

CS557 SDN Project

Table of Contents

1 Introduction	2
2 Overview.....	3
2.1 Components.....	3
3 Setup	4
4 Shortest-path Switching.....	8
4.1 Code overview	9
4.2 To-Do's	9
4.3 Testing and Debugging.....	10
5 What to Submit.....	11
6 Acknowledgments.....	11

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.

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. SDN as described 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.

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.

When a host wants to send a packet to the IP address of another host in the network, it will first, as it usually does, issue an ARP request. When this reaches the first switch, though, the switch will send the ARP request to the controller instead of flooding the request. The controller, who knows the topology, will respond to the ARP request, through the same switch, to the original sender, with the MAC address of the destination. The controller will also have installed rules on the switches for forwarding to each destination MAC.

2.1 Components

You will run the code for this project in an emulated network inside of a single Linux VM. You will use the Mininet¹ network emulator, which is designed to emulate arbitrary topologies of emulated OpenFlow² switches and Linux hosts. It uses container-based virtualization for very lightweight emulated nodes. The switches in your network run the open source Open vSwitch switch³ software that implements the OpenFlow protocol. The switches connect to an OpenFlow network controller, and you will use Floodlight⁴, a relatively mature Java-based controller. We will use OpenFlow version 1.0 for this project.

Code you run on Mininet is ready to run with no changes in real networks.

¹ <http://mininet.org>

² <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>

³ <http://openvswitch.org>

⁴ <http://projectfloodlight.org>

3 Setup

There is a prepared VM image for you to do this project. In order to run the Floodlight/Mininet VM the first thing you will need to do is to install the VirtualBox virtualization software (free, and available for all mainstream operating systems). Please refer to the instructions at <https://www.virtualbox.org/> to do so. *While the VM may run on other virtualization technologies, for grading purposes we will run the VM using VirtualBox. Therefore, assignments that do not work with VirtualBox will be rejected.*

Download the VM image by pointing your web browser to www.cs.colostate.edu/~cs557/Mininet_VM.ova. Your browser will ask whether you want to save this file. Save it, but be aware that the VM is slightly larger than 2 gigabytes, so it may take 45 minutes or more to complete the download.

Start VirtualBox and import the ova file as a new appliance (FILE / IMPORT APPLIANCE). When the import has finished, start the new VM. You may have to correct a few configuration items; for example, I had to disable USB devices before the VM would run, or perhaps change USB from 2.0 to 1.0... whatever works for you. You may also have to adjust networking configurations.

Both username and password for the VM are “mininet”.

This particular VM did not allow me to re-size the screen because the screen driver was not present. The easiest way to deal with this screen size issue is to avoid it. Logon to the VM GUI, start a terminal session, and type “ifconfig”. Determine the IP address of your VM, and from then on use one or more terminal windows in your laptop to SSH into the VM. (ssh mininet@x.y.z.w) My ifconfig gave a 10.0.2.15 address and a 192.168.56.101. I could only SSH into the second one.

We will NOT be using ECLIPSE to build our floodlight instances. You will simply edit the files mentioned later and use “ant” to recompile the floodlight code. On the other hand, Eclipse is present in the system if you want to figure out how to use it...

Once you’ve ssh’d into the VM, you can follow these steps to run your control application:

- a. Compile Floodlight and your applications

```
$ cd ~/openflow
$ ant
```

This will produce a jar file `FloodlightWithApps.jar` that includes the compiled code for Floodlight and your SDN applications.

- b. Start Floodlight and your SDN applications:

```
$ java -jar FloodlightWithApps.jar -cf shortestPathSwitching.prop
```

where the prop file configures your application.

When Floodlight starts, you should see output like the following:

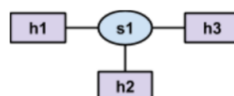
```
23:18:45.874 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules
from file shortestPathSwitching.prop
23:18:46.277 INFO [n.f.c.i.Controller:main] Controller role set to MASTER
23:18:46.285 INFO [n.f.c.i.Controller:main] Flush switches on reconnect -
- Disabled
23:18:46.302 INFO [ArpServer:main] Initializing ArpServer...
23:18:46.302 INFO [ShortestPathSwitching:main] Initializing
ShortestPathSwitching...
23:18:48.533 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting
autoportfast feature to OFF
23:18:48.579 INFO [ArpServer:main] Starting ArpServer...
23:18:48.580 INFO [ShortestPathSwitching:main] Starting
ShortestPathSwitching...
23:18:48.700 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet
configured; using fallback local configuration
23:18:48.701 INFO [o.s.s.i.SyncManager:main] [32767] Updating sync
configuration ClusterConfig
[allNodes={32767=Node [hostname=localhost, port=6642, nodeId=32767,
domainId=32767]},
authScheme=NO_AUTH, keyStorePath=null, keyStorePassword is unset]
23:18:48.790 INFO [o.s.s.i.r.RPCService:main] Listening for internal
floodlight RPC on localhost/127.0.0.1:6642
23:18:48.978 INFO [n.f.c.i.Controller:main] Listening for switch
connections on 0.0.0.0/0.0.0.0:6633
```

Keep the terminal with Floodlight open, as you will need to see the output for debugging. Use another terminal for the next step.

c. Start Mininet:

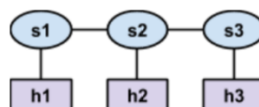
```
$ sudo ./run_mininet.py single,3
```

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

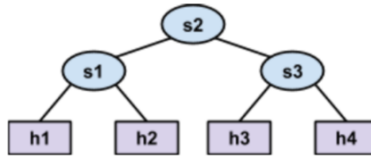


You can change the number of hosts by changing the numeric value included in the argument to the `run_mininet.py` script. You can also start Mininet with six other topologies:

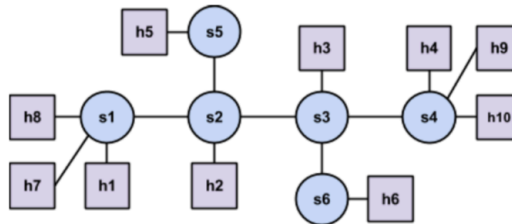
- `linear, n`: a chain of `n` switches with one host connected to each switch; for example, `linear, 3` produces the following topology:



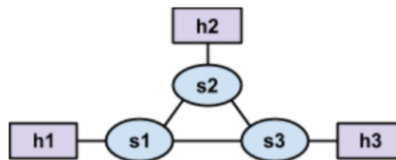
- `tree, n`: a tree of depth n with a single root switch ($s1$) and two hosts connected to each leaf switch; for example, `tree, 2` produces the following topology:



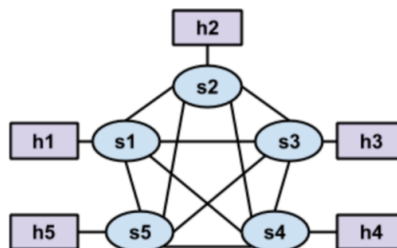
- `assign1` creates the following topology (the name is this way for historical reasons):



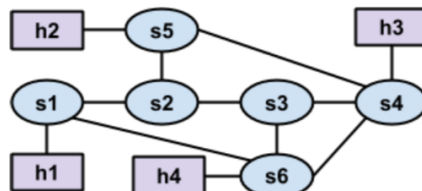
- `triangle` creates the following topology:



- `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:



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

```
23:24:10.304 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] New switch
connection from /127.0.0.1:58911
23:24:10.329 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1]
Disconnected switch [/127.0.0.1:58911 DPID[?]]
23:24:11.016 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] New switch
connection from /127.0.0.1:58912
23:24:11.101 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Switch
OFSwitchBase
[/127.0.0.1:58912 DPID[00:00:00:00:00:00:00:01]] bound to class class
net.floodlightcontroller.core.internal.OFSwitchImpl, writeThrottle=false,
description OFDescriptionStatistics [Vendor: Nicira, Inc., Model: Open vSwitch,
Make: None, Version: 2.0.2, S/N: None]
23:24:11.104 INFO [n.f.c.OFSwitchBase:New I/O server worker #2-2] Clearing all
flows on switch
OFSwitchBase [/127.0.0.1:58912 DPID[00:00:00:00:00:00:00:01]]
23:24:11.107 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01
connected.
23:24:11.108 INFO [ShortestPathSwitching:main] Switch s1 added
23:24:11.138 INFO [ShortestPathSwitching:Topology Updates] Link s1:0 -> host
updated
23:24:11.211 INFO [ShortestPathSwitching:Topology Updates] Link s1:1 -> host
updated
23:24:11.212 INFO [ShortestPathSwitching:Topology Updates] Link s1:3 -> host
updated
```

- d. You can now run commands (e.g., ping) and the like. Notice that initially ping will not work, as your switches do not know how to do anything. After your controller installs the correct rules, things should work.

You should always start Floodlight and your SDN applications before starting mininet. Also, we recommend that you restart Floodlight and your SDN applications whenever you restart mininet.

4 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.

Differently from regular L2 switches or L3 routers, SDN switches don't hold MAC learning tables or routing tables. Rather, they use a more general flow table structure that can replace these and other constructs. Each entry, or rule, in a flow table has match criteria that define (on the basis of fields in Ethernet, IP, TCP, UDP, and other headers) which packets the rule applies to. Each entry also has one or more instructions/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 a specific port on the SDN switch.

The match criteria serve 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, as you are installing paths individually and loops should not be a problem. In fact, you must test that your solution works on topologies with loops.

After the rules are all installed, the process a host will go through to send an IP packet to a destination IP address is as follows:

- a. Host OS determines that the node is in the same subnet (will always be true in this assignment). This means the node 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).
- b. Host OS issues an ARP request to determine the destination MAC address (if not already cached at the OS).
- c. The first switch to see the ARP request, rather than broadcasting it, sends it to the controller as a `PacketIn` message.
- d. The Floodlight module `ArpServer` (which we provide, see the `util` package) will respond with the (? Missing word in original document) if this host has sent any Ethernet frames before.
- e. The host OS will send the IP packet to the destination's MAC address.
- f. 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.

4.1 Code overview

You will complete the implementation of a Floodlight module in the file `ShortestPathSwitching.java` in the directory `edu.brown.cs.sdn.apps.sps`.

The file we provided already contains code to:

- Access host and topology information from other modules (or applications) included with Floodlight – see the `getHosts()`, `getSwitches()`, and `getLinks()` methods.
- Receive notifications about changes in the network: see the `deviceAdded()`, `deviceRemoved()`, `deviceMoved()`, `switchAdded()`, `switchRemoved()`, and `linkDiscoveryUpdate()` methods.

We have also provided code in the `edu.brown.cs.sdn.apps.utils` package for:

- A Floodlight module that responds to ARP requests from hosts – see `ArpServer.java`
- Telling a switch to install a rule in the flow table, remove rules from the flow table, and send a packet – see `SwitchCommands.java`.

In this project we will install rules to reach all hosts we know about. In the launch script for mininet we have added instructions for all hosts to issue an `arping`, which allows the controller to learn about the hosts' presence and populate its ARP cache.

4.2 To-Do's

You need to complete the To-Do's in `ShortestPathSwitching.java` to install and remove flow table entries from SDN switches such that traffic is forwarded to a host using the shortest path.

You should use either the Bellman-Ford or Dijkstra algorithms to compute the shortest paths to reach a host h from every other host $h' \in H$, $h \neq h'$ (H is the set of all hosts). You can use the `getHosts()`, `getSwitches()`, and `getLinks()` methods to get the topology information that you need to provide as input to the Bellman-Ford algorithm.

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. The rule should match IP packets (i.e., Ethernet type is IPv4) whose destination MAC is the MAC address assigned to host h . You can specify this in Floodlight by creating a new `OFMatch` object and calling the set methods for the appropriate fields. The rule's action should be to output packets on the appropriate port in order to reach the next switch in the path. You can specify this in Floodlight by creating an `OFInstructionApplyActions` object whose set of actions consists of a single `OFActionOutput` object with the appropriate port number.

SDN switches have multiple flow tables. You should install rules in the table specified in the table class variable in the `ShortestPathSwitching` class. Also, your rules should never timeout and have a default priority (both defined as constants in the `SwitchCommands` class).

When the topology changes you will have to recompute a subset of the 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.

4.3 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.

To help you debug, you can view the contents of an SDN switch’s flow tables by running the following command in your mininet VM (not in mininet itself):

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

This will output the contents of `s1`’s flow tables. Change the last argument to output the flow tables from a different switch.

Triggering Event Handlers

- You can trigger the `linkDiscoveryUpdate()` event handler by running any of the following commands in mininet (substituting switch and host names as desired):
 - `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.
 - `link s1 s2 up` – brings up the link between `s1` and `s2`.
 - `link s1 h1 down` – takes down the link between `s1` and `h1`; this will also result in a `deviceRemoved(...)` event and the `isAttachedToSwitch()` method for the Host object for `h1` will now return false.
 - `link s1 h1 up` – brings up the link between `S1` AND `H1`; this will also result in a `deviceMoved(...)` event and the `isAttachedToSwitch()` method for the Host object for `h1` will now return true.
- You can trigger the `deviceRemoved(...)` event handler by taking down a link between a switch and a host, as described above.

- You can trigger the `deviceMoved(...)` event handler by bringing up a link between a switch and a host, as described above.
- You can trigger the `switchRemoved(...)` event handler by running the following command in a regular terminal window (not in mininet):

```
sudo ovs-vsctl del-br s1
```

Note that once a switch is removed, you cannot easily add it back without restarting mininet. You can assume the network is a connected graph, so you should never remove a switch that would result in a disconnected graph.

Known Issues

- 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.
- In the VirtualBox image, it is possible that the system will start an `openvswitch-controller` process by default, which means your Floodlight controller will not be able to bind to port 6633. To prevent it from starting the next time you boot up, do

```
sudo update-rc.d -f openvswitch-controller remove
```

5 What to Submit

You will only hand in the contents of the `edu/brown/cs/sdn/apps/sps` directory. Thus, when you are ready to submit, create a tarball of the contents of that directory and copy (scp) to a department machine or your laptop. Submit the tarball to CANVAS.

6 Acknowledgments

This document, the virtual machine and the code for this project is based on an assignment from Professor Rodrigo Fonseca's course at Brown University. I have written to Dr. Fonseca to ask permission, but have not received a reply. Since it is available on the web I have taken the liberty of using it, but I still hope to receive formal permission at some point.

The Brown University project, in turn is based on Prof. Aditya Akella for CS640, Computer Networks, from the University of Wisconsin, Madison, with some changes.

I have modified the document with changes to adapt to the CSU environment, such as CANVAS, the downloading of the VM, etc.