking* use case as described in the paper is a good example. Whenever global knowledge is required, additional communication with the controller is inevitable, reducing the benefit of the local approach.

g) Using OpenState for a simple use case

At the end of Section 2.2 of the above referenced paper on OpenState, the "port knocking" example is realized using OpenState. Briefly explain how it is mapped to the two tables.

Solution:

- The five states of the state machine are mapped to the XFSM table and incoming packets that match the DEFAULT state event fields trigger setting a new entry in the state table.
- Subsequent matches of the predefined ports cause state transitions, finally opening port 22 and forwarding all traffic sent to it. Traffic to other ports is dropped.

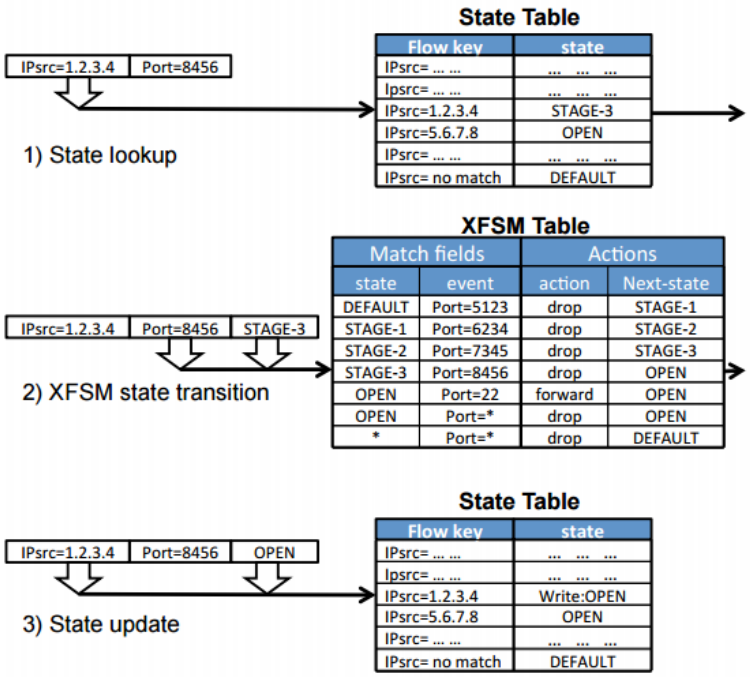

Figure 2: OpenState example. Source: http://openstate-sdn.org/pub/openstate-ccr.pdf

Problem 5.2 - Alcatel Lucent Bell Labs guest lecture: Challenges in NFV

a) What is the "state explosion" problem discussed in the lecture and how is it caused?

Solution:

- The state explosion is caused in use cases where aggregation of flows (e.g. using IP prefixes) is not sufficient. An example for such a use case is service chaining that requires granular per-flow paths through a network.
- In this case, individual flow entries are to be maintained at the flow paths that cannot be easily aggregated, thus leading to constant increase in flow entries with each new application path.

- Especially in the core of large networks the goal usually is to limit the number of required flow rules. In traditional networks, MPLS is an approach that addresses exactly this problem. Flows are mapped to equivalence classes at the edge of the network and core switches operate on a finite set of class labels.

b) What does the notion of *path switching* refer to and how does it compare to source routing?

The paper presented in the lecture can be accessed here if a deeper understanding is desired: `http://conferences2.sigcomm.org/co-next/2015/img/papers/conext15-final232.pdf`

Solution:

- *Path switching* is an approach that is proposed in the above mentioned paper. It allows a fine-grained, aggregation-free per-flow routing in an SDN network, with data forwarding runing purely at the data plane.
- It encodes variable-length network paths in a fixed-size header field in an efficient manner. While the length of the supported paths is limited (a result of the fixed-size header approach), it is shown to support path lengths that are sufficient for realistic deployments.
- Source routing, in contrast, typically results in variable-length paths which cannot be easily encoded in a fixed-size header field. Encoding variable-sized paths in a packet header field is not easy to support as, e.g., the MTU for the rest of the packet would have to be reduced, depending on the path length.

c) Why does *path switching* use a pointer field? What would be an alternative approach?

Solution:

- As interface labels are endcoded with variable bit lengths in the prefix-free encoding variant, a switch needs to determin the position in the path label string where the next label starts. The switch needs to be able to do this lookup in constant time to allow for line-speed processing. The pointer encodes the start position and thus enables this fast lookup.
- An alternative (also named in the paper) would be to perform a bit-shifting of the label string whenever a label was used and before the packet is forwarded to the next hop.