
Caveat

This lab is NOT graded, so you do not need to submit anything. However, if you have trouble understanding what is required of you for this lab, or are unable to devote enough time to the lab, it probably means you lack the necessary background for the class, or have too many commitments this semester. In that case, please consider dropping the class.

Even if you think you have the necessary background for the class, you are encouraged to do this lab, since it will familiarize you with VM setup, Mininet, and other tools you need for subsequent labs, and will help you understand what software-defined networking means. These in general have a steep learning curve, so you're better off using the time allotted to lab0 to prepare yourself for subsequent labs.

Introduction

This project aims to introduce basic concepts that will be used in the future projects. The goal is that you become familiar with the infrastructure of Virtual Machines that will be used for this course and the network emulation tool Mininet[1]. In this project we will implement a local network with programmable switches. At the end of the project you will have set up a software-defined network (SDN) capable of interconnecting a limited number of end-hosts grouped on a sub-network (more on SDN in Section 2.2). We have not yet covered software-defined networking in class---we will later, starting with the [Casado09a] Ethane paper---but this project will give you a foundation to build upon when we read those papers, plus a comparison point with the non-software-defined legacy BGP networks we will also discuss in class.

Local networks and L2 forwarding

Local networks interconnect a limited number of end-hosts, generally tens to hundreds, through devices that execute L2 forwarding, most commonly switches (for wired networks) and access points (for wireless networks). To forward packets from source to destination, such equipments analyse only the MAC addresses and, based on internal rules, make the best decision.

In this assignment we are interested in working with wired local networks where end-hosts are connected by many switches. When the number of end-hosts increases, forwarding based on MAC addresses does not scale and networks are divided into sub-networks. Forwarding between subnets is done through L3 forwarders, called routers. Figure 1 illustrates a scenario of a network divided into three subnets. Each subnet contains on average 3 to 4 switches, and each switch connects a small number of end-hosts, only some of which are shown in the figure.

The solution to be implemented in this assignment will provide connectivity between all end-hosts on a single subnet. The location of the end-hosts (which switch they are connected to) is unknown. And connectivity must be maintained if end-hosts move to a different location.

---

[1] Mininet: An Instant Virtual Network on your Laptop (or other PC) - http://mininet.org/

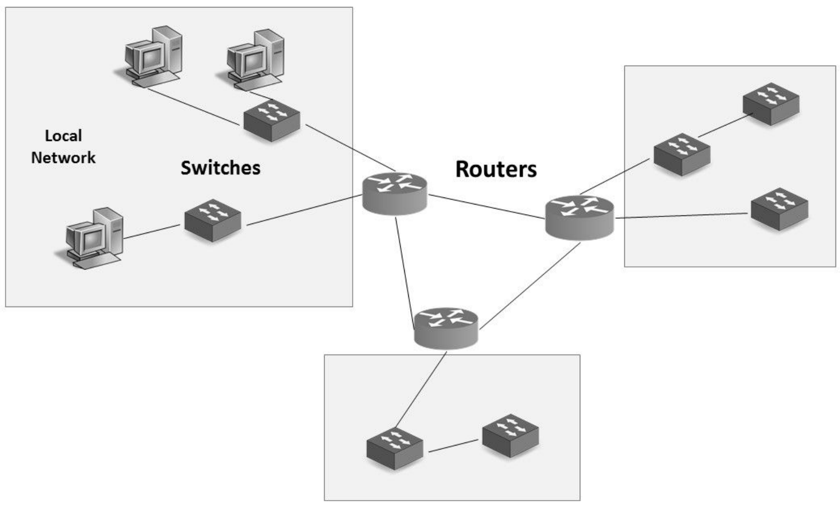


Figure 1 - Example of a network comprised of 3 subnets. Each subnet has a few switches that interconnect the end-hosts. Switches are L2 forwarders and routers are L3 forwarders

Computing a valid forwarding state in a local network

To create a valid forwarding state in a local network, we must avoid loops in the topology. Loops in an uncontrolled topology lead to a rapid depletion of network bandwidth (why?).

In general the operation of L2 switches can be split into three phases: (i) spanning tree calculation, (ii) flooding, and (iii) learning a forwarding table. We explain each phase below.

To avoid loops all switches need to negotiate a spanning tree[2] based on the physical topology and disable the links that are not necessary. The spanning tree calculation can be executed in a distributed or centralized manner, which is explained in detail later.

After the spanning tree calculation, all switches simply forward any received packet to all neighboring interfaces on the spanning tree, except the one where the packet arrived. This flooding process guarantees that the packet will reach the intended destination, but it generates unnecessary messages, and the performance of the network may drastically reduce as the number of end-hosts increases.

A way of improving performance is with a learning process and construction of a forwarding table. For all packets that a switch receives, it creates/updates a new entry on this table linking the incoming port with the source MAC address. In the future, the switch can use this entry to forward packets to that MAC address only via that port. This way, all packets with destination

[2] A *spanning tree* is a tree that connects all vertices in a graph. A *minimum spanning tree* has the minimal total weighting for its edges

address present in the forward table do not need to be flooded anymore.

Figure 2 summarizes this 3-phase process.

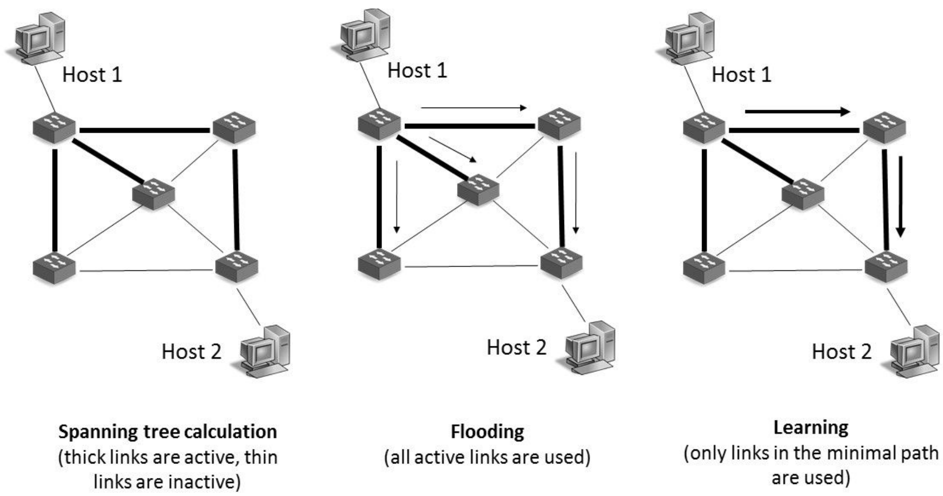


Figure 2 - Operation of a L2 switch

Distributed vs. Centralized management

The most common way of computing the spanning tree and creating the optimized forwarding table is employing distributed algorithms. Each switch executes a standardized Spanning Tree Protocol (STP) and, based on local information, populates the forwarding table. The STP was initially specified in IEEE 802.1D and a new optimized version (RSTP) has been incorporated into IEEE 802.1Q-2014 standard.

Even though distributed algorithms have many advantages, the incurred overhead in terms of control messages can bring disadvantages. The second way of operating an L2 local network is with a centralized architecture. Each switch communicates with a controller using standardized protocols (e.g. OpenFlow). The central controller has global knowledge of the location and status of the network (switches, end-hosts, etc.) and is able to compute optimal forwarding decisions. Such decisions are transmitted back to the switches and can be used to construct optimal forwarding tables. This paradigm is called Software-Defined Networking (SDN).

In SDN the forwarding decisions can be changed according to different optimization criteria, which creates a vast exploration field for researchers, students and network operators. In this assignment we will learn how to programmatically create the forwarding table on switches for a local network using the OpenFlow protocol.

Programming the forwarding tables with OpenFlow

OpenFlow (OF) is a standard communication interface to control the forwarding layers of an

SDN. It is currently defined by the Open Networking Foundation[3]. The communication between OpenFlow switches and the controller is bidirectional and uses TCP. The controllers should listen on TCP port 6653 for switches that want to set up a connection.

The forwarding tables in switches are usually implemented in hardware for fast lookup. Each entry in this table can be decomposed in two parts: match and action. The match is a combination of rules that filters packets based on several fields, from layer 2 to layer 4, as well as the port where the packet was received. All packets that match an entry on the forwarding table form a flow. For each flow a given action is specified, which can be simply to drop the packet, forward it to a specific output port, or encapsulate it and forward to the controller as an OFPacketIn message. Figure 3 depicts the structure of a forwarding table.

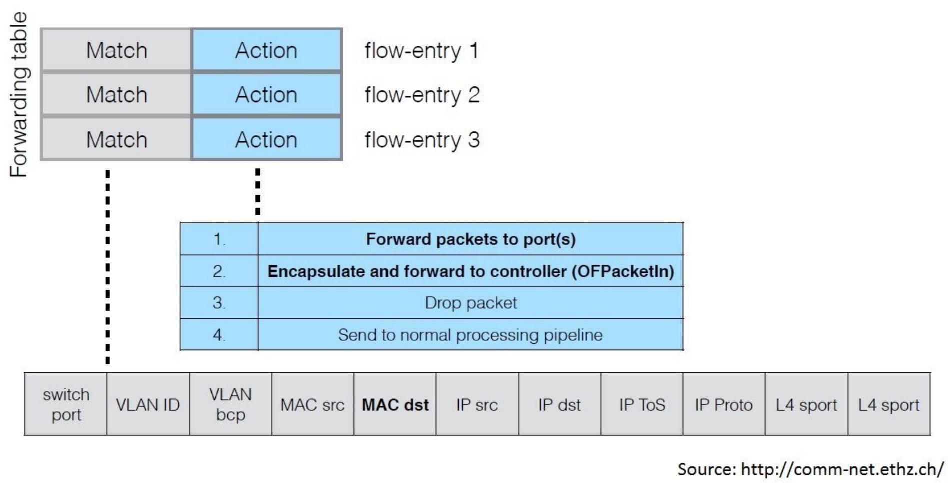


Figure 3 - Forwarding table on switches.

The OpenFlow protocol specifies several types of messages to be exchanged between switches and controllers. In this assignment we are interested in using three types of messages: OF_PacketIn, OF_PacketOut and OF_FlowMod[4].

- ***OF_PacketIn message***

This message is used when switches need to send data packets to the controller. This type of message is mainly used when a switch receives a packet that does not match any entry in the forwarding table. Upon receiving an OF_PacketIn message, the controller has to decide the

[3]OpenFlow - https://www.opennetworking.org/sdn-resources/openflow

[4] A complete list of OF messages and the complete protocol specifications can be found at https://www.opennetworking.org/technical-communities/areas/specification.

best forwarding decision and program the switch's forwarding table. This last step is executed with OF_FlowMod messages. After programming the forward table, the controller can send the packet back to the switch (using OF_PacketOut message), and the switch should now have a forwarding table entry to properly handle it.

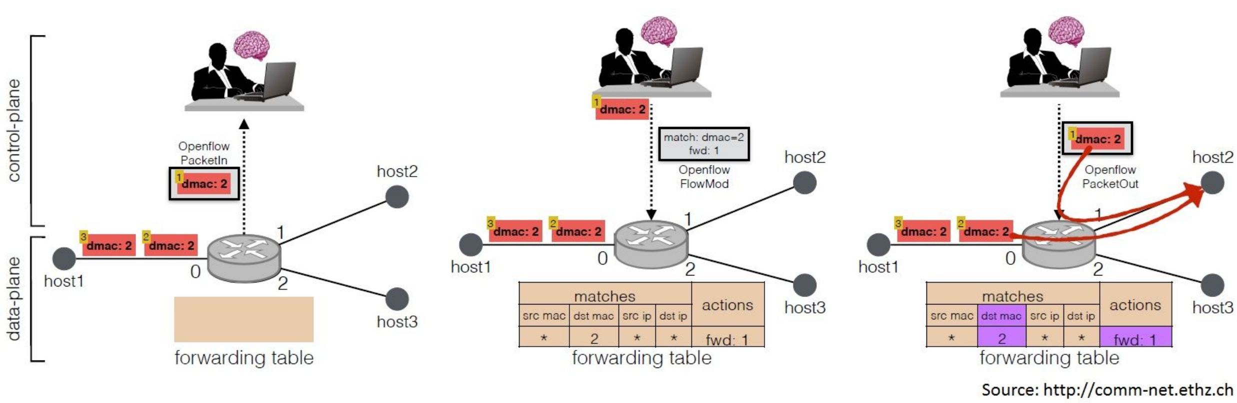
- ***OF_PacketOut message***

This message is used when the controller needs to send a data packet to a switch. This is generally employed in response to a previously received OF_PacketIn message, right after the corresponding entry is programmed in the switch's forwarding table.

- ***OF_FlowMod message***

This message is used when the controller needs to modify the switch's forwarding table. It can be used to insert, remove or update an entry, and it must specify both a match rule and an action.

Figure 4 shows an example of OpenFlow in action. In Figure 4a, the switch needs to forward a packet with destination MAC address 2, but its forwarding table is empty. As a consequence, the packet is encapsulated into an OF_PacketIn message and forwarded to the controller. The controller knows where address 2 is located and decides to insert an entry that forwards all packets with this destination address to port 1. The OF_FlowMod with the corresponding match and action fields is sent to the switch, as in Figure 4b. Finally, the packet is forwarded back to the switch (inside an OF_PacketOut message), which is finally able to transmit it to the destination node. The switch can forward subsequent packets that also match the entry (e.g., packets 2 and 3) directly without forwarding them to the controller.



(a) OF_PacketIn: when an OpenFlow switch does not know where to forward a packet, it sends it to the controller using

(b) OF_FLowMod: when the controller received an OF_PacketIn message it decides to modify the forwarding table of a switch. This is done

(c) OF_PacketOut: when the new forwarding entry is added to the switch, the current packet is sent back from the controller to the switch using OF_PacketOut

an OF_PacketIn message.	with an OF_FlowMod message.	message. New packets will be properly forwarded without interference of the controller.

Figure 4 - Programming a switch with OpenFlow

Getting Started

If you have not done so already, now would be a good time to follow these instructions to install the software that you will need for this and subsequent labs.

The Mininet emulator

Mininet is an open-source network emulator used for research and experimentation. It allows you to create a virtual network composed of hosts, switches, controllers and links. It runs on Linux and offers support for many protocols, including OpenFlow. Instead of spending thousands of dollars on expensive switches and hours setting up a real network - which would be really cool, by the way! - we can virtualize everything and work on a fully functional network in minutes.

You can start a network with Mininet via its console interface or with a high-level Python API. We are going to explore the console interface, but for the assignment you are supposed to use a script provided by us to create your network.

Mininet is executed with command sudo mn. After you have logged into your VM, try to execute the following sequence of commands.

```
$ sudo mn

mininet> help

mininet> nodes

mininet> net

mininet> h1 ifconfig -a

mininet> h1 ping -c 10 h2

mininet> pingall

mininet> xterm h1

mininet> exit
```

If Mininet is executed without arguments (like it is done above), it creates a simple topology with two hosts connected to one switch and one controller. The command help displays all commands that can be used on Mininet. The commands nodes and net display, respectively, information about the nodes on the network (hosts, switches and controllers) and the network interfaces of these nodes. You can get more information about the network interfaces with command **`<host> ifconfig -a`**. Command **`<host1> ping -c 10 <host2>`** sends 10 ping messages (ICMP) from host h1 to host h2, and **`pingall`** tests the connectivity between all hosts on the network. The command **`xterm <host>`** opens up a terminal interface to the end-host and allows you to execute any Linux-like command on it. Finally, command **`exit`** closes the emulator.

You should practice some of these commands and explore the features of Mininet. A full documentation can be found at https://github.com/mininet/mininet/wiki/Documentation.

Using Mininet with Python API and Ryu controller

Mininet provides a high-level Python API for automating the creation and configuration of networks. You can find the API reference at http://api.mininet.org. We have provided you with a script (start_mininet.py) which uses the Python interface and builds a virtual network based on a given topology of switches.

You can build the network and run it with the following command:

```
$ sudo python start_mininet.py
```

The script creates the switches and links based on the file topology.txt[5]. One end-host is connected to each switch and, finally, a remote controller is also created. The first time that you run the command above you should get an error saying that Mininet was “Unable to contact the remote controller…” and your network should not run properly. The goal of this assignment is to implement a controller for the network we are building.

Important: after exiting Mininet, you have to clean it up before re-launching with command `$sudo mn -c`

Mininet supports a number of SDN controllers. We are going to use Ryu[6] in this assignment. Ryu is a Python-based SDN controller that supports many different protocols, including OpenFlow. You can find a tutorial on how to program Ryu at https://github.com/osrg/ryu/wiki/OpenFlow_Tutorial.

We also provide you with a simple Ryu controller in **`ryu_controller.py`** that you can use to start off your assignment. Try this simple Ryu controller with the command below and re-run the Mininet network (with start_mininet.py script). You should be able to ping all nodes.

```
$ ryu-manager ryu_controller.py
```

The simple controller simply responds to each OF_PacketIN message with a OF_PacketOut

[5] The topology file must be named topology.txt. We suggest you to open and read the comments, as you will have to change it for this assignment

[6] https://osrg.github.io/ryu/

message to be flooded to all ports of the switch. This action is taken by method packet_in_handler(), which is called every time an OF_PacketIn message is received.

This solution works for a topology that doesn't have loops. As we will see in the questions, this controller is not effective for topologies with loops.

Python networkx library

To ease the task of creating a spanning tree and calculating the best action that should be taken by the switch, we use a Python library called networkx[7], which implements a number of operations and data structures on graphs.

When our controller (ryu_controller.py) loads the topology, it adds two attributes to the nodes. Both attributes are dictionaries. The first is called ports, and it has the node IDs as key mapping to the port number of the switch that should be used to reach the node. The second is called mactoport and it maps the node MAC address (used as key) to the number of the port on the switch.

You can use these data structures to facilitate the implementation of the forwarding rules.

The Lab

The lab is divided in three parts. In the first part you will evaluate the performance of our simple Ryu controller (ryu_controller.py). In the second part you will implement a learning controller. And in the third part you will implement a spanning tree calculation algorithm for your learning controller.

Part 1: Evaluate the performance of simple Ryu controller

First replace "topology.txt" with "topology_bus.txt", but still keep its file name as "topology.txt". Then run the network with the default topology without loops (the first topology on the file topology.txt). Execute a ping from host 1 to host 4 transmitting 10 packets. Now, repeat the same experiment with the second topology with loops (you have to edit the file topology.txt and uncomment the second topology). What differences do you see, if any, between the two topologies? Try to find out why this difference arises.

Part 2: Implement a learning controller

You have to change the Ryu controller and make it work on any type of topology (including but not limited to the given topology with loops). You need to re-implement the method **`packet_in_handler()`** on **`ryu_controller.py`** to create a learning controller. Your controller has to insert the correct entries on the switch using [`OF_FlowMod`] messages.

[7] https://networkx.github.io/

You should use the spanning tree calculated by method **`compute_spanning_tree()`** and stored at variable **`ST`**. For this part, you do not need to modify the **`compute_spanning_tree()`** method, just use the tree that is given to you. You have to make sure that Mininet achieves 100% success when command pingall is executed, on any topology including but not limited to the included ones.

Debugging hint: While developing this part and the next, you might need to check the flow entries installed in a switch. To do this, you can use the **`dpctl`** utility installed in your VM.

Part 3: Implement a spanning tree algorithm

You have to re-implement the function **`compute_spanning_tree()`** in **`ryu_controller.py`** and replace the line where networkx library is used by your own implementation of any spanning tree calculation algorithm. ***You cannot use any other library that performs the calculation for you***. But you can use any algorithm that you may find easy or interesting to use. And there is no need to worry about efficiency, the only requirement is that your algorithm correctly calculate a valid spanning tree. You have to make sure that Mininet gets 100% success when command pingall is executed, on any topology including but not limited to the included ones.

Concluding Remarks

To be completely sure that you have implemented the controller correctly, you should run the following command:

```
$ sh run_tests.sh
```

If you pass all these tests by the assigned deadline, you likely have the necessary background to do the labs in CS 551. If not, please consider dropping the class.