Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

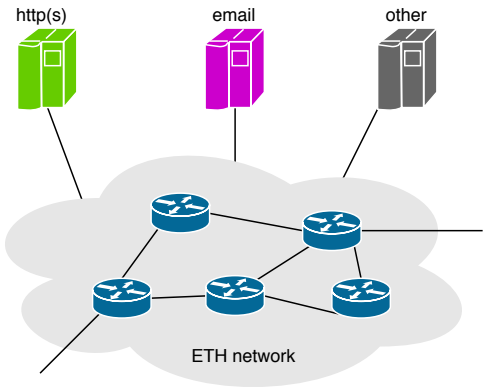
Networked Systems
ETH Zürich — seit 2015

Communication Networks

Prof. Laurent Vanbever

Exercise 11 – Programmable Networks

11.1 Network Virtualization with OpenFlow



(Fictional) ETH network with three OpenFlow controllers.

OpenFlow simplifies *network virtualization*, i.e. managing different parts of your network or traffic separately.

Assume that ETH has installed OpenFlow Switches and wants to benefit from network virtualization. Let there be three teams: One responsible for *web-traffic, i.e. HTTP(s)*, one for *emails*, and one for everything else. Each team operates their own OpenFlow Controller.

As you have learned, if an OpenFlow switch does not know what to do with a packet, it forwards the packet to the controller. However, it is also possible to install an explicit matching rule to send packets to a controller, which we want to use for network virtualization.

Use the following pseudo-code command to install such a matching rule (and remember that all OpenFlow matching rules are ordered by the *priority* you assign to them):

```
send_to_controller(match, priority, controller)
```

where controller can simply be *web*, *email*, or *other* and you can use the following syntax for matches:

```
{src=42.0.0.*, dst=*}
```

You may use any fields you need, and use `*` as a wildcard.

- Use pseudo-code to install matching rules for forwarding packets the correct controller. In particular, consider which fields you have to match on, and which priority you set. Note: You can (and have to) use `send_to_controller` multiple times.

- How does the priority you choose influence other OpenFlow rules installed by the controllers?

11.2 Flexible OpenFlow Switches

An OpenFlow switch can emulate different devices.

With the capabilities to match on-, and modify (some) header fields, and either forwarding or dropping the packet, OpenFlow switches can emulate various hardware. In this exercise, we compare *NAT* and *firewalls*.

First, consider *matching* (for simplicity, only look at TCP and UDP traffic):

- Which header fields does the switch need to match as a NAT, or as a firewall? Are there differences?

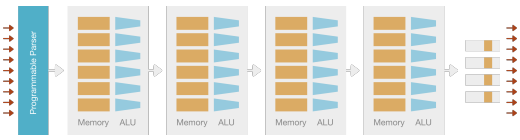
Next, assume the switch has matched on a packet and sent it to the controller. On the controller, for NAT and firewall respectively:

- Which decisions need to be made?
- Which data needs to be stored?
- Is any external information required or useful?
- Which rules need to be installed on the switch?

Finally, with an OpenFlow switch, we do not need to decide between NAT and firewall, as our switch can be both at the same time. However, this is not without challenges.

- What difficulties arise when combining multiple goals, such as address translation and filtering?

11.3 Programmable Dataplanes



P4 offers a programmable processing pipeline.

In OpenFlow, only the controller is programmable. While an OpenFlow switch provides an API such that different controllers can interact with it, the actual matching and actions are fixed hardware functions.

The *Protocol Independent Switch Architecute (PISA)* along with the domain specified programming language *P4* are a natural next step in programmability, as they allow to program the switch itself, i.e. the packet parsing as well as modifications to the packet.

What led to this development? Why are fixed switch functions not enough?