Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems Group

Spring 2017

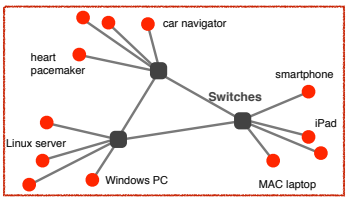
Prof. L. Vanbever/ T. Bühler, R. Birkner, T. Holterbach, R. Meier

Communication Networks

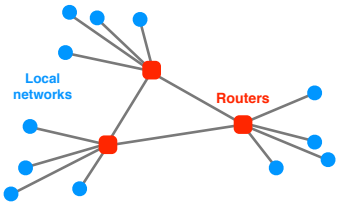
Assignment 1: Enabling end-to-end connectivity in a Local Network

1 Introduction

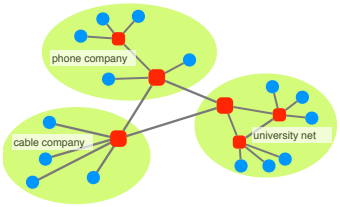
In the first lectures of this course, you have learned what a network is made of, how it is organized and how communication happens. Local networks interconnect a limited (generally tens to a thousand) number of end-hosts. To forward packets to a destination, they usually rely on many switches (Fig 1(a)). Switches forward packets based on the destination MAC address. When the number of end-hosts increases, a destination MAC address based forwarding (also called a layer 2 forwarding) does not scale anymore. In this case, end-hosts are grouped into subnetworks (using IP submasks), and the forwarding between subnetworks is done by IP Routers. This set of subnetworks forms a network (Fig 2). Routers rely on the destination IP address to forward packets. Routers can also interconnect networks of different organizations (Fig 1(c)).



(a) A local network is composed of end-hosts and switches. For instance, all the hosts connected in a building can form a local network, and each floor is connected to another floor via a switch.



(b) A network is composed of subnetworks and IP routers. For instance, a large company can have several buildings. Inside each buildings there is a local network, and routers interconnect those local networks, such that two people in two different buildings can communicate.



(c) IP routers also interconnect networks of different organizations. For instance, Orange and Deutsche Telekom, two major European ISPs, are interconnected with IP routers. This is the Internet!

Figure 1: The Internet is a network of networks! In each case, different protocols and concepts are used to build and update a forwarding state that ensures connectivity.

In this assignment, we will learn how to build a forwarding state in a simple local network such that two internal hosts can communicate at layer 2.

2 Configuring a valid forwarding state in a local network

To be valid, the forwarding state in a local network must avoid dead ends and loops. It should also maximize the performance. You have learned during the class, that layer 2 switches use a spanning tree and flood packets to build a valid forwarding state. A switch floods a packet by sending it to all its activated ports *except* the incoming one. When a switch has learned where a host is located in the network, it does not need to flood packets destined to this host anymore. Instead, it just forwards it to the right output port.

In this assignment, we will use the very simple topology depicted in Figure 2. Because we know the topology and you have a limited amount of time allocated for this assignment, we are not going to follow the traditional

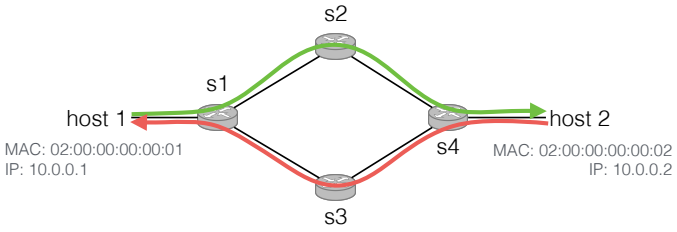
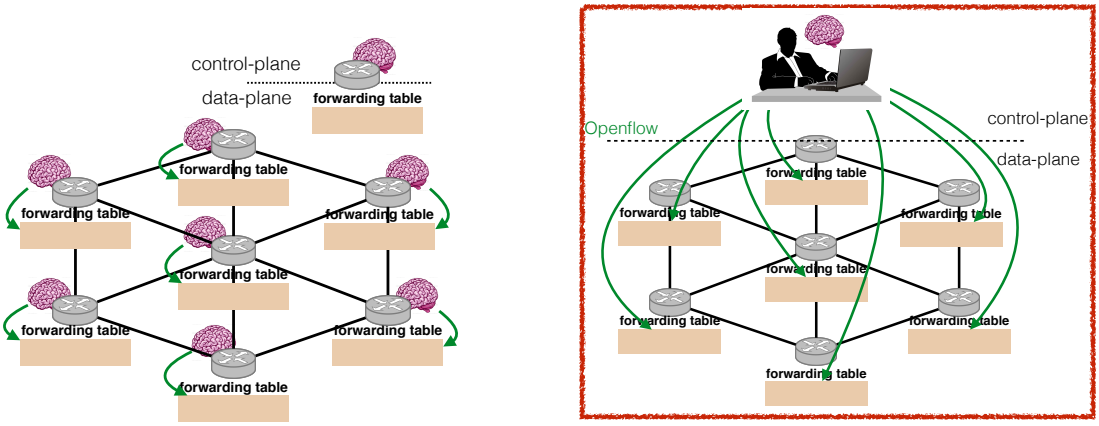


Figure 2: A very simple topology, with two hosts and four switches. Your goal will be to enable connectivity between host 1 and host 2.

spanning-tree computation, flooding and learning process. Instead, you will statically install forwarding rules in the switches such that the traffic from host 1 to host 2 traverses switch 2, and the traffic from host 2 to host 1 traverses switch 3.

3 Centralized decisions using Software-defined Networks

Forwarding decisions can be computed in two different ways. The first (and most used in the today's Internet) is a distributed computation (Fig. 3(a)). Each switch in the network collects information using standardized protocols (e.g. STP). Based on this local information, each switch populates the forwarding table. The second (and more recent) way is a centralized approach (Fig 3(b)). Each switch communicates with a centralized controller using standardized protocols (the most used in this case is OpenFlow). The controller knows the local state of all the switches composing the network. Based on this information, it computes the global state of the network and sends to each switch the corresponding local forwarding decisions. Such a network is called a Software-defined Network (SDN). Note that it is possible to have a hybrid network, where some routing decisions are centralized and some others are distributed.



(a) **Distributed decisions**.

(b) **Centralized decisions**.

Figure 3: Routing decisions can be computed locally on each device 3(b) or can be centralized using a controller 3(a). Networks that compute routing decisions with a centralized controller are called Software-defined Networks (SDN).

In a traditional network with distributed routing decisions, the software and the hardware blocks composing the switches are closed, meaning that network operators, researchers and students have limited control over the routing decisions. The new centralized Software-defined Network (SDN) approach offers network operators, researchers and students the ability to program, by themselves, their own routing decisions. For this reason, we use SDN for this assignment.

4 Programming switches' forwarding table with OpenFlow

The routing decisions of a switch are stored in a forwarding table. Forwarding tables are usually implemented in hardware to allow for very fast lookups. They are composed of a set of entries (Fig. 4). Each entry in a forwarding table can be decomposed in two parts: a "match" and an "action" part. The match part allows an entry to filter packets, such that its actions are only executed on the corresponding packets. Packets can be matched based on several fields, from layer 2 to layer 4, as well as the switch port number where the packets have been received. All the packets that match an entry form a flow. Several types of actions are possible. For instance, one action is to forward packets to a port. Another possible action is to drop the packets. Figure 4 depicts all the possible match rules and actions.

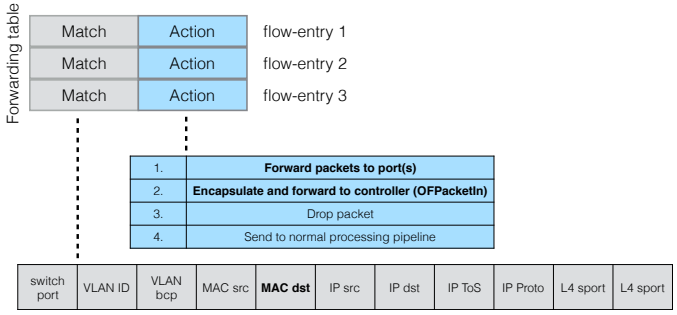


Figure 4: Description of a forwarding table in an OpenFlow switch. An entry is composed of a match and an action part. In bold are the matches and actions we will use in this assignment.

4.1 OpenFlow

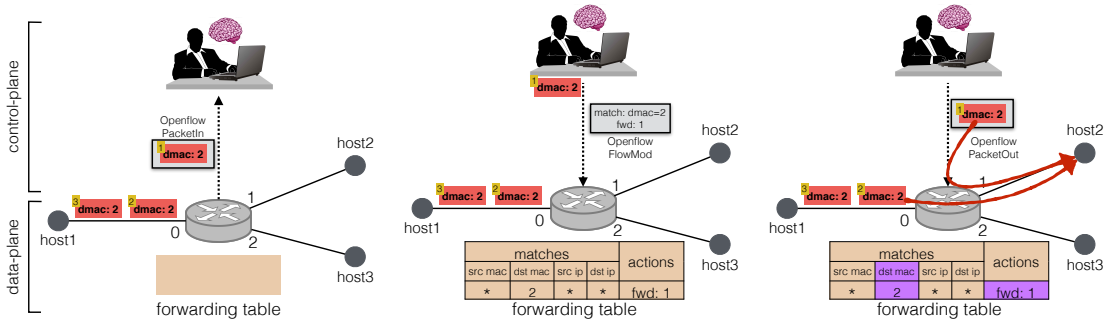
OpenFlow (OF) is an interface to remotely program the forwarding table of switches. The communication between OpenFlow switches and the controller is bidirectional, from the switches to the controller, and vice-versa.

Switches to controller [only required in the advanced version of this assignment]. Switches can send data-plane packets to the controller. In this case, packets are encapsulated in an **OF PacketIn** message (see action 2 in Figure 4) and sent to the controller. This OF PacketIn message is mainly used when a switch receives a packet that does not yet match any entry in its forwarding table (Fig 5(a)). When the controller receives the OF PacketIn message, it can then program the switch's forwarding table so that the switch is now able to forward this packet directly in hardware. In this assignment, we will use OF PacketIn messages to inform the controller when a switch receives a packet with an unknown destination MAC address. In practice, OF switches can send additional information to the controller. For instance, they can inform the controller when a port becomes up or down.

Controller to switches. The controller modifies the forwarding state of a switch using **OF FlowMod** messages. An OF FlowMod message instructs the switch to add, remove or update a flow-entry in its forwarding table. For instance, upon reception of an OF PacketIn message, a controller can decide to modify the forwarding state of the switch using an OF FlowMod message such that the packet in the OF PacketIn, as well as the next ones of the same flow, can now be forwarded by the switch in hardware (Fig. 5(b)). In such a scenario, we say that the controller installs forwarding rules **reactively**. The controller can also send an **OF PacketOut** message to a switch. This is used by the controller to physically send a packet via one of the switch ports. For instance, the controller can instruct the switch to send a packet previously diverted using an OF PacketIn message (Fig. 5(c)). The controller can also modify the forwarding state of a switch with an OF FlowMod even if it has not received any OF PacketIn before. In this case, we say that the controller installs forwarding rules **proactively**.

5 Your turn to play!

The goal of this assignment is to enable end-to-end connectivity in a very simple local network composed of 2 hosts and 4 OpenFlow switches (see Fig. 2). The traffic from host 1 to host 2 must traverse switch 2, while the



(a) **OF PacketIn**. When an OpenFlow-enabled switch does not know where to forward a packet, it can send it to the controller with an OpenFlow PacketIn message.

(b) **OF FlowMod**. Upon reception of an OF PacketIn message, the controller can decide to modify the forwarding state of a switch. This is done with an OpenFlow FlowMod message.

(c) **OF PacketOut**. After an OF PacketIn and a OF FlowMod, the controller can send the initial packet back to the switch. The packet, as well as the next packets of the same flow, are forwarded based on the new forwarding state of the switch.

Figure 5: In a Software-defined Network, the forwarding state of the switches can be updated reactively. When a switch does not know what to do with a packet, it sends an OF PacketIn message to the controller. The controller modifies the forwarding state of the switch with an OF FlowMod message and sends the packet back to the switch with an OF PacketOut message.

traffic from host 2 to host 1 must traverse switch 3. The forwarding decisions will be computed centrally by the SDN controller. You will have to program the forwarding rules of the switch *proactively*. But we also propose an advanced version where the forwarding rules can be computed *reactively*.

As Openflow switches are expensive devices, it is not possible to provide each group with a dedicated hardware-based network. Yet, it is very easy to provide you with a virtual network composed of software-based OpenFlow switches running in a Virtual Machine (VM).

Each group has its own Virtual Machine (VM). To make your life easier, you do not need to run a virtual network inside your own laptop. Instead, we have set up a remote VM for each group where your virtual network already runs and desperately awaits for your code to work. To access your VM, you'll use SSH. SSH is a UNIX-based command interface and protocol for securely getting access to a remote computer. It is widely used by system administrators to control network devices and servers remotely. A SSH client is available by default on any Linux and MAC installation through the Terminal application. For Windows user, a good and free SSH client is PuTTY[1].

Once you've installed a SSH client, use the following command to connect yourself to your VM:

```
> ssh -p X root@samichlaus.ethz.ch
```

where $X = 2000 + group\_number$. For instance, if you are group 7, use the following command:

```
> ssh -p 2007 root@samichlaus.ethz.ch
```

During your first login, you will probably be asked to accept the RSA key fingerprint of the server. Answer with "yes". You are then asked for a password. If you already have a group, you should have received the password by email. Otherwise, we will give you a temporary password during the exercise session. If you want to simplify the access to your VM, please use the SSH keys authentication[2], but *do not change the password*. To edit a file in a SSH session, you can use a text editor in the terminal, such as `vim` or `emacs`. You can also edit files locally on your laptop using your favorite text editor and remotely send them to your VM with `scp`, using the following command (e.g. for group 7):

```
> scp -P 2007 your_file root@samichlaus.ethz.ch:
```

Note the uppercase P parameter, as opposed to the lowercase one when using `ssh`.

[1] `http://www.chiark.greenend.org.uk/~sgtatham/putty/`

[2] Ask us if you want to know more about this, and please do not remove the content of the file ~/.ssh/authorized_keys

In this assignment, you will have to run several programs at the same time. To do that, you can open several terminals and connect to the VM with ssh on each of them. Another option is to use a terminal multiplexer such as `tmux`[3], which allows you to open several virtual consoles in one terminal window. `tmux` is already installed in the VM.

Build your virtual network. Once you are in the VM, the next step is to build a virtual network composed of switches and end-hosts. To do that, we use Mininet[4]: "Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine, in seconds, with a single command". Mininet provides a high-level python interface to build and run a virtual network. To ease your life, we provide you with a script **run_mininet.py** which uses this python interface and builds a virtual network with Mininet for you. To build the virtual network, run the following command:

```
vm> sudo python run_mininet.py
```

The python script will create the virtual topology depicted in Figure 2. Mininet uses Open vSwitch[5] to create switches in the virtual network. Open vSwitches support OpenFlow out of the box, so we will be able to configure their forwarding table as describe in section 4.

`Note`: you will see the following warning *"Unable to contact the remote controller at 127.0.0.1:6633"* because we have not started the controller yet.

Test the connectivity. Once the virtual topology is running, Mininet will provide you a Command Line Interface (CLI). To see the available commands, you can type `help`:

```
mininet> help
```

Try the `net` command which shows the current virtual topology. "Nodes" shows you the nodes in the network ("h" are hosts, "s" are switches). To test the connectivity inside your virtual network, you can use the command `ping`:

```
mininet> h1 ping h2
```

In this case, host 1 will ping host 2. If you want to do a full-mesh ping, you can use `pingall`.

```
mininet> pingall
```

Since no controller is running so far, the switches have an empty forwarding table, and the ping fails. At the end of this assignment, the goal is to be able to ping between host 1 and host 2. The `pingall` command should show you a full connectivity.

If you want to exit Mininet, use the following command:

```
mininet> exit
```

Warning: If you want to relaunch mininet, make sure to clean it up beforehand:

```
vm> sudo mn -c
```

Running the controller. Now that the virtual network is running, you need to run the controller. Luckily, you won't need to write the OpenFlow messages by yourself. SDN controllers have been designed to simplify the programmer's life by providing a high-level interface to communicate with the switches. In this assignment, we will use Ryu[6], a python-based SDN controller. The file `ryu_controller.py` is an example of an application you

[3] https://tmux.github.io
[4] http://mininet.org/
[5] http://openvswitch.org/
[6] https://osrg.github.io/ryu/

can run on top of Ryu. To run this application, use the following command (in another terminal or another `tmux` pane):

```
vm> ryu-manager ryu_controller.py
```

Initially, this command fails. Your goal is to complete the file in order to enable connectivity between host 1 and host 2 while making sure that the traffic from host 1 to host 2 traverses switch 2 and the traffic from host 2 to host 1 traverses switch 3.

`Note:` Python uses indentation to define a block of code. In contrast, programming languages such as C or Java use braces { }. In the source code we provide you, we used 4 spaces for each indentation. This is configurable on any text editor.

Populating the switches' forwarding table. With the Ryu controller, you can add a forwarding rule in a switch forwarding table using the `OFPFlowMod` function. This function takes the "match" and the "action" of the forwarding rule, builds the corresponding OFPFlowMod message, and sends it to the switch. There are other parameters, but you do not need to modify them for this assignment. To simplify the code, we have prepared the function `insert_l2forwarding_rule`. This function takes as input an object representing the switch, the destination MAC address to match, the incoming port to match and finally, the port where the matched packets are sent out. The function then builds the OFPFlowMod message and sends it to the controller.

6 Goal

The goal is to enable connectivity between host 1 and host 2. The traffic from host 1 to host 2 must traverse switch 2, while the traffic from host 2 to host 1 must traverse switch 3. We propose a simple way to do that (§6.1) and a more advanced one (§6.2). For each version, we have a prepared a file where you just need to complete the missing parts. To guide you through the code, we have commented it. Each part you need to complete is indicated with `### To complete ###`. When you are done, the following command on the mininet CLI should show 100% success.

```
mininet> pingall
```

6.1 Install the forwarding rules proactively

For this version, you must complete the file `ryu_controller.py`. In this simple version, you will install the forwarding rules proactively, meaning you do not need to wait for any OFPacketIn message. As such, you can install the forwarding rules directly when the switches connect with the controller, *i.e.,* in the function `_ev_switch_enter_handler`.

6.2 Advanced version: Install the forwarding rules reactively

If you are motivated, we propose you an advanced version, where you must install the forwarding rules reactively instead of proactively. For this version, you must complete the file `ryu_controller_advanced.py` and run the Ryu controller with this file: `ryu-manager ryu_controller_advanced.py`. To insert forwarding rules reactively, you will have to wait for OFPacketIn messages. By default, we have configured the switches to have one forwarding rule sending all packets to the controller in an OFPacketIn message (see function `_ev_switch_enter_handler`). Your goal in this advanced version is twofold:

`First`, upon reception of one OFPacketIn, the controller must install a new forwarding rule into the switch's forwarding table that will override the default one, such that the next packets will be forwarded by the switch itself and won't go to the controller anymore. As a result, you must now update the function `packet_in_handler`. You can still use the function `insert_l2forwarding_rule` to help you.

`Second`, you must avoid OFPacketIn messages as much as possible because each of them implies a time consuming interaction with the controller. Because you know the path a flow must follow, you can populate the forwarding table of all the switches in the path upon reception of the first OFPacketIn message for that flow.

7 General Information

7.1 This assignment is optional but highly recommended

This is a relatively easy assignment. It is optional (we won't grade you), but highly recommended as it will help you for the next two assignments, as well as the final exam. We will allocate two 2-hours practical sessions to help you fulfill the assignment. You should just attend one of the two sessions.

7.2 If you still have questions after the practical session: ask us on Slack!

Use the slack channel available at `https://comm-net17.slack.com/messages/exercises/`. Please do not ask questions in the `#general` channel, but use the `#exercises` channel instead. You can also send questions on Slack directly to Maximilian Schütte (`@mschuette`), Alexander Dietmüller (`@adietmue`), Thomas Holterbach (`@thomas`), Tobias Bühler (`@buehlert`), Rüdiger Birkner (`@rbirkner`) or Roland Meier (`@roland`).