# Programming Assignment 1

AY 2025 – 2026 / Semester 1

## Overview

In this assignment, you will learn to use the Mininet network emulation environment to set up a virtual network, and program the OpenFlow controller Ryu to implement several network applications. Through this project, you will learn (1) to setup virtual networks by creating virtual hosts, switches and links and connecting them together, (2) to implement a simple learning switch using Ryu, (3) to implement the Spanning Tree Protocol to handle switch loops in Ryu, and (4) to implement fault tolerance using events handlers in Ryu.

**DEADLINE: October 24th, 2025 23:59 (Week 10 Friday)**

## Understanding Provided Files

You will receive this assignment as two ZIP files and an x86 VM image.

You should be able to find the ZIP files, as well as instructions to retrieve the VM image from the announcement.

If you choose to use the VM image, please import the image to VirtualBox or VMWare (we have tested both on x86 platforms). The password is "**`P@ssw0rd`**".

To install all the dependencies for PA1, we have provided a set of installer scripts for you to set up the environments. The scripts have been tested on the VM we provided and the VM SoC provided. The source templates and checkers for the assignment come in a separate ZIP file.

You will need to edit **`topology.py`** for task 1, **`simple_switch.py`** for task 2, and **`stp_switch.py`** for task 3&4 to complete this assignment.

There is also a README file in the folder that illustrates more on the file contents.

## Submission Guideline

You need to submit to Canvas Programming Assignment 1 a zip file named with your student ID (e.g. `A1234567X.zip`) containing the three files edited. A script **`packer.sh`** has been provided to you. After you are happy with your code, call **`./packer.sh <your_student_id>`** to generate the submission file.

## Task 1: Building a Virtual Network (30%)

In this section, your task is to build virtual networks in the Mininet network emulation environment. The network topologies are given as input files. Please edit **`topology.py`** to complete the logic for processing input and create topology accordingly in Mininet.

Quote from Mininet GitHub README: *Mininet creates virtual networks using process-based virtualization and network namespaces - features that are available in recent Linux kernels. In Mininet, hosts are emulated as bash processes running in a network namespace, so any code that would normally run on a Linux server (like a web server or client program) should run just fine within a Mininet "Host". The Mininet "Host" will have its own private network interface and can only see its own processes. Switches in Mininet are software-based switches like Open vSwitch or the OpenFlow reference switch. Links are virtual ethernet pairs, which live in the Linux kernel and connect our emulated switches to emulated hosts (processes).*

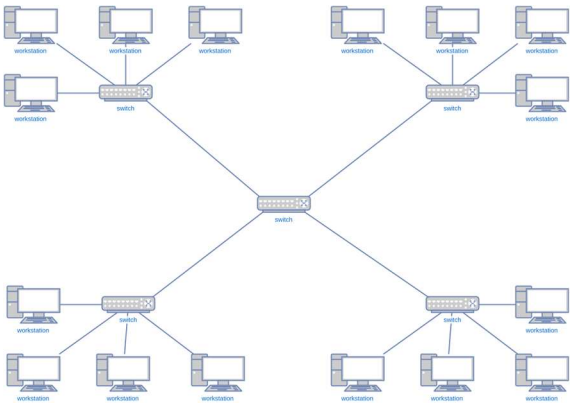

**Figure 1:** Illustration of the star topology

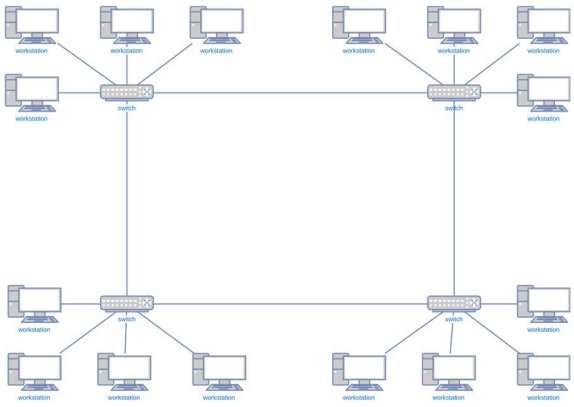

**Figure 2:** Illustration of the ring topology

Figure 1 is an example of a network topology, consisting of 16 **hosts**, 5 **switches** and 20 **links** that connect them together. In this network, the 16 hosts are connected with 4 switches (with dpid from 1 to 4) by 16 links. The 4 switches are then connected to a central switch (with dpid 5) by 4 links. The input file for this topology is **`star.in`**.

Figure 2 is another example. It consists of 16 **hosts**, 4 **switches** and 20 **links** that connect them together. In this network, the 16 hosts are connected with 4 switches (with dpid from 1 to 4) by 16 links. The 4 switches are then connected in a loop by 4 links. The input file for this topology is **`ring.in`**.

You need to build a network with the topology and specifications described in an provided input file **`topology.in`**. The first line of the file has 3 integers $N$ and $M$, $L$ indicating that there are $N$ hosts $(h_1, h_2, \dots, h_N)$, $M$ switches $(s_1, s_2, \dots, s_M)$ and $L$ links in the network. The following are $L$ lines of tuples in the form of $dev1, dev2$ that describe the links, meaning that $dev1$ and $dev2$ are connected via a bi-directional link. A partial input file for the network in Figure 1 looks like follows:

16 5 20

h1,s1

h2,s1

h3,s1

h4,s1

h5,s2

...

s1,s5

…

The sample input files **`star.in`** and **`ring.in`** for the network in Figure 1 and 2 are provided for your reference (a smaller version of each is provided for debugging purposes). You will also get a template script **`topology.py`**, based on which you can add your own code to create the hosts, switches and links in Mininet.

To test your implementation, a script **`topo_checker.py`** is provided; you can use this script by calling `` `python checker/topo_checker.py` `` from the home directory. The script will check your implementation on the two provided topologies: **star** and **ring**. **NOTE THAT THE SCRIPT WILL CREATE A FILE `topology.in`, DO NOT PUT NAME YOUR CUSTOM TOPOLOGIES THAT WAY AS IT WILL BE OVERWRITTEN!!!** More inputs will be used in actual grading. **DO NOT HARD-CODE YOUR SOLUTION!** If you want to manually run your implementation, copy any input files (`star.in`, `ring.in`, or maybe even something you write yourself) into a file named `topology.in` and run `` `sudo python3 topology.py` ``.

## Task 2: Fine-Grain Learning Switch (35%)

Building a virtual network is not enough for the network to operate. In order for the hosts to communicate with each other, the switches need to know how to correctly route a packet to its destination. In this part, your task is to implement the application of self-learning switches using the Ryu controller, which enables the switches to learn how to route packets without any prior knowledge of the network.

Your task is to complete the **`_packet_in_handler`** function to implement the learning behavior. A template, **`simple_switches.py`** has been provided for your reference. The template contains a fully implemented event handler **`switch_features_handler`** to demonstrate how a flow entry can be installed. Please pay attention to the links to the documentation in the comments. The **`add_flow`** helper function is also fully implemented for you. You are highly encouraged to read through the code and understand its content. You can (although strongly advised against) deviate from the template but your implementation **MUST** have the learning behavior **AND** install flow rules so that the subsequent packets are not forward to the control plane.

The key idea is to record the MAC address and the corresponding port number when receiving a packet that does not match any entry in the forwarding table. Thus, the switches can gradually learn the port number for reaching each host in the network. In the following part, we provide a simple example for illustration.

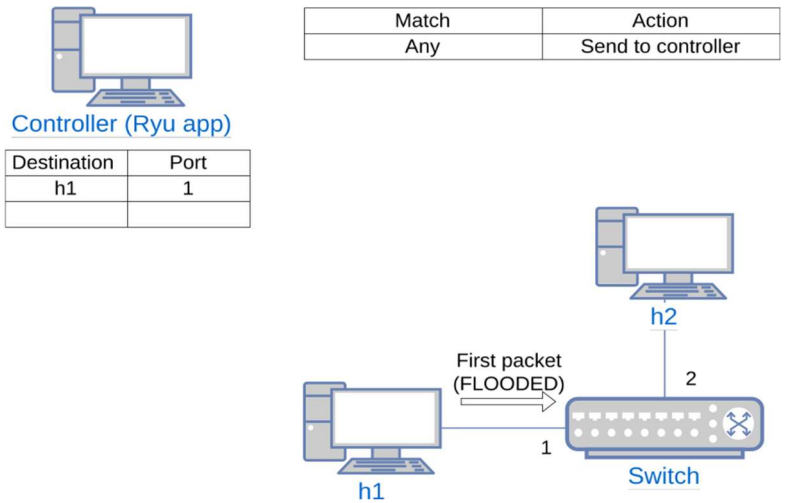


**Figure 3.** Example of a self-learning switch: First packet

Initially, we have no knowledge of the 2 computers h1 and h2 that connect to the switch as is shown in Figure 3. When h1 sends a frame to h2, we do not know how to route the frame. We then **flood** this frame to all the

ports. Also, we have learned that h1 can be reached by port 1. While this information can be immediately installed as a flow rule on the switches, in SDNs it is often desirable to have **finer-grain control of the forwarding behavior**. Specifically, we want the flow rule to be **host-to-host specific**. Therefore, we will keep this information only in the controller for now.

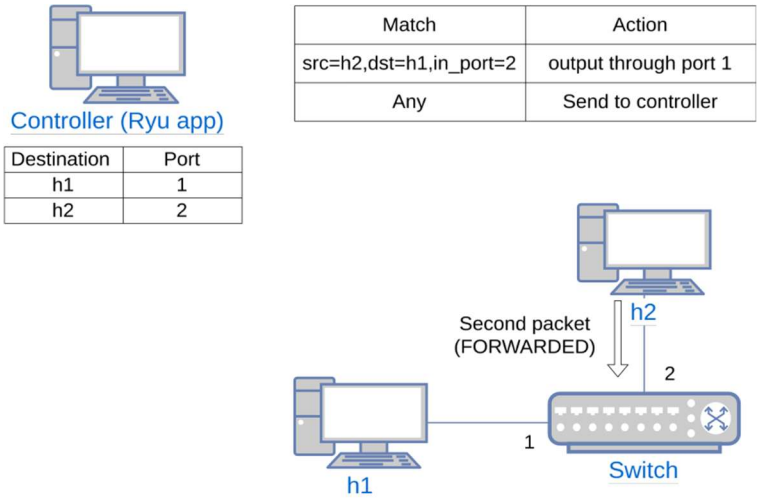


**Figure 4.** Example of a self-learning switch: installing a flow rule on response.

As shown in figure 4, when h2 sends a reply frame to h1, we now know that h1 can be reached from port 1. We then add this as a flow rule to the switch that specifies: **if a packet sent from h2 to h1 is received from port 2, send the packet out from port 1**. This kind of fine-grained control offers the controller more options on manipulating traffic. Furthermore, now we know h2 can be contacted from port 2; we keep this information in the controller for now.

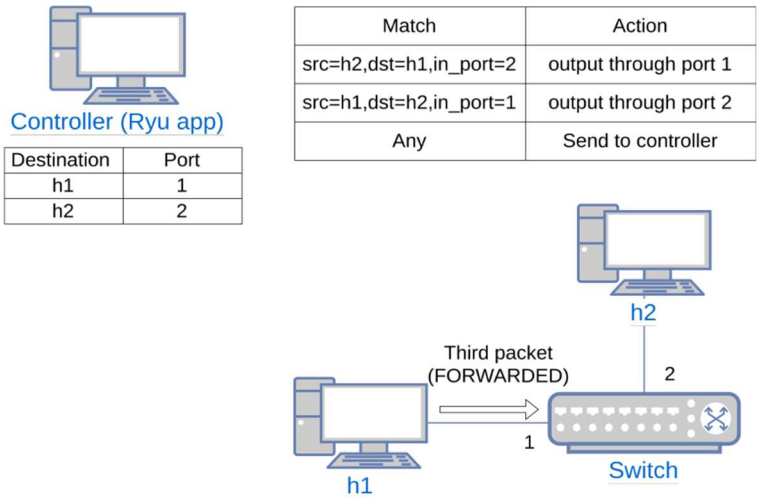


**Figure 5.** Example of a self-learning switch: installing a flow rule on response.

Finally, as shown in figure 5, when h1 sends another packet to h2, we now know that h2 can be reached from port 2 and add this as a flow rule to the switch that specifies: **if a packet sent from h1 to h2 is received from port 1, send the packet out from port 2**.

**DO NOT HARD-CODE YOUR SOLUTION!** If you want to manually run your implementation, set up a virtual network (that does not have a loop) as in task 1 and in a separate terminal, run `` `ryu-manager learn_switches.py` ``.

## Task 3: Spanning Tree Protocol (0%)

Have you noticed that your task 2 implementation is not tested on the ring topology (and that we mentioned the virtual network you setup cannot have a loop)? While a learning switch is simple and easy to implement, it does not work on topologies containing loops as the initial broadcast will quickly (within seconds) lead to a broadcast storm that saturates all devices with repeated dataframes. You can try this by simply setting up a ring topology and running the learning switch. (**WARNING: This will likely hang your VM, save your work before trying!**)

The solution to this problem is the Spanning Tree Protocol (STP), defined in IEEE 802.1D standard. The core idea of the STP is to disable certain ports by configuration so that the topology left forms a spanning tree (and thus no loops). When initiated, the switches will first elect a root bridge (switch), all other switches will then form a spanning tree with the root bridge as the root. All links that are not part of the spanning tree will be disabled to prevent loops.

Note that STP is only for loop avoidance. You will still need the learning behavior implemented in task 2 to make you network functional.

Ryu has provided an implementation of STP in its library. In this task, you will use this library to avoid broadcast storms in case there are loops in the network topology.

Your task is to implement the learning behavior in the **`_packet_in_handler`** in **`stp_switches.py`**. Try to reuse the code you wrote for task 2 when completing the implementation. You can (although advised against) deviate from the template but your implementation **MUST** have the learning behavior **AND** install flow rules so that the subsequent packets are not forward to the control plane.

To test your implementation, a script **`stp_switch_checker.py`** is provided. The script will check your implementation on the ring topology. More topologies will be used in actual grading. **DO NOT HARD-CODE YOUR SOLUTION!** If you want to manually run your implementation, set up a virtual network with loops as in task 1 and in a separate terminal, run `` `ryu-manager stp_switches.py` ``. Note that STP needs time (around 1 minute, might be more) to run so wait until you see no more "topology change" log entries in a while before you start playing with the network.

## Task 4: Topology Change (35%)

In reality, the topology of a network might change at any time. A link might go down due to hardware failure or someone might plug a new network cable which introduces a loop. In either case, the STP will need to be run again, which might lead to a different effective topology. The old MAC table might become incompatible with the new topology after change and thus needs to be re-learned.

Your task is to add an event handler for the **`stplib.EventTopologyChange`** event to the **`stp_switches.py`** file which you used in task 3. The handler should react to topology change by deleting all installed flow rules for forwarding. (The `stplib` will automatically re-run STP when the topology changes) Note that you need to keep the table-miss entry around so that you can still get the learning behavior in the future.

**DO NOT HARD-CODE YOUR SOLUTION!** If you want to manually run your implementation, set up a virtual network with loops as in task 1 and in a separate terminal, run `` `ryu-manager stp_switches.py` ``. After STP finishes running (you can verify by pingall, or check the ryu log), disconnect some previously used links between switches by typing `` `link sX sX down` ``.

## Got Questions

If you encounter a problem, please post the problem on Piazza. Do not include your answer as it will be seen publicly and will be considered a violation of academic integrity (follow the common sense).

If you encounter a problem which you believe discussing the problem requires sharing of answers, please email your TA. Please begin the title of the email with "[CS4226 PA1]". We might repost the problem on Piazza if we believe it is common. Note that we will NOT debug your code, nor will we guarantee to handle environment issues if you are not using the distributed VM by us or SoC.