**Programming Assignment 4: Software Defined Networking**

**Overview**

For this project you will implement two control application 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.

[Step 1: Getting Started](#STEP1)

[Step 2: Layer-3 Routing Application](#STEP2)

[Step 3: Distributed Load Balancer Application](#STEP3)

**Learning Objectives**

After completing this programming assignment, students should be able to:

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

**Clarifications**

* You should make the following edit to the L3Routing.java file we provided:  
  1) Change the line this.knownHosts.remove(host) in the deviceRemoved(...) event handler to this.knownHosts.remove(device)

————————————————————————————————————

如果使用云平台的环境，只需执行“Preparing Your Environment”中的第4.步，剩余不用执行（5. 6. 步也不需要执行）

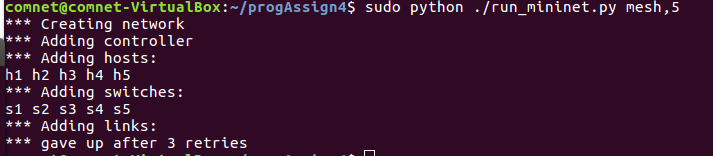
————————————————————————————————————

**Step 1: Getting Started**

Your SDN applications will be written in Java and run atop the [Floodlight](http://www.projectfloodlight.org/floodlight/) OpenFlow controller. You will use [Mininet](http://mininet.org/) to emulate a variety of network topologies consisting of OpenFlow switches and hosts. You should continue to use your Mininet virtual machine (VM) for this project.

**Preparing Your Environment**

Before beginning this project, there are some additional steps you need to complete to prepare your VM:



1. Install required packages

sudo apt-get update  
sudo apt-get install -y curl traceroute ant openjdk-7-jdk git iputils-arping

1. Download Floodlight:  
           cd ~  
           git clone https://bitbucket.org/sdnhub/floodlight-plus.git
2. Get the starter codes and

cd ~/progAssign4

1. Symlink Floodlight  
           cd ~/progAssign4/  
           ln -s ../floodlight-plus
2. Patch Floodlight  
           cd ~/progAssign4/floodlight-plus  
           patch -p1 < ~/progAssign4/floodlight.patch
3. patch Mininet

cd mininet/mininet

patch -p1 < ~/progAssign4/mininet.patch

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

（目前eclipse使用好像有些问题，可用使用vscode等）

1. cd floodlight-plus  
   ant eclipse
2. Open Eclipse and create a new workspace. **Do not** choose the floodlight-plus or progAssign4 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 progAssign4 folders.
5. Check the boxes for floodlight-plus and progAssign4. Click *Finish*.
6. Exclude the src/test/java folder from the build path:
7. Right click on this folder in the *Project Explorer*pane.
8. Select *Build Path* →*Remove from Build Path*.
9. Add the floodlight-plus project to the build path of progAssign4:
10. Right click progAssign4 in the *Project Explorer*pane.
11. Select *Build Path* →*Configure Build Path*.
12. Under the *Projects* tab; click *Add*.
13. Check the floodlight-plus project; click OK.

**Running Your Control Applications**

1. Compile Floodlight and your applications:

  cd ~/progAssign4/  
 ant

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

1. 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. You can start both your layer-3 routing and load balancer applications by using loadbalancer.prop for the -cf (configuration file) argument.

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

19:15:37.239 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from file l3routing.prop

19:15:37.666 INFO [n.f.c.i.Controller:main] Controller role set to MASTER

19:15:37.676 INFO [n.f.c.i.Controller:main] Flush switches on reconnect -- Disabled

19:15:37.697 INFO [ArpServer:main] Initializing ArpServer...

19:15:37.697 INFO [L3Routing:main] Initializing L3Routing...

19:15:38.835 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF

19:15:38.891 INFO [ArpServer:main] Starting ArpServer...

19:15:38.892 INFO [L3Routing:main] Starting L3Routing...

19:15:39.122 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local configuration

19:15:39.122 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]

19:15:39.185 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:6642

19:15:39.421 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. Now is a time to use [screen](https://www.linux.com/learn/taking-command-terminal-gnu-screen%20). You can install screen in your VM by running: sudo apt-get install screen.

Note:

Start Floodlight and your SDN applications时可能会遇到6633端口占用问题

原因为6633端口会被未正常关闭的ovs-controller默认占用

——————————————————————————————————

解决方法：

查看端口占用

- sudo netstat -anp | grep 6633

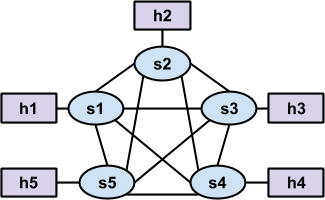
kill 占用进程

- sudo kill -9 <pid>

再启动Floodlight

—————————————————————————————————

1. Start Mininet:  
     sudo python ./run\_mininet.py mesh,5  
   The above command will create a topology with 5 SDN switches and 5 hosts:

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


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

11:04:44.960 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] New switch connection from /127.0.0.1:50921

11:04:44.999 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] Disconnected switch [/127.0.0.1:50921 DPID[?]]

11:04:46.278 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] New switch connection from /127.0.0.1:50922

11:04:46.355 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Switch OFSwitchBase [/127.0.0.1:50922 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: 2.0.1, S/N: None]

11:04:46.364 INFO [n.f.c.OFSwitchBase:New I/O server worker #2-2] Clearing all flows on switch OFSwitchBase [/127.0.0.1:50922 DPID[00:00:00:00:00:00:00:01]]

11:04:46.377 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 connected.

11:04:46.383 INFO [L3Routing:main] Switch s1 added

11:04:46.432 INFO [L3Routing:Topology Updates] Link s1:0 -> host updated

11:04:46.484 INFO [L3Routing:Topology Updates] Link s1:2 -> host updated

11:04:46.489 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 port s1-eth1 changed: UP

11:04:46.585 INFO [L3Routing:Topology Updates] Link s1:1 -> host updated

1. You can now run commands (e.g., ping) in mininet like you did for the previous assignments. However, until you complete Step 2, 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](https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf)
* [Floodlight-plus Javadoc](https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343622/Javadoc+Entry)

**Step 2: Layer-3 Routing Application**

**Background**

In Programming Assignment 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 programming assignment. 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 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 [Step 3](#STEP3).) The match criteria serves 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. 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 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 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; this instance handles 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’ ∈ H*, *h ≠ 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.

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 [Step 3](#STEP3). 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).

**Testing and Debugging (You must show this to TA face-to-face on time!)**

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.

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):

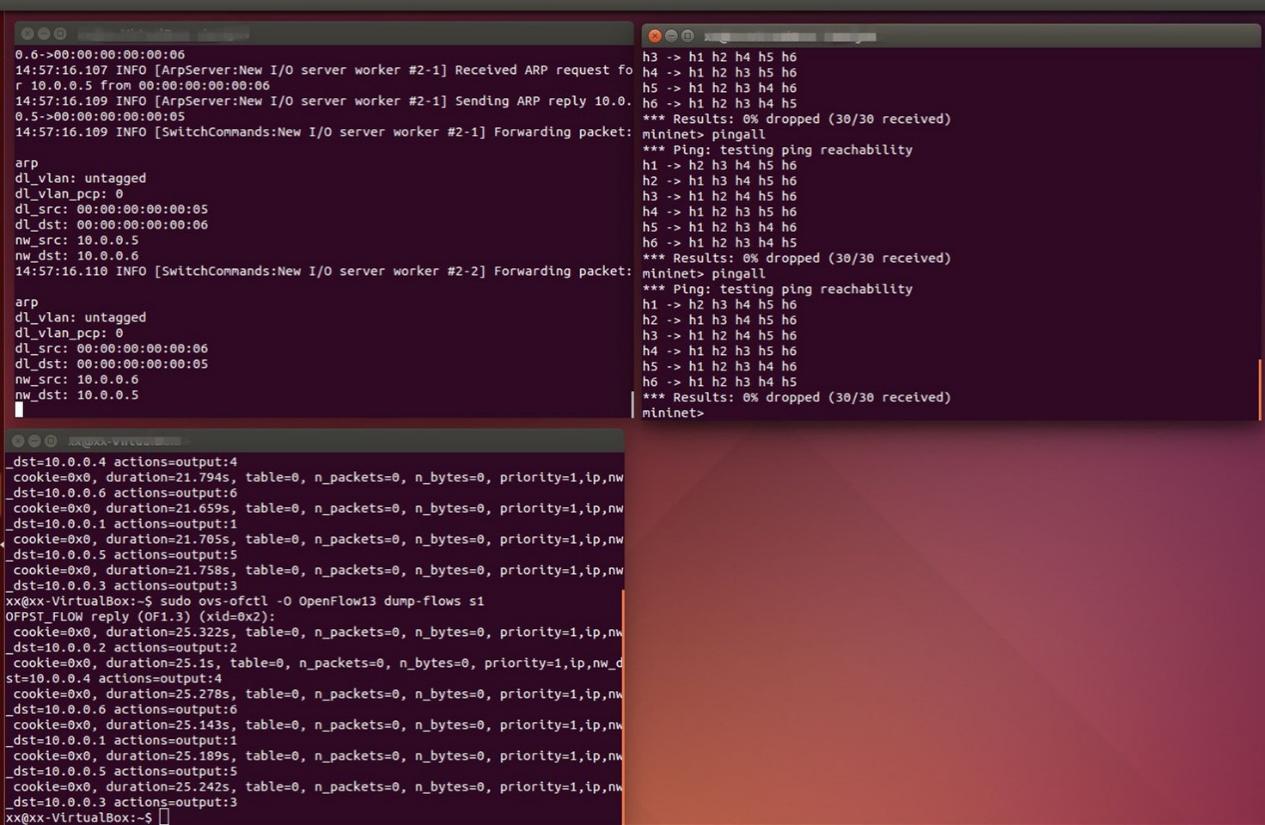
sudo ovs-ofctl -O OpenFlow13 dump-flows s1

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.

When you finish this, your screen may be like this:



**Step 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). In class, we 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 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 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).

**Code Overview**

The code for your load balancer application will reside in LoadBalancer.java in the 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.

**Step 4: Other network-control applications proposed by yourself (Optional)**

You are encouraged to propose and implement an additional network-control application, e.g., firewall, and traffic engineering.

**TODOs**

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

* Install rules in every switch (when it joins the netw············ ork) 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](#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](#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 send directly 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(You must show this to TA face-to-face on time!)**

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

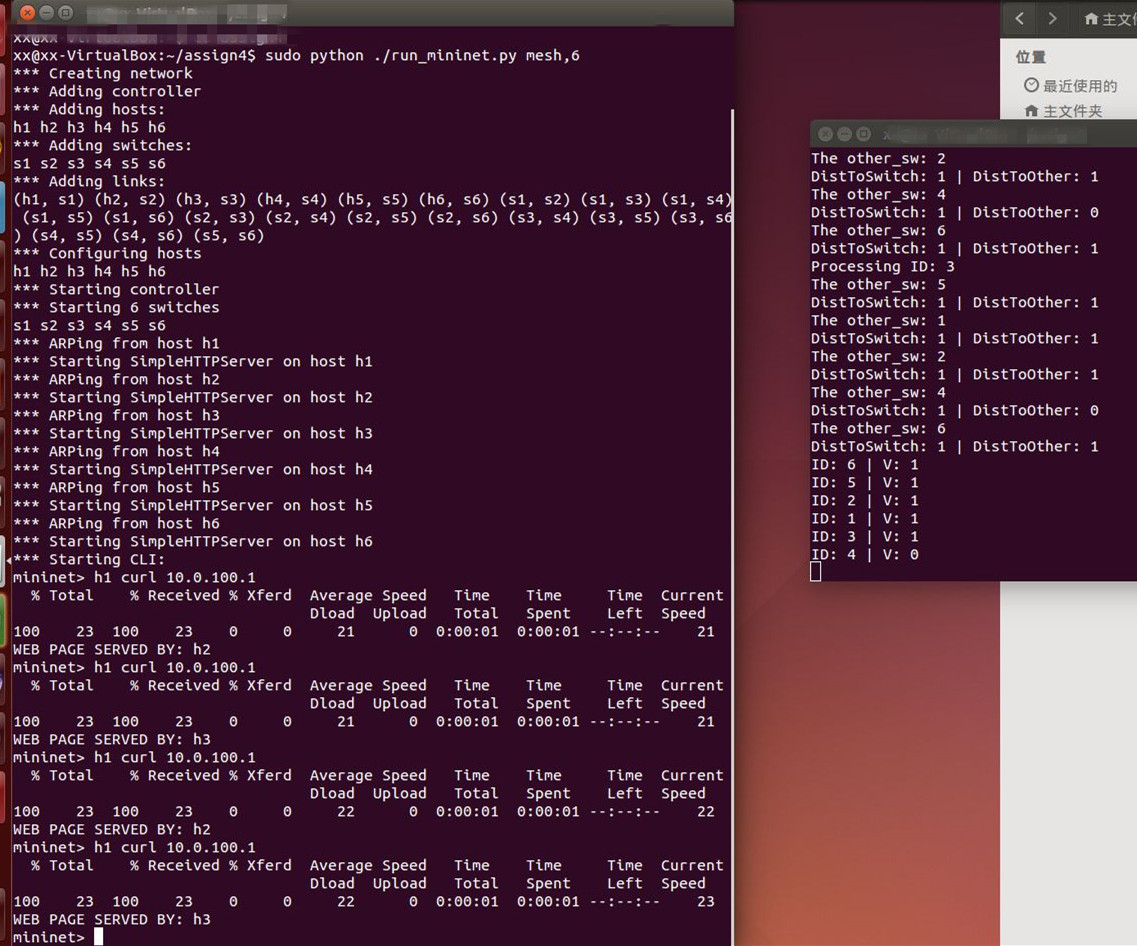
You can add or remove virtual IPs and hosts by modifying the loadbalancer.prop file.

To see which packets a host is sending/receiving, run:

tcpdump -v -n -i hN-eth0

replacing N with the host’s number.

When you finish this, your screen may be like this:



**Submission Instructions**

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; do not submit any other files or directories. Also, you should submit a readme file.

You must submit a single tar file named assign4\_username1\_username2 containing the above. Please submit only one tar file per group.

**Grading**

1) Correctness 80%.

L3Routing 40%

Distributed Load Balance 40%

2) Additional network-control application (up to extra 20%)

3) Code quality 20% (Dont copy from others! )

Make code more readable.

**Acknowledgements**

We acknowledge CS 640: Computer Networks with University of Wisconsin-Madison.