New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Course Title: Data Communications & Networks **Course Number:** g22.2662-001
Instructor: Jean-Claude Franchitti **Session:** 10

Assignment 9: Final Project

I. Due

Monday, December 21st 2020 by 11:59 pm.

II. Objectives

Software-defined networking (SDN) is a new paradigm for running networks. As per the Networking Layer topics covered in the course, the network is divided into the control and data planes. The control plane provides a 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 project, you will implement the logic in such a controller to manage the following:

1. A layer-3 routing application that installs rules in SDN switches to forward traffic to hosts using the shortest, valid path through the network. Your application logic will 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 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. A distributed load balancer application that redirect new TCP connections to hosts in a round-robin order.

As always, the NYU and class policy about plagiarism must be followed in this project. If you use ANY code in your project that is not of your own creation, then you MUST attribute that code to the author, even if you modify it (ANY modification).

III. References

1. Slides and handouts posted on the course Web site
2. Textbook chapters as applicable
3. Mininet network emulator documentation (http://mininet.org)
4. Openflow documentation (https://www.opennetworking.org/sdn-resources/onf-specifications/openflow)
5. Open vSwitch switch software documentation (http://openvswitch.org)
6. Floodlight Java-based SDN controller documentation (https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview)
7. If you have additional questions about SDN, OpenFlow, or Floodlight you may want to consult: openflow-switch-v1.5.1.pdf (opennetworking.org) (sections 2, 3, and 5.1 - 5.4 are likely to be the most useful), and Floodlight-plus Javadoc
8. Additional readings:
   - Software Defined Networking Concepts
   - The Road to SDN: An Intellectual History of Programmable Networks
   - SDN Reading List

IV. Software Required

1. Microsoft Word
2. Win Zip as necessary
3. Oracle VirtualBox
4. Virtual Box Image with all necessary software provided
5. Java Programming language, Eclipse, and other development tools installed in Virtual Box Image provided
6. Additional code for Part 4

V. Assignment

1. Background:

You will run the code for this project in an emulated network inside of a single Linux VM. You will use the Mininet network emulator, which is designed to emulate arbitrary topologies of emulated OpenFlow switches and Linux hosts. It uses container-based virtualization for very light-weight emulated nodes. The switches in your network run the open source Open vSwitch switch software, which implements the Openflow protocol. The switches connect to an Openflow network controller, and you will use Floodlight, a relatively mature Java-based controller. We will use OpenFlow version 1.0 for this project. Your SDN applications will be written in Java and run atop the Floodlight OpenFlow controller. You will use Mininet to emulate a variety of network topologies consisting of OpenFlow switches and hosts.

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

2. Environment Setup:

a. Install Oracle VirtualBox as necessary.

b. Download the Virtual Box Image with all necessary software provided. It is a .ova image that will enable you to run the necessary software on your computer using the latest version of Oracle VirtualBox. To install the .ova file go to File and Import Appliance on VirtualBox. This VM uses "mininet" as username and password.

c. To ssh into the VM from your host computer, log in first using the GUI, open a terminal, and type ifconfig. This will show you the IP addresses of the VM. You will be able to connect to one of them from your host computer via ssh. The VM also has Eclipse installed, which you can use inside the VirtualBox graphical console or remotely via X. Once you've ssh'd into the VM, you can go through the following steps to run your control applications.

d. Optional (see acknowledgement in item 8 below):
Refactor `edu.brown.cs.sdn.apps.sps` to `edu.nyu.cs.sdn.apps.sps`

e. Compile Floodlight and your applications:

```
$ cd ~/project3/
$ ant
```

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

f. Start Floodlight and your SDN applications:

```
$ java -jar FloodlightWithApps.jar -cf l3routing.prop
```

The above command will start Floodlight and only your layer-3 routing application. The `.prop` file configures your application.

Note: For future reference when working on part 4, you can start both your layer-3 routing and load balancer applications by using `loadbalancer.prop` for the `-cf` (configuration file) argument. The `loadbalancer` application code is provided separately.

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.

Note: 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
```

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

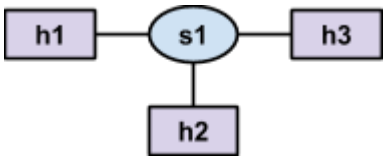
```
23:18:45.874 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from file shortestPathSwitching.prop
23:18:46.277 INFO [n.f.c.i.Controller:main] Controller role set to MASTER
23:18:46.285 INFO [n.f.c.i.Controller:main] Flush switches on reconnect -- Disabled
23:18:46.302 INFO [ArpServer:main] 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.

g. Start Mininet:

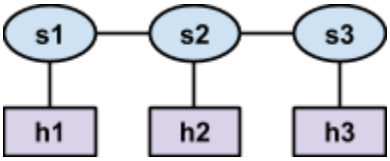
```
$ sudo ./run_mininet.py single,3
```

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

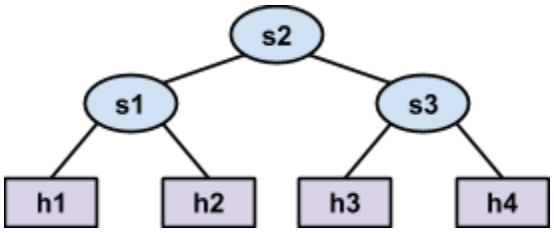


You can change the number of hosts by changing the numeric value included in the arguments to the run_mininet.sh script. You can also start Mininet with four other topologies:

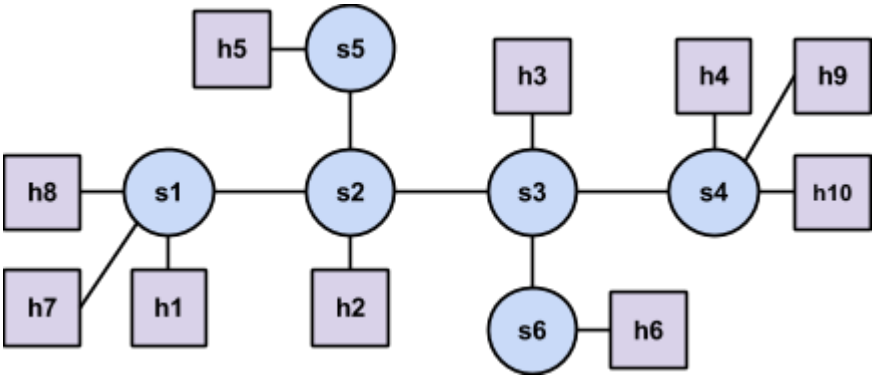
- `linear,n`: a chain of n switches with one host connected to each switch; for example, `linear,3` produces the following topology:



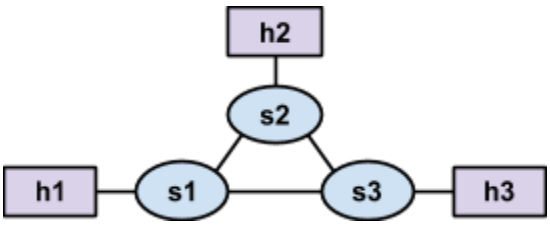
- `tree,n`: a tree of depth n with a single root switch (`s1`) and two hosts connected to each leaf switch; for example `tree,2` produces the following topology:



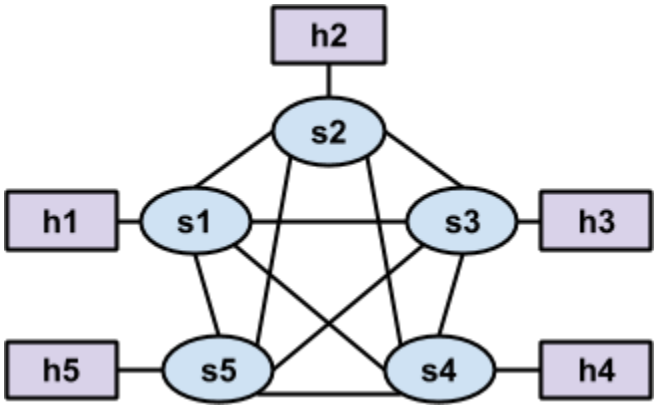
- `assign1`: creates the following topology (the name is this way for historical reasons):



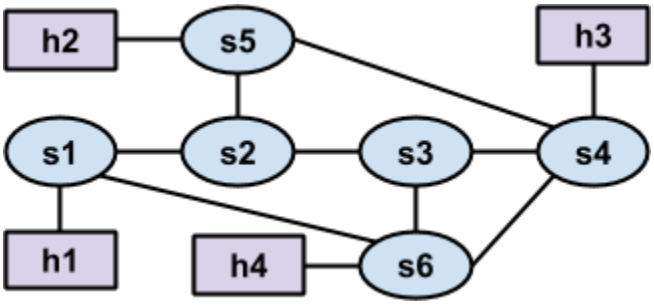
- `triangle`: creates the following topology:



- `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
```

for debugging. Use another terminal for the next step.

h. You can now run commands (e.g., ping) and the like in Mininet. Note 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.

3. Layer-3 "Shortest-Path Switching" Routing Application Implementation:

Your first SDN controller application consists of 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.

Your task is therefore 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. Your application will construct route tables based on a global view of the network topology. The appropriate route table will then be installed in each SDN switch, and each SDN switch will forward packets according to the route table installed by your application.

Differently from regular L2 switches or L3 routers, SDN switches don't hold MAC learning tables or routing tables (used in traditional layer-3 routers). Rather, they use a more general flow table structure, which can replace these, as well as MAC learning tables (used in traditional layer-2 switches), and many other constructs. Each entry, or rule, in a flow table has match criteria that defines (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 which should be taken for each packet that matches the rule. There is no concept of a gateway in SDN flow tables, but that's okay—your router only uses the gateway to determine how to rewrite a packet's destination MAC address to ensure correct layer-2 forwarding, and we are not using traditional layer-2 forwarding in SDN.

Your layer-3 "Shortest-Path Switching" routing application will install entries that match packets based on their destination IP address (and Ethernet type), and execute an output action to send the packet out a specific port on the SDN

switch. (You'll use other match criteria and additional instructions/actions for the other SDN application you will write, which is described in Part 4 of this project.) The match criteria serve the same purpose as the destination and mask fields in a traditional route table, and the output action servers 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 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.

3.1 Code Overview

You will complete the implementation of a Floodlight module in the file `ShortestPathSwitching.java` in `edu.brown.cs.sdn.apps.sps` (or `edu.nyu.cs.sdn.apps.sps` if you refactored that package earlier).

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.util` package ((or `edu.nyu.cs.sdn.apps.util`) if you refactored that package earlier) 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.

3.2 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 Djikstra 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 (we discuss this more in Part 4 below). For now, 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.

Part 3 Extra Credit:

- Implement flooding without loops (essentially calculate and install a spanning tree for broadcasts).
- 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!

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.)

To help you debug, you can view the contents of an SDN switchs 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 Issue: when you issue a `link ... down` command, sometimes we have seen mininet ressurect 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.

4. Distributed Load Balance Routing Application Implementation:

Networks employ load balancing to distribute client requests among a collection of hosts running a specific service (e.g., a web server). In class, we briefly discussed how DNS could be used to implement load balancing. Load balancing is also commonly implemented using a special piece of hardware.

A hardware load balancer is placed in the network and configured with an IP address (e.g., `10.0.100.1`) and a set of hosts among which it should distribute requests (e.g., `10.0.0.2` and `10.0.0.3`). Clients wanting to communicate with a service (e.g., a web server) running on those hosts are provided with the IP address of the load balancer, not the IP address of a specific host. Clients initiate a TCP connection to the IP address of the load balancer (`10.0.100.1`) and the TCP port associated with the service (e.g., port `80`).

For each new TCP connection, the load balancer selects one of the specified hosts (usually in round robin order). The load balancer maintains a mapping of active connections—identified by the client's IP and TCP port—to the assigned hosts.

For all packets sent from clients to the load balancer, the load balancer rewrites the destination IP and MAC addresses to the IP and MAC addresses of the selected host. The mapping information stored by the load balancer is used to determine the appropriate host IP and MAC addresses that should be written into a packet arriving from a client. For all packets sent from servers to clients, the load balancer rewrites the source IP and MAC addresses to the IP and MAC addresses of the load balancer.

Your second SDN application will implement the same functionality as a set of hardware load balancers. Your application will be provided with a list of `virtual IP`s and a set of hosts among which connections to the `virtual IP`s should be load balanced. (We use the term `virtual IP` because the IP address is not actually assigned to any node in the network.) When clients initiate TCP connections with a specific `virtual IP`, SDN switches will send the TCP `SYN` packet to the SDN controller. Your SDN application will select a host from a pre-defined set, and install rules in an SDN switch to rewrite the IP and MAC addresses of packets associated with the connection. You will also instruct the SDN switch to match the modified packets against the flow rules installed by your layer-3 routing application and apply the appropriate actions (i.e., send the packets out the appropriate ports).

4.1 Code Overview

The code for your load balancer application will reside in the `LoadBalancer.java` source file provided in `FinalProject-Part4-Code.tgz` (see `edu.wisc.cs.sdn.apps.loadbalancer` package). The file provided already contains code to:

- Receive a notification when a switch joins the network—`switchAdded(...)`
- Receive a packet from a switch when the packet did not match any entries in the switch's flow table—`receive(...)`

The `LoadBalancerInstance` class represents a single distributed load balancer. (We use the term distributed because the load balancing is performed at many switches, rather than at a single hardware load balancer.) Each load balancer instance has a virtual IP address, virtual MAC address, and set of hosts among which TCP connections should be distributed. The instances class variable in the LoadBalancer class maps a virtual IP address to a specific load balancer instance.

4.2 To-Do's

It is recommended to refactor the package provided in FinalProject-Part4-Code.tgz to `edu.nyu.cs.sdn.apps.loadbalancer` (see acknowledgment in item 8). Also note that `edu.wisc.cs.sdn.apps.l3routing` is not used in this project.

You need to complete the To-Do's in `LoadBalancer.java` to:

- Install rules in every switch to:
  - Notify the controller when a client initiates a TCP connection with a virtual IP—we cannot specify TCP flags in match criteria, so the SDN switch will notify the controller of each TCP packet sent to a virtual IP which did not match a connection-specific rule (described below)
  - Notify the controller when a client issues an ARP request for the MAC address associated with a virtual IP
  - Match all other packets against the rules in the next table in the switch (described below)

These rules should be installed when a switch joins the network.

- Install connection-specific rules for each new connection to a virtual IP to:
  - Rewrite the destination IP and MAC address of TCP packets sent from a client to the virtual IP
  - Rewrite the source IP and MAC address of TCP packets sent from server to client

Connection-specific rules should match packets on the basis of Ethernet type, source IP address, destination IP address, protocol, TCP source port, and TCP destination port. Connection-specific rules should take precedence over the rules that send TCP packets to the controller, otherwise every TCP packet would be sent to the controller. Therefore, these rules should have a higher priority than the rules installed when a switch joins the network. Also, we want connection-specific rules to be removed when a TCP connection ends, so connection-specific rules should have an idle timeout of 20 seconds.

- Construct and send an ARP reply packet when a client requests the MAC address associated with a virtual IP
- Construct and send a TCP reset packet if the controller receives a TCP packet that is not a TCP SYN

Multiple Tables

Your load balancer application should work in tandem with your layer-3 "Shortest-Path Switching" routing application. To achieve this, you will need to leverage the *multiple tables* feature of OpenFlow switches. When packets first arrive at an OpenFlow switch, they are matched against the rules in table `0`. The actions for these rules can specify that the packets be modified, output, sent to the controller, and/or matched against the rules in a different table.

Your load balancer application should install rules in the table specified in the `table` class variable in the `LoadBalancer` class—set to table `0` in the `loadbalancer.prop` configuration file. The connection-specific rules that modify IP and MAC addresses should include an instruction (see Rule Instructions/Action paragraph below) to match the modified packets against the rules installed by your layer-3 routing application. Since your layer-3 routing application will install rules in the `table` class variable in the `ShortestPathSwitching` class, this instruction should direct packets to the table defined in this class variable. The modified packet will then be matched against these rules and forwarded out the appropriate port.

All packets which are not TCP packets destined for a virtual IP, or packets associated with a connection that has already been assigned to a specific host, should be send directly the table used by your layer-3 routing application.

Sending TCP Resets

Once a particular connection has been assigned to a particular host, all packets for that connection should be directed to that host. However, if no packets are transmitted for more than 20 seconds (specified by the `IDLE_TIMEOUT` constant in the `LoadBalancer` class), then we want to remove the rules that perform the rewriting for that particular connection.

Ideally, the 20 second idle period should only occur once a flow has ended. However, it's possible that an active TCP flow could also go idle for some time. If this happens, an entry could timeout prematurely, and the SDN switch will receive TCP packets destined for the virtual IP for which it has no connection-specific flow table entry that matches. These packets will instead match the lower priority rule that sends any TCP packets destined for the virtual IP to the controller. When the controller receives these TCP packets, which are not TCP `SYN` packets, it should construct and send a TCP reset. You can construct the packet using the classes in the `net.floodlightcontroller.packet` package. You can use the `sendPacket(...)` method in the `SwitchCommands` class to send the packet.

Sending ARP Packets

When a client wants to initiate a connection with the virtual IP, it will need to determine the MAC address associated with the virtual IP using ARP. The client does not know the IP is virtual, and since it's not actually assigned to any host, your SDN application must take responsibility for replying to these requests.

You can construct an ARP reply packet using the classes in the `net.floodlightcontroller.packet` package. You can use the `sendPacket(...)` method in the `SwitchCommands` class to send the packet.

Rule Instructions/Actions

When a rule should send a packet to the controller, the rule should include an `OFInstructionApplyActions` whose set of actions consists of a single `OFActionOutput` with `OFPort.OFPP_CONTROLLER` as the port number.

When a rule should rewrite the destination IP and MAC addresses of a packet, the rule should include an `OFInstructionApplyActions` whose set of actions consists of:

- An `OFActionSetField` with a field type of `OFOXMFieldType.ETH_DST` and the desired MAC address as the value
- An `OFActionSetField` with a field type of `OFOXMFieldType.IPV4_DST` and the desired IP address as the value

The actions for rewriting the source IP and MAC addresses of a packet are similar.

When a packet should be processed by the SDN switch based on the rules installed by your layer-3 routing application, a rule should include an `OFInstructionGotoTable` whose table number is the value specified in the table class variable in the `ShortestPathSwitching` class.

4.3 Testing and Debugging

You should test your code by issuing web requests (using curl) from a client host to the virtual IPs.

You can add or remove virtual IPs and hosts by modifying the `loadbalancer.prop` file.

To see which packets a host is sending/recieving run:

```
$ tcpdump -v -n -i hN-eth0
```

replacing `N` with the host's number.

5. Evaluation:

This project is worth 100 points (extra credit not included). You will be graded on both the completeness and accuracy of your program, as follows:

- Part 3 functionality [70 points].
- Part 4 functionality [25 points]
- Style [5 points]:
  1. Coding Style: Well-structured, well documented, clean code, with well defined interfaces between components. Appropriate use of comments, clearly identified variables, constants, function names, etc.
  2. Assignment Layout:
     - Assignment is neatly assembled on 8 1/2 by 11 paper
     - Cover page with your name (last name first followed by a comma then first name), username and section number with a signed statement of independent effort is included
     - Program and documentation submitted for Assignment #8 are satisfactory
     - File name is correct.
- Part 3 extra credit questions [20 points (10 each)]:

6. What to Submit

a. All of your source code files for part 3 (apps/sps directory) and part 4 (apps/loadbalancer directory).
b. tests.txt/doc - file containing a brief description of your testing and debugging methods for part 2 and part 4.
c. vulnerabilities.txt/doc – file identifying at least one vulnerability in your current implementation for each of part 3 and part 4.
d. readme.txt: file containing a thorough description of your design and implementation for part 3 and 4. Please note that **all** code that you do not freshly write for this assignment must be clearly documented in this readme.txt file.
e. Report document that describes your project briefly, explains your design, outlines some of the implementation details, and provides as assessment of what went well and not so well in your project. Problems should be clearly stated and solution approaches should be clearly documented (i.e., both current and new features that you implemented). You should also clearly document any simulation or modeling used in your approach and any evaluation metrics you used for comparative analysis of the current and

new solutions. The format for your final project report should be similar to the standard conference paper formats and should include (at the very least) sections on Introduction, Related Work, Proposed Solution/Architecture/Algorithms, Simulation/Implementation, Results, and Conclusion.

> Submissions that are incomplete or will not compile, link, or run according to your instructions will be returned to you and you will have to re-submit and accept an incomplete in the course and a loss of one half letter grade when you resubmit. No resubmitted assignment will be accepted after midnight on May 13, 2018.

7. **Upload your final project deliverables via NYU Classes.**

8. **Acknowledgements**

   The code and instructions for this project is partially based on assignments developed at University of Wisconsin and at Brown University.

   Note that the original software packages provided are as follows:
   `edu.wisc.cs.sdn.apps.loadbalancer` and `edu.brown.cs.sdn.apps.sps`
   It is fine to refactor these packages as follows for the purpose of this project:
   `edu.nyu.cs.sdn.apps.loadbalancer` and `edu.nyu.cs.sdn.apps.sps`
   Also note that `edu.wisc.cs.sdn.apps.l3routing` is not used in this project.

VI. Deliverables

1. Electronic Archive:

   Your project submission archive file must be uploaded via NYU Classes. The file must be created and sent by the deadline. After the deadline, the project is late. The email clock is the official clock.

   Your project submission archive file should contain your report file as well as your program source code packaged as a .jar file. The various documentation files submitted should be placed in the .jar file in a separate directory called "project documentation".

   To create the .jar file containing your Java source code (please do not include the class files), change your working directory to the directory where your Java files are located and execute the command:

   jar cvf DCN-Spring2020-FinalProject-**xxx**.jar *

   where **xxx** is **YOUR FULL STUDENT ID.**

Include the jar file in your project zip file and send the zip file as an email attachment to the instructor.

You may send questions to the class mailing list ONLY.

2. Report File (to be included in the electronic archive):

   PDF of your project report.

   The cover page supplied on the next page must be the first page of your project report. Please fill in the blank area for each field.

   NOTE:

   The sequence of the report submission is:
   1. **Cover sheet**
   2. **Assignment Answer Sheet(s)**

VII. Sample Cover Sheet:

Name _____ Username: _____ Date: _____
(last name, first name)
Section: _____

Assignment 8: Final Project

_____ Part 3 functionality [Max 70 points]
_____ Part 4 functionality [Max 25 points]
_____ Style [Max 5 points]
_____ Part 3 extra credit questions [Max 20 points (10 each)]

_____Total [Max 100 points + Extra Credit points]

Total in points: _____

Total in extra credit points: _____

Professor's Comments:

Affirmation of my Independent Effort: _____
(Sign here)