Assignment 3: An Introduction to the World of SDN

COL 334/672, Diwali'25

October 7, 2025
Deadline: October 13, 2024

Goal: The goal of this assignment is to give you hands-on experience with software-defined networking. In particular, you will learn how to implement basic network policies using OpenFlow-like APIs. You will use Mininet (installed in Assignment 2) along with Ryu controller for implementation and experimentation.

Part 1: Hub Controller and Learning Switch (20%)

In this part, you will compare the performance of two types of controllers: a *Hub Controller* and a *Learning Switch*.

- **Hub Controller**: Redirects all traffic from a switch to the controller. The controller maintains a MAC address table of hosts connected to ports. If the destination MAC is already known, the controller instructs the switch to forward the packet to the corresponding port; otherwise, it floods the packet. Importantly, the MAC rules are stored only at the controller and are not installed as flow rules on the switches.
- **Learning Switch**: Learns MAC-to-port mappings from incoming packets and installs flow rules directly on the switches. Once rules are in place, the switches can forward packets to the correct port without contacting the controller. Only packets that do not match any existing flow rule are sent to the controller.

Steps:

1. Begin by implementing a Controller Hub and Learning Switch.
2. You are given a network topology file implementing the topology showing in Figure 1. Run the controller against the given network topology file and answer the following questions:
   (a) Run `pingall` for both controllers. Record the installed rules in the switches in your report and explain your observations.
   (b) Run a throughput test between *Host 1* and *Host 3* using `iperf` in both cases. Report the observed values and explain the differences in speed values between the Hub Controller and the Learning Switch.

Hint: You may use the example code in the Ryu codebase as a starting point. This part is relatively straightforward and is intended to help you understand the basic working of the Ryu controller.

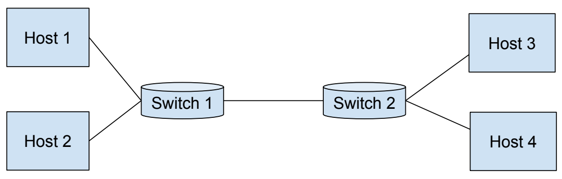

Figure 1: Example Network Topology for Part 1

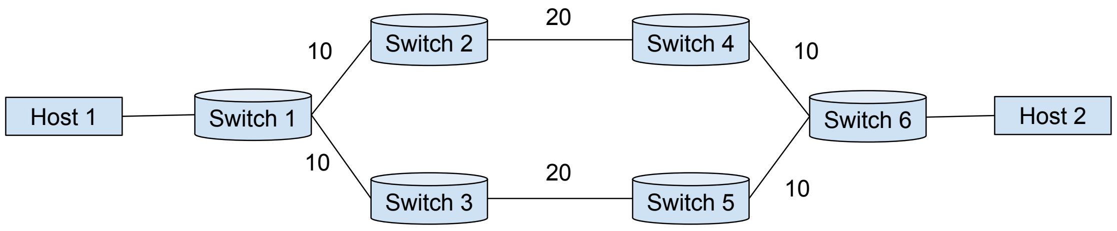

Figure 2: Example Network Topology for Part 2

Part 2: Layer2-like Shortest Path Routing (30%)

In this part of the assignment, you will implement a custom Ryu controller that performs shortest-path routing across the network while still performing L2-like forwarding (i.e. assume there is only one subnet). The controller will be provided with a network topology file containing switches, links and their costs. You can assume the entire network is part of a single subnet. For reference, we have provided a topology file corresponding to Figure 2.

Your controller should:

- Read the network topology and link cost matrix, and represent it as a weighted graph.
- Use Dijkstra's algorithm to compute the shortest path between the source and destination for each flow. You may use the Python's built-in library for Dijkstra's algorithm, but make sure you understand how the algorithm works.
- In addition, implement a load-balancing strategy when multiple equal-cost paths (ECMP) exist. You may randomly select among equal-cost paths. The ECMP policy (whether to use it or not) should be configured based on the flag argument in the input configuration file (see example file).
- Install OpenFlow rules along the chosen path for each flow.

Testing and Measurement:

1. Run `iperf` between between H1 (client) and H2 (server) in Figure 2 for 10 seconds with 2 parallel TCP connections (the number of connections can be specified in the command-line arguments while starting an `iperf` client).
2. Run the above experiment twice, once with the `ECMP` flag set to `true` and once with it set to `false`.
3. Record the throughput and the flow rules installed in both runs. Summarize and explain your observations in the report.

Bonus (10%): Instead of randomly selecting among the equal-cost paths, implement a weighted load-balancing strategy that selects a path for each flow based on the current link utilization. For example, if one path is lightly loaded while another is heavily loaded, new flows should be assigned to the lighter path.

To validate your implementation, you should modify the experiment above to run `iperf` with UDP flows that generate different loads on the network. Validate whether new flow rules are installed on paths in weighted manner. In your report, explain your load balancing mechanism and the validation results, comparing them with the random selection methodology.

Part 3: Layer3-like Shortest Path Routing (25%)

In this part of the assignment, you will implement a custom Ryu controller that performs shortest-path routing across the network, similar to a layer-3 switch (i.e., the network may contain multiple subnets). The controller will be provided with a configuration file containing information about the switches, their subnets, and link costs. An example configuration file has been shared with you. Note that this information is typically available to a network operator. In addition, an example topology file, similar to the one in Figure 2, has also been provided.

Your controller should:

- Read the network topology and cost matrix, and represent them as a graph.
- Use Dijkstra's algorithm on this weighted graph to compute the shortest path between the source and destination switches.
- Install OpenFlow rules on the switches based on the computed shortest path. Note that you don't need to implement ECMP in this case.
- Rewrite Ethernet headers where necessary so that the openflow switches behave like routers and enable inter-subnet communication. The controller should also decrement the TTL and drop packets if TTL becomes zero, just like a regular L3 switch. You should appreciate that an OpenFlow switch is not a inherently an L2 or L3 switch; it simply operates on packet headers, *matches* fields, and takes *actions*.

Experiment and Reporting: For this part of the assignment, you are required to test your controller application with the sample network topology as follows:

- Ping `h2` from `h1` 5 times. Report the results from ping and explain them briefly.
- Report the flow rules installed on each switch by your computer. You may use `ovs-ofctl dump-flows <switch>` or equivalent to list the rules.
- Also report any assumptions you made.

Part 4: Comparison with Traditional Routing (OSPF)(20%)

In this part, you will compare your Dijkstra-based SDN controller from Part with traditional shortest-path routing using OSPF. Running OSPF on Mininet is non-trivial, so you are provided with most of the starter code needed to set up the OSPF network. Your main task is to run the code by installing the required dependencies.

Note about the starter code: In this setup, the OSPF routers are actually Mininet hosts (and not OVSSwitches) because an OVSSwitch does not natively support OSPF. The startup code uses the FRR routing library, which in turn runs Zebra and OSPF deamons, to provide full routing functionality. Zebra is used to manage routing tables and interfaces, while the OSPF daemon computes shortest-path routes dynamically and updates Zebra with the routing information. You will need to install these on the BaadalVM (the host machine running Mininet). These packages are then available to the Mininet hosts. See the `readme` file in the starter code for the exact steps. Make sure you read the code.

Warm-up experiment: Conduct the following experiment: Run `iperf` between *Host 1 (h1)* and *Host 2 (h2)* in the provided topology once the forwarding rules are installed and OSPF has converged. Record the results of the iperf and the forwarding rules that have been installed on Switch 1 and Switch 6 in Figure 2.

Comparison under link failure

In the next part, evaluate how your Dijkstra-based SDN controller and the OSPF setup respond to a link failure. Consider the following experiment:

1. Run `iperf` between h1 (acting as client) and h2 (acting as server) for 15 seconds. You should wait for the OSPF to stabilize in the beginning before starting `iperf`.
2. **Emulate link failure**: Two seconds into the test, one of the links in the network path goes down for 5 seconds. The logic is implemented in `link_flap_exp` function located in `runner.py` for OSPF. You can write similar code for the SDN controller.

3. Both the SDN controller and the OSPF should detect the failure/recovery and calculate an alternate path. While OSPF performs this automatically, the corresponding logic needs to be implemented in the SDN controller.
4. Record and compare the throughput and the convergence times. You can infer the convergence time either from the per-second throughput reported by `iperf` or by directly inspecting the updated switch rules and routing tables. For OSPF, you can do this by logging and inspecting the control packets (e.g., OSPF LSA) exchanged during failure and convergence on one of the routers. Explain your results.

Setup and Resources

You can use the same installation of Mininet as in Assignment 2. In addition, you will need to install Ryu for this assignment. The installation instructions and tutorial for Ryu can be found here: `https://ryu.readthedocs.io/en/latest/getting_started.html`

Some useful commands for Mininet:

- Ping between hosts h1 and h2: `h1 ping h2`
- Any command you want to send to a host, say h1: h1 cmd
- To open a new terminal for host h1: xterm h1. If you are ssh-ing to a VM on Baadal, make sure you set up x11 forwarding.
- Print the rules currently installed on switches: dpctl dump-flows
- Running a ryu app: ryu-manager app.py

Submission Instructions

Your submission should contain a single PDF (other formats will not be graded) called `report.pdf`. In addition, you should submit the following:

- **Part 1**: Attach the screenshots of the rules as well as the ping/iperf results in the report PDF. In addition, submit two controller applications, naming them `p1_learning.py` and `p1_hub.py`.
- **Part 2**: Submit the controller application named `p2_l2spf.py`. Include any assumptions, results in the report. If you are attempting the bonus part, submit it as a separate file named `p2bonus_l2spf.py`
- **Part 3**:Submit the controller application named `p3_l3spf.pdf`. Include the results in your report.
- **Part 4**: Submit the code for the controller `p4_l3spf_lf.py`. This code may be same as Part 3 if you were already handling link updates earlier. Include the results as well as supporting logs in your report.

You should submit a single zipped folder containing all the code as well as the report. Name it `<entry_no_1>_<entry_no_2>.zip`, i.e., the names of your and your partner's entry number. *Please note: Only one submission per group is required.*