Lab 4: Software Defined Networking
CS640 Fall 2022

Released: Tuesday, Nov 8 2022
Due: Friday, Dec 2 2022 11:59pm
Points: 60

Important notes

- All answers and code that you write for this lab must be your own work. Plagiarism is a serious violation of university statutes and all answers will be thoroughly checked for it.
- Ensure that your answer has proper headings and subheadings wherever necessary.
- Late policy:
    - Upto 30 minutes late — lose 0% of points
    - Upto 24 hours late — lose 10% of points
    - Upto 48 hours late — lose 30% of points
    - Upto 72 hours late — lose 60% of points
    - Beyond 72 hours — lose 100% of points

Overview

For this lab, you will implement two control applications for a software defined network (SDN). A layer-3 routing application will install rules in SDN switches to forward traffic to hosts using the shortest, valid path through the network. A distributed load balancer application will redirect new TCP connections to hosts in a round-robin order.

Learning Objectives

After completing this lab, students should be able to:

- Contrast SDN applications and traditional network control planes
- Create SDN applications that use proactive or reactive flow installation

==========

Part 1: Getting Started

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. **You should use Mininet 2.2.2 running on Ubuntu 14 for this project (link)**

Preparing Your Environment

Before beginning this project, there are some additional steps you need to complete to prepare your VM. **Please following the instructions below carefully**:

1. **Install required packages**
   ```
   sudo apt update
   sudo apt install -y curl traceroute ant openjdk-7-jdk git iputils-arping
   ```

2. **Download Floodlight:**
   ```
   cd ~
   git clone https://github.com/parthosa/floodlight-plus.git
   ```

3. **Download the starter code:**
   ```
   cd ~
   mkdir lab4
   cd lab4
   wget https://pages.cs.wisc.edu/~mgliu/CS640/F22/labs/lab4/lab4.tgz --no-check-certificate
   tar xzvf lab4.tgz
   ```

4. **Symlink Floodlight**
   ```
   cd ~/lab4/
   ln -s ~/floodlight-plus
   ```

5. **Patch Floodlight**
   ```
   cd ~/lab4/floodlight-plus
   patch -p1 < ~/lab4/floodlight.patch
   ```

If you want to use Eclipse (on a machine other than your VM) for development, you should do the following:

1. `cd floodlight-plus`
   `ant eclipse`

2. Open Eclipse and create a new workspace. Do not choose the `floodlight-plus` or `lab4` folders as the location for your workspace.
3. Select *File* → *Import* → *General* → *Existing Projects into Workspace*. Click *Next*.

4. From *Select root directory* click *Browse*. Select the parent folder containing your `floodlight-plus` and `lab4` folders.

5. Check the boxes for `floodlight-plus` and `lab4`. Click *Finish*.

6. Exclude the `src/test/java` folder from the build path:
   a. Right click on this folder in the *Project Explorer* pane.
   b. Select *Build Path* →*Remove from Build Path*.
7. Add the `floodlight-plus` project to the build path of `lab4`:
   a. Right click `lab4` in the *Project Explorer pane*.
   b. Select *Build Path* →*Configure Build Path*.
   c. Under the *Projects* tab; click *Add*.
   d. Check the `floodlight-plus` project; click OK.

Running Your Control Applications

1. Compile Floodlight and your applications:
   `cd ~/lab4/`
   `ant clean && ant`
   This will produce a jar file `FloodlightWithApps.jar` that includes the compiled code for Floodlight and your SDN applications. If the build fails, try to execute ant clean and ant separately.

2. Start Floodlight and your SDN applications:
   `java -jar FloodlightWithApps.jar -cf loadbalancer.prop`
   The above command will start both your layer-3 routing and load balancer applications.

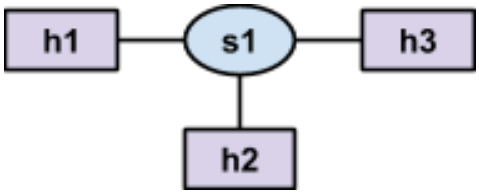
   Note - You should use the same loadbalancer.prop for both L3 and Load Balancer applications.

   When Floodlight starts, you should see output like the following:
   `14:18:13.144 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from file loadbalancer.prop`

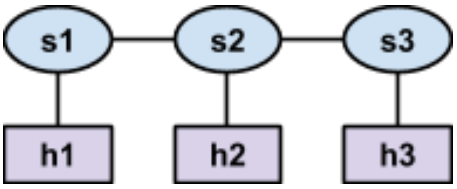
```
14:18:13.377 INFO [n.f.c.i.Controller:main] Controller role set to MASTER
14:18:13.380 INFO [n.f.c.i.Controller:main] Flush switches on reconnect -- Disabled
14:18:13.389 INFO [ArpServer:main] Initializing ArpServer...
14:18:13.390 INFO [L3Routing:main] Initializing L3Routing...
14:18:13.390 INFO [LoadBalancer:main] Initializing LoadBalancer...
14:18:13.392 INFO [LoadBalancer:main] Added load balancer instance: 10.0.100.1 00:00:01:00:00:01 10.0.0.2,10.0.0.3
14:18:13.392 INFO [LoadBalancer:main] Added load balancer instance: 10.0.110.1 00:00:01:10:00:01 10.0.0.4,10.0.0.6
14:18:14.532 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF
14:18:14.552 INFO [ArpServer:main] Starting ArpServer...
14:18:14.552 INFO [L3Routing:main] Starting L3Routing...
14:18:14.553 INFO [LoadBalancer:main] Starting LoadBalancer...
14:18:14.642 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local configuration
14:18:14.644 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]
14:18:14.677 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:6642
14:18:14.761 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 (alternatively, you can also use screen. Install screen in your VM by running: `sudo apt install screen`)

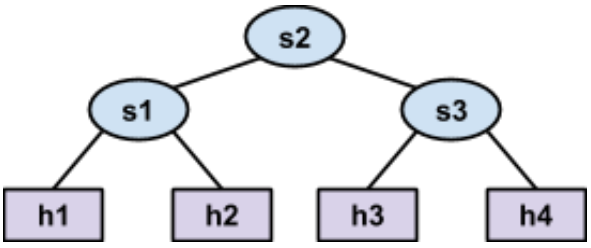
3. Start Mininet:
   cd ~/lab4
   `sudo python2 run_mininet.py single,3`
   The above command will create a topology with a single SDN switch (`s1`) and three hosts (`h1 - h3`) directly connected to the switch:



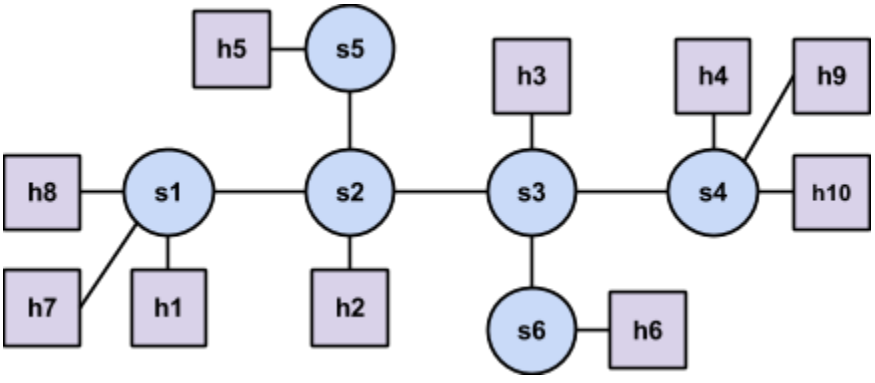
- You can change the number of hosts by changing the numeric value included in the arguments to the `run_mininet.py` script. You can also start Mininet with six other topologies:
  - `linear,n`: a chain of `n` switches with one host connected to each switch; for example, `linear,3` produces the following topology:



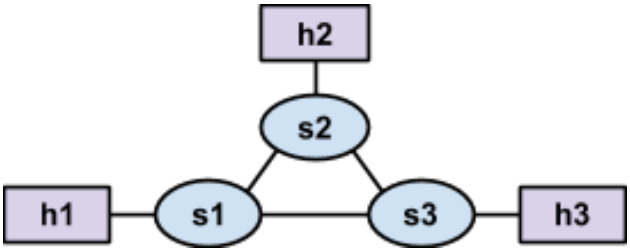
  - `tree,n`: a tree of depth `n` with a single root switch (`s1`) and two hosts connected to each leaf switch; for example `tree,2` produces the following topology:



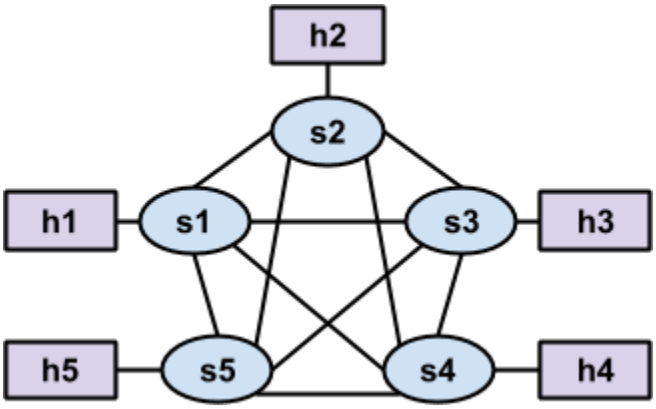
  - `assign1`: creates the topology from Part 3 of Lab 1:



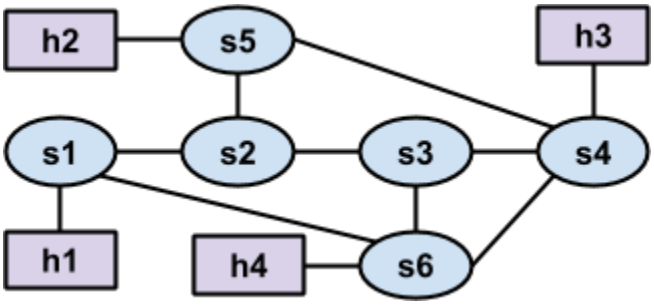
  - `triangle`: creates a topology like the `triangle_rt` topology from Lab 3:



- `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:

```
14:18:40.229 INFO [n.f.c.i.OFChannelHandler:New I/O server
worker #2-1] New switch connection from /127.0.0.1:55876
14:18:40.236 INFO [n.f.c.i.OFChannelHandler:New I/O server
worker #2-1] Disconnected switch [/127.0.0.1:55876 DPID[?]]
14:18:40.363 INFO [n.f.c.i.OFChannelHandler:New I/O server
worker #2-2] New switch connection from /127.0.0.1:55878
14:18:40.880 INFO [n.f.c.i.OFChannelHandler:New I/O server
worker #2-2] Switch OFSwitchBase [/127.0.0.1:55878
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: 2.13.3, S/N: None]
14:18:40.883 INFO [n.f.c.OFSwitchBase:New I/O server worker
#2-2] Clearing all flows on switch OFSwitchBase
[/127.0.0.1:55878 DPID[00:00:00:00:00:00:00:01]]
14:18:40.886 WARN [n.f.c.i.C.s.notification:main] Switch
00:00:00:00:00:00:00:01 connected.
14:18:40.886 INFO [L3Routing:main] Switch s1 added
14:18:40.886 INFO [LoadBalancer:main] Switch s1 added
14:18:40.915 INFO [L3Routing:Topology Updates] Link s1:0 ->
host updated
14:18:40.991 INFO [L3Routing:Topology Updates] Link s1:1 ->
host updated
14:18:40.992 INFO [L3Routing:Topology Updates] Link s1:2 ->
host updated
14:18:40.993 INFO [L3Routing:Topology Updates] Link s1:3 ->
host updated
```

4. You can now run commands (e.g., ping) in mininet like you did for the previous labs. However, until you complete Part 2 and Part3, no packets will be forwarded.

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.

Resources

If you have questions about SDN, OpenFlow, or Floodlight you may want to consult:

- OpenFlow 1.3 Standard — sections 2, 3, and 5.1 - 5.4 are likely to be the most useful
- Floodlight-plus Javadoc

==========

Part 2: Layer-3 Routing Application

Background

In Lab 3, you wrote a router control plane that used RIP to construct a route table. The route table was then used by your router's data plane to determine where (i.e., out which interface) a packet should be forwarded.

Your first SDN application will also construct route tables. However, rather than exchanging RIP messages between routers, your application will construct route tables based on a global view

of the network topology. The appropriate route table will be installed in each SDN switch by your application, and each SDN switch will forward packets according to the installed route table.

It is important to note that SDN switches do not directly support the route table structure used in the last lab. Rather, SDN switches use a more general *flow table* structure, which can accommodate route tables (used in traditional layer-3 routers), 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.

Your layer-3 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 to a specific port on the SDN switch. (You'll use other match criteria and additional instructions/actions for the other SDN application you write, which is described in Part 3.) 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. There is no concept of a *gateway* in SDN flow tables, but that's okay—your router only used the gateway to determine how to rewrite a packet's destination MAC address to ensure correct layer-2 forwarding, and we aren't using traditional layer-2 forwarding in SDN.

Code Overview

The code for your layer-3 routing application will reside in `L3Routing.java` in the `edu.wisc.cs.sdn.apps.l3routing` package. 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—`deviceAdded(...)`, `deviceRemoved(...)`, `deviceMoved(...)`, `switchAdded(...)`, `switchRemoved(...)`, and `linkDiscoveryUpdate(...)`

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

- An SDN application 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`

TODOs

You need to complete the TODOs in `L3Routing.java` to install and remove flow table entries from SDN switches such that traffic is forwarded to a host using the shortest path. Note that Floodlight creates only one instance (i.e., object) of the `L3Routing` class; these instances handle all switches.

Computing Shortest Paths

You should use the Bellman-Ford algorithm 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. (Note: You can use other all pair shortest path algorithms as well)

There will be two link objects between pairs of switches, one in each direction. Due to the way links are discovered, there may be a short period of time (tens of milliseconds) where the controller has a link object only in one direction. If you have a link object in one direction, you can assume the physical link is bidirectional.

When a host joins the network, both the `deviceAdded(...)` and `linkDiscoveryUpdate(...)` event handlers will be called. There are no guarantees on which order these event handlers are called. Thus, a host may be added but we may not yet know which switch it is linked to. The `isAttachedToSwitch()` method in the `Host` class will return true if we know the switch to which a host is connected, otherwise it will return false. If the method returns false, then you do not need to install any rules to route traffic to this host.

You can assume the following will always hold true in the network:

- The network is a connected graph. In other words, there will always be at least one possible path between every pair of switches.
- There is only one physical link between a pair of switches.
- Links are undirected. However, be aware that Floodlight maintains a Link object for each direction (i.e., there are two Link objects for each physical link).

Installing Rules

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 IP is the IP 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; you must set the Ethernet Type before you set the destination IP. 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 3. For now, you should install rules in the table specified in the table class variable in the `L3Routing` class. Also, your rules should never timeout and have a default priority (both defined as constants in the `SwitchCommands` class).

==========

Part 3: Distributed Load Balancer Application

Background

Networks employ load balancing to distribute client requests among a collection of hosts running a specific service (e.g., a web server).

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 IPs and a set of hosts among which connections to the virtual IPs 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 predefined 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).

Code Overview

The code for your load balancer application will reside in `LoadBalancer.java` in the `edu.wisc.cs.sdn.apps.loadbalancer` package. The file we 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.

TODOs

You need to complete the TODOs in `LoadBalancer.java` to:

- Install rules in every switch (when it joins the network) 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)
- 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

Multiple Tables

Your load balancer application should work in tandem with your layer-3 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 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 `L3Routing` 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 sent directly to the table used by your layer-3 routing application.

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 `L3Routing` class.

==========

Testing and Debugging - L3 Routing

You should test your code by sending traffic between various hosts in the network topology—mininet built-in `pingall` command is very useful for this.

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. [NOTE: You may need to explicitly send a message for the link discovery to detect the host. Use `h1 arping -c 2 -A -I h1-eth0 10.0.0.1`. Accordingly replace hostname and IP.]

- 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.

Testing and Debugging - LoadBalancer

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/receiving run:

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

replacing N with the host's number.

==========

Submission Instructions (Read carefully: Instructions have changed)

You must submit a single tar file of the src directory containing the Java source files for your SDN applications. Please submit the entire src directory and a README that contains any information needed for the TAs along with your group member names and NetIDs; do not submit any other files or directories. To create the tar file, run the following command, replacing username1 and username2 with the WISC username (NetID) of each group member:

IMPORTANT: username1 and username2 should be NetID, not CS username (this is different from Lab 3)

```
tar czvf username1_username2.tgz src README.md
```

Upload the tar file to Canvas. Please submit only one tar file per group.