Project: SDNs

Contents

1 Introduction

Software-defined networking, or SDN, is a new paradigm for running networks. As you have seen so far in this course, the network is divided into the control and data planes. The control plane is the set of protocols and configurations that set up the forwarding elements (hosts, switches, and routers) so that they can forward packets. This includes, for example, ARP resolution, DNS, DHCP, the Spanning Tree Protocol, MAC learning, NAT and access control configuration, as well as all of the routing protocols.

Usually, switches and routers have to run all of these protocols, detect topology changes, issue heartbeats, manage caches, timeouts, etc. Meanwhile, in many cases network administrators achieve desired goals with the network indirectly, by tweaking parameters in the routing protocols like link weights and local BGP preference. While the data plane is nicely organized in the familiar layered scheme, the aggregate structure of the control plane is a lot less clean.

SDN is a radical departure from this organization. The main idea is a separation of the control plane from the forwarding elements. SDN switches and routers do not run control plane protocols and mostly only forward packets based on matching of packet predicates to a set of forwarding rules. They export a simple API to configure these rules, as well as some feedback about current and past packets. The currently accepted standard for this API is the OpenFlow protocol, which has been implemented by dozens of switch vendors and has fostered a rich software ecosystem. The intelligence of the control plane is (logically) centralized in a *network controller*. The controller decides which rules to install based on its configuration, and on a global view of the network topology and flows. As described, SDN is suitable for networks under a single administrative domain (e.g., the network in a single AS), but there are ongoing research projects to use its flexibility across domains, integrating with and perhaps even replacing BGP.

In this assignment, you will implement the logic in such a controller to manage the efficient switching of packets among hosts in a large LAN with multiple switches and potential loops. You will write the code for an SDN controller application that will compute and install shortest path routes among all of the hosts in your network.

2 Overview

As stated above, your task is to write code that will run in an SDN controller to compute, install, and maintain shortest paths on a large local area subnet. Given a packet destined to an MAC address, your network will use a shortest path among the switches to deliver it to the host.

The hosts in the project will not be changed in any way, it is only the switches in the network that will behave differently. While all the hosts in this project will be in the same subnet, we will not use any broadcasts, including for ARP. Instead, the controller will keep track of the location of each host and set up each switch's forwarding tables accordingly.

As usual, when a host wants to send a packet to the IP address of another host in the network, it will first issue an ARP request. However, when this reaches the first switch, the switch will send the ARP request to the controller instead of broadcasting it. The controller, which is aware of the network topology, will create an ARP response for the switch to reply to the sender. In addition, the controller will install rules on each switch to configure the forwarding table for each destination MAC address.

2.1 Architecture

To build this project, you will develop and test your controller in an emulated network running in a VM. To do this, we will use the Mininet[1] emulator, which uses lightweight network namespaces to emulate arbitrary topologies of emulated SDN switches and Linux hosts. The switches will run the open source software switch Open vSwitch[2], which implements the OpenFlow protocol.

For our controller, we will use Ryu[3], a relatively lightweight controller platform written in Python.

There are several versions of the OpenFlow protocol with varied feature sets. For this project, we will be using OpenFlow 1.0.

3 Shortest-path Switching

Your task is to build a global shortest-path switching table and install forwarding rules on the switches to implement these paths. You will build this table on the controller based on global topology information the controller gathers.

Unlike traditional L2 switches or L3 routers, SDN switches do not have dedicated MAC learning tables or routing tables. Instead, SDN switches use a more general *flow table* structure which can replace these and other constructs. Each entry, or *rule*, in a flow table contains a set of match criteria (based on the fields of Ethernet, IP, TCP, UDP, and other headers) that select specific packets and contain a set of *actions* that should be taken for each packet that matches the rule.

Your switching module will install entries that match packets based on their destination MAC address (and Ethernet type), and execute an output action to send the packet out on the correct port to reach its destination.

The match criteria serves the same purpose as the destination and mask fields in a traditional route table, and the output action serves the same purpose as the interface field in a traditional route table. In the aggregate, your network will resemble a network in which all switches have converged with the spanning tree protocol and MAC learning, with one important difference: your topology is not constrained to a tree—since you are installing paths individually, loops should not be a problem. In fact, you must test that your solution works on topologies with loops.

3.1 Approach

We can leverage the centralized SDN architecture to perform shortest path switching without broadcasts, as follows:

a. As usual, when a host wants to send a packet, it consults its routing table to determine if the destination is in the same subnet (will always be true in this assignment). This means the host will send the packet to the IP destination as an Ethernet frame destined to the MAC address of the destination (as opposed to the MAC address of a gateway or router). If the host does not know the MAC address for the destination, it issues an ARP request

[1]http://mininet.org

[2]https://openvswitch.org

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

b. When a switch receives the ARP request, it will send the request to the controller as a PacketIn message, rather than broadcasting it

c. The controller will receive the PacketIn message and look up the MAC address of the destination host, then generate a response (inside a PacketOut message) for the switch to send back to the sender host

d. Upon receiving the response, the host will send the IP packet to the destination's MAC address

e. At each switch along the path to the destination (as determined previously by your code), the packet will match on the destination MAC address and be forwarded on the correct port.

In order for the controller to know the MAC address of each host, we must establish a protocol for hosts to inform the controller of its address. For this assignment, we require that hosts send an unsolicited ARP reply (also called a "gratuitous ARP", or an arping) when connecting to tell the network its MAC and IP address—we have configured Mininet to do this automatically when starting the emulated network.

Finally, since we are not broadcasting ARP messages, all ARP requests will be sent to the controller instead. When you receive an ARP request, you should generate an appropriate response so a host can populate its ARP table.

3.2 Your Task

We have provided a starter Ryu application that receives events from switches when the network topology changes or when hosts are added or removed from the network. Your task is to extend this application to build a map of the network's topology, compute the shortest path to forward packets between each host, and install flow rules accordingly.

Ryu applications are commonly built using event handlers. The starter application has stub event handlers that print information when a host or switch joins the network, a link is added or removed between two switches, and when a host or switch leaves the network. You will need to use this event information to build an accurate map of the network's topology. You can do this however you like—feel free to create additional data structures or files as necessary.

To compute shortest paths, you should use either the Bellman-Ford or Djikstra's algorithm to compute the shortest paths to reach host *h* from every other host $h' \in H$, $h = h'$. *H* is the set of all hosts, which will be provided by your topology graph.

Once you have determined the shortest path to reach host *h* from *h'*, you must install a rule in the flow table in every switch in the path. Thus, you should install switch rules *proactively*: after startup, each switch should have a flow table containing rules for each known host on the network. Each rule should match IP packets (i.e., Ethernet type is IPv4) whose destination MAC is the MAC address assigned to host *h*. In your controller, you can create rules with a single action to output packets on the appropriate port to reach the next switch in the path.

When the topology changes, you should update the rules for the affected paths. For this assignment, you may choose to recompute all of the topology or be more efficient and only remove and install the rules that need to change.

3.3 Testing and Debugging

You should test your code by sending traffic between various hosts in the network topology—mininet's built-in pingall command is very useful for this. While you MUST handle loops in the topology correctly, you can assume that the topology is connected, i.e., we will not test your code with topologies where a host is unreachable from other hosts.

Overall, this assignment is designed to get you to experiment with the SDN paradigm. We do not intend for you to spend a lot of time handling edge cases associated with non-standard host behavior or route consistency. In particular, you will find that there are many edge cases associated with updating flow tables across multiple switches consistently—you are not required to handle these, but you may do so if you wish.

4 Getting Started

We have prepared a starter repository with a template Ryu application from which you can begin building your implementation.

Like IP and TCP, this assignment should be completed in teams.

The starter repository contains the following files:

- run_mininet.py: Runs mininet with different topologies. You should not need to modify this.
- shortest_paths.py: Starter Ryu application with handlers for topology events. See the comments for more information.
- topo_manager_example.py: An example of a framework for keeping track of the network topology. You may extend this, or ignore it and create your own instead.
- ofctl_utils.py: Contains utility functions for building and sending packets and OpenFlow messages.

4.1 Development Environment

You should implement this project on the provided Vagrant VM or a similar system. Since mininet requires superuser privileges to run, running a VM on your own system is suggested.

As before, you can find details on how to use our Vagrant VM in vagrant.pdf.

Note: **If you have been developing on your own system instead of the VM**, we *highly* recommend you use a VM for this project. Mininet modifies a lot of kernel state to create its required network namespaces—it is a good idea to use it only in a VM so that your own network is not affected if it crashes. Mininet also runs a lot of Python code as root, which may not be desirable.

To prepare your VM for this project, you will first need to install the required components. You can do this by running:

```
sudo apt-get install mininet python3-ryu iputils-arping
```

For grading, your implementation must be able to run using the system versions of mininet, Ryu, and arping provided in our Vagrant VM as installed by apt. Please **do not** install other versions of these packages (such as from pip, or by building from source), which may cause compatibility issues.

We will be using Python 3 for this assignment. Further, you should not require any Python libraries other than those already installed. If you believe any libraries are missing or would like to install additional dependencies, please contact the course staff first.

4.2 Starting the network

a. First, start the controller with your application:

```
$ ryu-manager --observe-links shortest_paths.py
```

(For additional debugging information, you can also add --verbose, or change the log level–run with --help for details.) With the default settings, you should see output like the following:

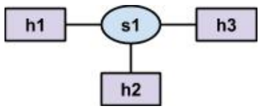
```
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app sp_switch.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app sp_switch.py of ShortestPathSwitching
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
```

Keep the terminal with Ryu open, as you will need to see the output for debugging.

b. In another terminal, start mininet using our provided start script, as follows:

```
$ sudo python run_mininet.py single 3
```

The above command will create a topology with a single SDN switch (s1) and three hosts (h1 - h3) directly connected to the switch:



You can change the number of hosts by changing the numeric value included in the argument to the run_mininet.py script. You can also start Mininet with six other topologies, which are shown in Appendix A.

Once mininet has started, you should see Ryu produce output like the following:

```
      Added Switch switch1 with ports:
                1:    22:52:03:45:39:f4
                2:    be:8f:22:b6:b7:d7
                3:    6e:86:9d:0b:bf:15
Host Added:                                          00:00:00:00:00:01 (IPs:
                                                     ['10.0.0.1']) on switch1/1
                                                        (22:52:03:45:39:f4)
Host Added:                                          00:00:00:00:00:02 (IPs:
                                                     ['10.0.0.2']) on switch1/2
                                                        (be:8f:22:b6:b7:d7)
Host Added:                                          00:00:00:00:00:03 (IPs:
                                                     ['10.0.0.3']) on switch1/3
                                                        (6e:86:9d:0b:bf:15)
```

These messages are printed by the event handlers, demonstrating how your controller can get information about the network.

c. You can now run commands in mininet. For example, to send a ping packet from h1 to h2, run h1 ping h2. **Note that, initially, ping will not work**, since your switches do not know how to do anything! After your controller installs the correct rules, you can start sending packets.

4.3 Working with Mininet and Ryu

Once mininet starts, it creates its own shell that you can use to run commands on each host or change the network. The following sections list some useful commands for working with your emulated network.

In the lists below, commands prefixed with MN> should be run in the mininet CLI; commands prefixed with shell> should be run in a standard shell on your VM.

4.3.1 Triggering Event Handlers

You can trigger the various event handlers to using mininet commands to change the state of different links. You can run commands like the following to manipulate links (substituting the switch and host names as necessary):

MN> link s1 s2 down Takes down the link between s1 and s2; you can assume the network is a connected graph, so you should never take down a link that would result in a disconnected graph.

MN> link s1 s2 up Brings up the link between s1 and s2.

MN> link s1 h1 down Takes down the link between s1 and h1.

MN> link s1 h1 up Brings up the link between s1 and h1.

shell> sudo ovs-vsctl del-bar s1 Removes a switch from the network. Note that once a switch is removed, you cannot easily add it back without restarting mininet.

MN> arping h1 Send an arping from h1, which generates an ARP request identifying h1's MAC and IP address. Triggers a EventHostAdd event.

MN> arping_all Send an arping from all hosts. This command is run automatically when mininet starts. You can also re-run it yourself—this is very useful if you want to restart the controller without restarting mininet.

Try out these commands and observe the messages they generate. How can you use these to build a map of the network's topology?

4.3.2 Other useful commands

Here are some other commands that may be useful:

MN> h1 ping h2 -c 1 Send a *single* ping packet from h1 to h2

MN> pingall Ping all hosts from all other hosts

MN> net View the current network topology

MN> dpctl dump-flows Dump the flow tables for all switches

shell> sudo ovs-ofctl dump-flows s1 Dump the flows table on a single switch

shell> sudo mn -c Forces an existing mininet instance to clean up all its state and close. Run this if mininet ever crashes and fails to restart properly.

5 Implementation

A few more notes:

- You can install flow rules using OfCtl.set_flow, providing the values of the fields you want to match as keyword arguments. The actions should be a list containing a single action of type OFPActionOutput to output packets on the appropriate port. **See Appendix B for examples.**
- In general, all rules you install should have the default
- You can generate and send ARP packets using the function OfCtl.send_arp. See the comments in the utils file for information on the appropriate fields.
- All testing will be conducted using the provided Mininet environment using the commands entered by a human operator. You are not required to devote time to handling edge cases involving forwarding table consistency over multiple switches, or topologies that are not fully-connected. However, you must be able to handle networks with loops.
- You may assume that each host will only be connected to one switch and only have one IP address. In addition, the hosts will never move to a different port during a single test.

Overall, this project is not meant to be as complex as other assignments. If you find yourself stuck, or encounter any issues with the tools, please do not hesitate to ask us!

Known Issues

- When issuing a link ... down, it seems that the switch will send a PortStatus message with the UP state before immediately sending one with the DOWN state. This appears to be normal behavior. In this case, we recommending ignoring the UP message, since it reflect the port's current state before the message was received.
- When you issue a link ... down command, sometimes we have seen mininet resurrect the link. This seems to be a problem with mininet. In case this happens, bringing the link down a second time seems to kill it for good.

6 Capstone Credit

If you are taking this course for capstone, one of the options for the capstone project is to implement an extra feature in this assignment. The following are some options, though you may suggest your own as well:

a. Implement flooding without loops (essentially calculate and install a spanning tree for broad-casts).
b. Implement ECMP on networks with multiple paths (determine the number of paths between two nodes) and install rules that match on, say, even and odd TCP ports!

Please consult with us before starting your capstone work for this project.

7 Helpful Resources

Here are some links to resources about the various software components we will be using:

- Ryu's API documentation[4]
- When the Ryu documentation lacking, the best place to look is in the Ryu source code itself. On the VM, the source is located in /usr/lib/python3/dist-packages/ryu. In particular, the following locations may prove useful:
  - lib/topology/switches.py: This file contains most of the data structures that are returned from the event handlers
  - lib/packet: Utility classes for building and parsing packet types
  - app/: Library of example Ryu applications. Ignore any examples ending in numbers (eg. switch_12, switch_13, etc.), which pertain to different OpenFlow versions.
- Mininet tutorials and documentation[5]

4 https://ryu.readthedocs.io/en/latest/index.html
5 https://github.com/mininet/mininet/wiki/Documentation

- The OpenFlow v1.0 specification[6]: While it is not necessary for you to be familiar with the wire format of OpenFlow messages, this very readable document can give you a good idea on the types of interactions that are possible between controllers and switches in OpenFlow

8 Handing In and Interactive Grading

In your final submission, you should include a README documenting any design decisions you made, including how you build the graph of the network topology, and your strategy for installing switch rules.

9 Acknowledgments

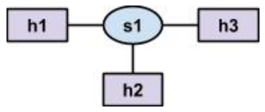
The code for this project is based on an assignment from Prof. Aditya Akella for CS640, Computer Networks, from the University of Wisconsin, Madison, and Prof. Rodrigo Fonseca for CS168, Computer Networks, from Brown university, with some changes.

6 https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf

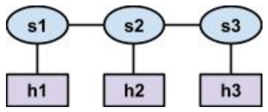
Appendix A Topologies

Our provided mininet start script can create the following topologies:

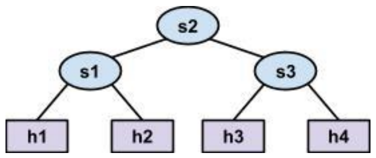
- single n: A single switch with n directly-connected hosts. single 3 produces the following topology:



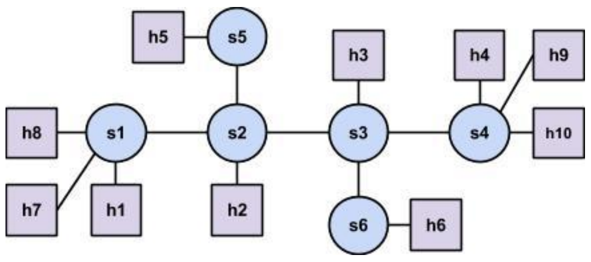
- linear n: a chain of n switches with one host connected to each switch; for example, linear 3 produces the following topology:



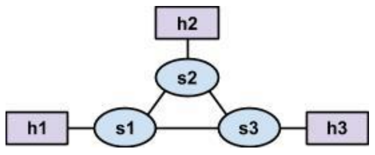
- tree n: a tree of depth n with a single root switch (s1) and two hosts connected to each leaf switch; for example, tree 2 produces the following topology:



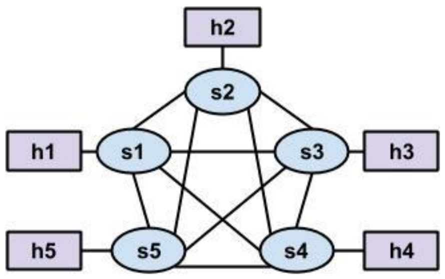
- assign1 creates the following topology (the name for historical reasons):



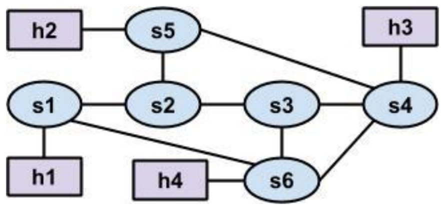
- triangle creates the following topology:



- mesh n: a complete graph with n switches and one host attached to each switch; for example, mesh 5 produces the following topology:



- someloops: creates the following topology:



Appendix B Updating Flow Rules

B.1 Adding flow rules

In general, switch rules have two main components:

- A set of *match criteria* to define the types of packets for which the flow applies. These criteria are based on various packet fields, including source and destination addresses, protocol numbers, etc.
- A set of *actions* for the switch to to take if the packet matches the flow. Typical actions include sending the sending the packet out on a specific port, modifying it in some way, or dropping it.

In addition, OpenFlow adds two more parameters for flow creation: a *cookie* value, which is used to uniquely identify rules, and a *priority* value. In this assignment, you can use a cookie value of 0 and use the default priority, which is also zero.

You can create and install flow rules on a switch using the set_flow method provided in the OfCtl utility class, defined in ofctl_utils.py. Take a moment to look at the definition and documentation for this method.

It is typical to construct rules that match on a subset of the available match fields. To do this, you can simply call the function with only the arguments you need—the rest will be replaced with default values. Other arguments are required: all rules should have a cookie value, a priority, and a VLAN ID; you can start by setting all of these zero. The actions field must contain a list of Ryu OFPAction objects.

To summarize, here is an example for installing a rule that matches IP packets with a particular destination MAC address (dl_dst) and outputs packets on a certain port (port):

```
def add_forwarding_rule(self, datapath, dl_dst, port):
        ofctl = OfCtl.factory(datapath, self.logger)

        actions = [datapath.ofproto_parser.OFPActionOutput(port)]
        ofctl.set_flow(cookie=0, priority=0,
                                dl_type=ether_types.ETH_TYPE_IP,
                                dl_vlan=VLANID_NONE,
                                dl_dst=dl_dst,
                                actions=actions)
```

Note that this function takes one additional argument called datapath, which is the Datapath object corresponding to a particular switch. This argument provides the necessary metadata to generate OpenFlow packets destined for a particular switch.

B.2 Deleting flow rules

When a new flow is added with the same match criteria and priority as an existing flow, it replaces the existing flow in the switch.

It is also possible to delete flows in a similar way to adding flows using OfCtl.delete_flow. To do this, you can construct an OFPMatch object that specifies a selection of flows you want to delete. By default *any* flows that contain *any* matching fields will be deleted. Here is an example for deleting flows that match a certain destination MAC address:

```
def delete_forwarding_rule(self, datapath, dl_dst):
    ofctl = OfCtl.factory(datapath, self.logger)

    match = datapath.ofproto_parser.OFPMatch(dl_dst=dl_dst)
    ofctl.delete_flow(cookie=0, priority=0, match=match)
```