Name _____Luo Lingwei_____ Username: _____Luo Lingwei_____ Date: ___12/22___

(last name, first name)

Section: _____9_____

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: _____Luo Lingwei_____ (Sign here)

Introduction

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, I will implement the logic in such a controller to manage the following:

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

Related Work

Over the past few years, SDN has gained significant traction in industry. Many commercial switches support the OpenFlow API. Initial vendors that supported OpenFlow included HP, NEC, and Pronto; this list has since expanded dramatically. Many different controller platforms have emerged [1-7]. Programmers have used these platforms to create many applications, such as dynamic access control [8, 9], server load balancing [10,11], network virtualization [12,13], energyefficient networking [14], and seamless virtual-machine migration and user mobility [15]. Early commercial successes, such as Google's wide-area traffic-management system [16] and Nicira's Network Virtualization Platform [17], have garnered significant industry attention. Many of the world's largest information-technology companies (e.g., cloud providers, carriers, equipment vendors, and financial-services firms) have joined SDN industry consortia like the Open Networking Foundation [18] and the Open Daylight initiative [19].

Although the excitement about SDN has become more palpable during the past few years, many of the ideas underlying SDN have evolved over the past twenty years (or more!). In some ways, SDN revisits ideas from early telephony networks, which used a clear separation of control and data planes to simplify network management and the deployment of new services. Yet, open interfaces like OpenFlow enable more innovation in controller platforms and applications than was possible on closed networks designed for a narrow range of telephony services. In other ways, SDN resembles past research on active networking, which articulated a vision for programmable networks, albeit with an emphasis on programmable data planes. SDN also relates to previous work on separating the control and data planes in computer networks.

The history begins twenty years ago, just as the Internet takes off, at a time when the Internet's amazing success exacerbated the challenges of managing and evolving the network infrastructure. We focus on innovations in the networking community (whether by researchers, standards bodies, or companies), although we recognize that these innovations were in some cases catalyzed by progress in other areas, including distributed systems, operating systems, and programming languages. The efforts to create a programmable network infrastructure also clearly relate to the long thread of work on supporting programmable packet processing at high speeds [20-26].

The etymology of the term "SDN" is itself complex, and, although the term was initially used to describe Stanford's OpenFlow project, the definition has since expanded to include a much wider array of technologies. (The term has even been sometimes co-opted by industry marketing departments to describe unrelated ideas that predated Stanford's SDN project.) Thus, instead of attempting to attribute direct influence between projects, we instead highlight the evolution of and relationships between the ideas that represent the defining characteristics of SDN, regardless of whether or not they directly influenced specific subsequent research. Some of these early ideas may not have directly influenced later ones, but we believe that the connections between the concepts that we outline are noteworthy, and that these projects of the past may yet offer new lessons for SDN in the future.

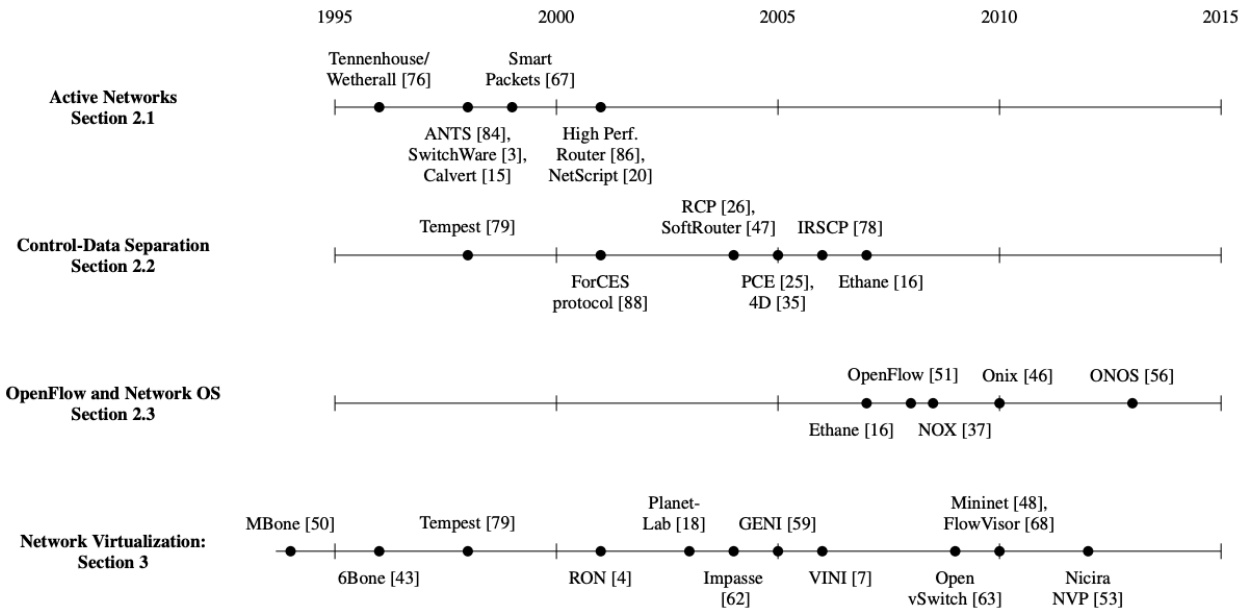


Figure 1: Selected developments in programmable networking over the past 20 years, and their chronological relationship to advances in network virtualization (one of the first successful SDN use cases).

Proposed Solution

- Shortest-Path Switching Routing Application

  I will build a global shortest-path switching table and install forwarding rules on the switches to implement these paths. I will build this table on the controller based on global topology information the controller gathers. My 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 my 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— my 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.

  My 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. 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, my network will resemble a network in which all switches have converged with the spanning tree protocol and MAC learning, with one important difference: my topology is not constrained to a tree, as I am installing paths individually and loops should not be a problem. In fact, I must test that my solution works on topologies with loops.

  After the rules are all installed, the process a host will go through to send an 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 my code), the packet will match on the destination MAC address and be forwarded on the correct port.

- Distributed Load Balance Routing Application

  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.

  My second SDN application will implement the same functionality as a set of hardware load balancers. My 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. My 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. I will also instruct the SDN switch to match the modified packets against the flow rules installed by my layer-3 routing application and apply the appropriate actions (i.e., send the packets out the appropriate ports).

Implementation

- Shortest-Path Switching Routing Application

(1) 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.

```java
/**
 * New mthod to append rules
 */
public void appendRule(IOFSwitch curSwitch, int switchPort, int ipAddress) {
    OFMatch matchCondition = new OFMatch();
    matchCondition.setDataLayerType(OFMatch.ETH_TYPE_IPV4);
    matchCondition.setNetworkDestination(ipAddress);

    List<OFInstruction> instructions = Methods.redirectToPort(switchPort);
    SwitchCommands.removeRules(curSwitch, table, matchCondition);
    SwitchCommands.installRule(curSwitch, table, SwitchCommands.DEFAULT_PRIORITY, matchCondition, instructions);
}

/**
 * New mthod to update rules for switch
 */
public void updateSwitchRules() {
    // log.info(String.format("Enter updateSwitchRules!!!"));

    for (long curSwitchId: this.netgraph.table.keySet()) {
        Map<Long, LDPair> linkMap = this.netgraph.table.get(curSwitchId);

        IOFSwitch curSwitch = this.floodlightProv.getSwitch(curSwitchId);

        for (Host host : getHosts()) {
            long destinationSwitch = host.getSwitch().getId();
            if (!linkMap.containsKey(destinationSwitch)) continue;
            LDPair ldPair = linkMap.get(destinationSwitch);

            int port;
            if (ldPair.link.getSrc() == curSwitchId) port = ldPair.link.getSrcPort();
            else port = ldPair.link.getDstPort();

            appendRule(curSwitch, port, host.getIPv4Address());
        }
    }
}
```

Leverage methods appendRule and updateSwitchRules to handle deviceAdded, deviceRemoved, deviceMoved, switchAdded, switchRemoved, linkDiscoveryUpdate events.

(2) 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

```
public void updateTable(Collection<Link> links) {
    for (Link link : links) {
        addSwitch(link.getSrc());
        addSwitch(link.getDst());
        table.get(link.getSrc()).put(link.getDst(), new LDPair(link, 1));
        table.get(link.getDst()).put(link.getSrc(), new LDPair(link, 1));
    }

    while (true) {
        boolean update_success = false;
        for (Link link : links) {
            for (long dst : table.keySet()) {
                if (dst == link.getSrc() || dst == link.getDst()) continue;
                LDPair leftLDP = table.get(link.getSrc()).get(dst), rightLDP = table.get(link.getDst()).get(dst);
                if (leftLDP == null && rightLDP == null) continue;
                if (leftLDP == null) {
                    table.get(link.getSrc()).put(dst, new LDPair(link, rightLDP.dist + 1));
                } else if (rightLDP == null) {
                    table.get(link.getDst()).put(dst, new LDPair(link, leftLDP.dist + 1));
                } else if (leftLDP.dist > rightLDP.dist+1) {
                    leftLDP.link = link;
                    leftLDP.dist = rightLDP.dist+1;
                } else if (rightLDP.dist > leftLDP.dist+1) {
                    rightLDP.link = link;
                    rightLDP.dist = leftLDP.dist+1;
                } else {
                    continue;
                }
                update_success = true;
            }
        }
        if (!update_success) break;
    }
}
}
```

Implement Bellman-Ford algorithm in NetGraph class to update our network table.

- Distributed Load Balance Routing Application

(1) (2)

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)

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

```java
/**
 * Event handler called when a switch joins the network.
 * @param DPID for the switch
 */
@Override
public void switchAdded(long switchId)
{
    IOFSwitch curSwitch = this.floodlightProv.getSwitch(switchId);
    log.info(String.format("Switch s%d added", switchId));

    /*********************************************************************/
    /* TODO: Install rules to send:                                      */
    /*       (1) packets from new connections to each virtual load       */
    /*       balancer IP to the controller                               */
    /*       (2) ARP packets to the controller, and                      */
    /*       (3) all other packets to the next rule table in the switch  */

    /*********************************************************************/

    for (Integer virtualIpAddr : instances.keySet()) {

        List<OFInstruction> instrs = Methods.redirectToPort(OFPort.OFPP_CONTROLLER.getValue());

        OFMatch desiredArp = new OFMatch(), desiredTcp = new OFMatch();
        desiredArp.setDataLayerType(OFMatch.ETH_TYPE_ARP);
        desiredArp.setNetworkDestination(virtualIpAddr);
        desiredTcp.setDataLayerType(OFMatch.ETH_TYPE_IPV4);
        desiredTcp.setNetworkDestination(virtualIpAddr);
        desiredTcp.setNetworkProtocol(OFMatch.IP_PROTO_TCP);

        SwitchCommands.removeRules(curSwitch, table, desiredArp);
        SwitchCommands.removeRules(curSwitch, table, desiredTcp);
        SwitchCommands.installRule(curSwitch, table, (byte) 2, desiredArp, instrs);
        SwitchCommands.installRule(curSwitch, table, (byte) 2, desiredTcp, instrs);
    }

    SwitchCommands.installRule(curSwitch, table, SwitchCommands.DEFAULT_PRIORITY, new OFMatch(), Methods.getInstr());

}
```

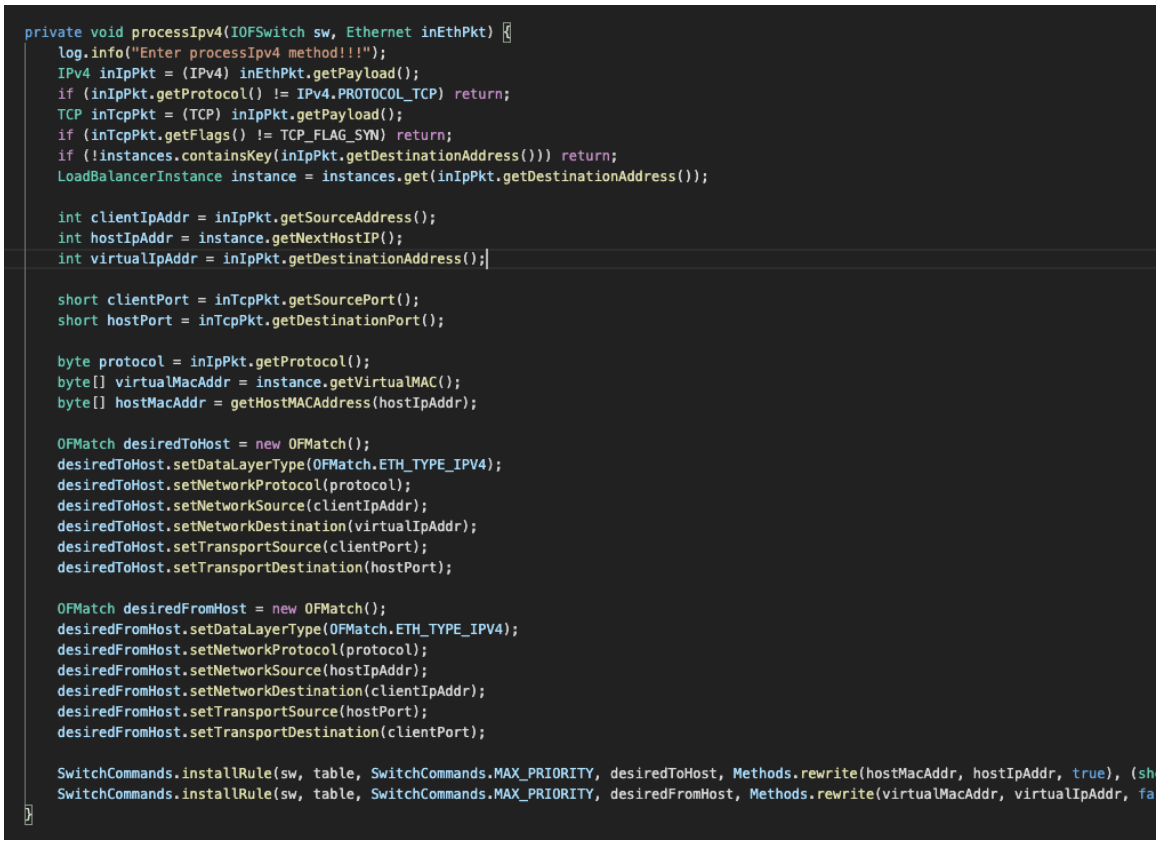
Call SwitchCommands.installRule() to install all required rules in (1) and (2).

(3) Construct and send an ARP reply packet when a client requests the MAC address associated with a virtual IP

```
private void processArp(IOFSwitch sw, Ethernet inEthPkt, short inPort){
    log.info("Enter processArp method!!!");
    ARP inArpPkt = (ARP) inEthPkt.getPayload();
    if (inArpPkt.getOpCode() != ARP.OP_REQUEST) return;
    int ip = IPv4.toIPv4Address(inArpPkt.getTargetProtocolAddress());
    if (!instances.containsKey(ip)) return;
    byte[] virtualMacAddr = instances.get(ip).getVirtualMAC();

    ARP arpOutPkt = new ARP();
    arpOutPkt.setHardwareType(ARP.HW_TYPE_ETHERNET);
    arpOutPkt.setProtocolType(ARP.PROTO_TYPE_IP);
    arpOutPkt.setHardwareAddressLength((byte) Ethernet.DATALAYER_ADDRESS_LENGTH);
    arpOutPkt.setProtocolAddressLength((byte) MACAddress.MAC_ADDRESS_LENGTH);
    arpOutPkt.setOpCode(ARP.OP_REPLY);
    arpOutPkt.setTargetHardwareAddress(inArpPkt.getSenderHardwareAddress());
    arpOutPkt.setTargetProtocolAddress(inArpPkt.getTargetProtocolAddress());
    arpOutPkt.setSenderHardwareAddress(virtualMacAddr);
    arpOutPkt.setSenderProtocolAddress(ip);

    Ethernet outEthPkt = (Ethernet) new Ethernet();
    outEthPkt.setEtherType(Ethernet.TYPE_ARP);
    outEthPkt.setSourceMACAddress(virtualMacAddr);
    outEthPkt.setDestinationMACAddress(inEthPkt.getSourceMACAddress());
    outEthPkt.setPayload(arpOutPkt);

    SwitchCommands.sendPacket(sw, inPort, outEthPkt);
    System.out.println("ARP reply from switch: " + sw.getId() + " on port: " + inPort);
}
```

Handle Arp request and construct ARP reply packet in loadbalancer.

(4) Construct and send a TCP reset packet if the controller receives a TCP packet that is not a TCP SYN



```
private void processIpv4(IOFSwitch sw, Ethernet inEthPkt) {
    log.info("Enter processIpv4 method!!!");
    IPv4 inIpPkt = (IPv4) inEthPkt.getPayload();
    if (inIpPkt.getProtocol() != IPv4.PROTOCOL_TCP) return;
    TCP inTcpPkt = (TCP) inIpPkt.getPayload();
    if (inTcpPkt.getFlags() != TCP_FLAG_SYN) return;
    if (!instances.containsKey(inIpPkt.getDestinationAddress())) return;
    LoadBalancerInstance instance = instances.get(inIpPkt.getDestinationAddress());

    int clientIpAddr = inIpPkt.getSourceAddress();
    int hostIpAddr = instance.getNextHostIP();
    int virtualIpAddr = inIpPkt.getDestinationAddress();

    short clientPort = inTcpPkt.getSourcePort();
    short hostPort = inTcpPkt.getDestinationPort();

    byte protocol = inIpPkt.getProtocol();
    byte[] virtualMacAddr = instance.getVirtualMAC();
    byte[] hostMacAddr = getHostMACAddress(hostIpAddr);

    OFMatch desiredToHost = new OFMatch();
    desiredToHost.setDataLayerType(OFMatch.ETH_TYPE_IPV4);
    desiredToHost.setNetworkProtocol(protocol);
    desiredToHost.setNetworkSource(clientIpAddr);
    desiredToHost.setNetworkDestination(virtualIpAddr);
    desiredToHost.setTransportSource(clientPort);
    desiredToHost.setTransportDestination(hostPort);

    OFMatch desiredFromHost = new OFMatch();
    desiredFromHost.setDataLayerType(OFMatch.ETH_TYPE_IPV4);
    desiredFromHost.setNetworkProtocol(protocol);
    desiredFromHost.setNetworkSource(hostIpAddr);
    desiredFromHost.setNetworkDestination(clientIpAddr);
    desiredFromHost.setTransportSource(hostPort);
    desiredFromHost.setTransportDestination(clientPort);

    SwitchCommands.installRule(sw, table, SwitchCommands.MAX_PRIORITY, desiredToHost, Methods.rewrite(hostMacAddr, hostIpAddr, true), (sh
    SwitchCommands.installRule(sw, table, SwitchCommands.MAX_PRIORITY, desiredFromHost, Methods.rewrite(virtualMacAddr, virtualIpAddr, fa
}
```

Handle IPv4 request and install corresponding rules in loadbalancer.

Results

- Shortest-Path Switching Routing Application

  Start application with shortestPathSwitching.prop

```
[^Cmininet@mininet-VirtualBox:~/openflow$ java -jar FloodlightWithApps.jar -cf shortestPathSwitching.prop ]
16:11:20.449 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from file shortestPathSwitching.pro
p
16:11:21.808 INFO [n.f.c.i.Controller:main] Controller role set to MASTER
16:11:21.856 INFO [n.f.c.i.Controller:main] Flush switches on reconnect -- Disabled
16:11:21.933 INFO [ArpServer:main] Initializing ArpServer...
16:11:21.934 INFO [ShortestPathSwitching:main] Initializing ShortestPathSwitching...
16:11:24.658 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF
16:11:24.743 INFO [ArpServer:main] Starting ArpServer...
16:11:24.748 INFO [ShortestPathSwitching:main] Starting ShortestPathSwitching...
16:11:25.064 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local conf
iguration
16:11:25.065 INFO [o.s.s.i.SyncManager:main] [32767] Updating sync configuration ClusterConfig [allNodes={3
2767=Node [hostname=localhost, port=6642, nodeId=32767, domainId=32767]}, authScheme=NO_AUTH, keyStorePath=
null, keyStorePassword is unset]
16:11:25.178 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:
6642
16:11:25.466 INFO [n.f.c.i.Controller:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6633
16:11:32.142 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] New switch connection from /127.0.0
.1:60634
16:11:32.201 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] Disconnected switch [/127.0.0.1:606
34 DPID[?]]
16:11:34.953 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] New switch connection from /127.0.0
.1:60635
16:11:35.189 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Switch OFSwitchBase [/127.0.0.1:606
35 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]
16:11:35.226 INFO [n.f.c.OFSwitchBase:New I/O server worker #2-2] Clearing all flows on switch OFSwitchBase
 [/127.0.0.1:60635 DPID[00:00:00:00:00:00:00:01]]
16:11:35.238 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 connected.
16:11:35.239 INFO [ShortestPathSwitching:main] Switch s1 added
16:11:35.355 INFO [ShortestPathSwitching:Topology Updates] Link s1:0 -> host updated
16:11:35.401 INFO [ShortestPathSwitching:Topology Updates] Link s1:-2 -> host updated
16:11:35.402 INFO [ShortestPathSwitching:Topology Updates] Link s1:1 -> host updated
16:11:35.403 INFO [ShortestPathSwitching:Topology Updates] Link s1:3 -> host updated
16:11:35.406 INFO [ShortestPathSwitching:Topology Updates] Link s1:2 -> host updated
16:11:36.038 INFO [ShortestPathSwitching:New I/O server worker #2-2] Host h1 added
16:11:36.125 INFO [ShortestPathSwitching:New I/O server worker #2-2] Host h2 added
16:11:37.148 INFO [ShortestPathSwitching:New I/O server worker #2-2] Host h3 added
16:11:42.631 INFO [ArpServer:New I/O server worker #2-2] Received ARP request for 10.0.0.2 from 00:00:00:00
:00:01
16:11:42.633 INFO [ArpServer:New I/O server worker #2-2] Sending ARP reply 10.0.0.2->00:00:00:00:00:02
16:11:42.635 INFO [SwitchCommands:New I/O server worker #2-2] Forwarding packet:
arp
dl_vlan: untagged
dl_vlan_pcp: 0
dl_src: 00:00:00:00:00:02
dl_dst: 00:00:00:00:00:01
nw_src: 10.0.0.2
nw_dst: 10.0.0.1
16:11:42.643 INFO [ArpServer:New I/O server worker #2-2] Received ARP request for 10.0.0.1 from 00:00:00:00
:00:02
16:11:42.649 INFO [ArpServer:New I/O server worker #2-2] Sending ARP reply 10.0.0.1->00:00:00:00:00:01
16:11:42.656 INFO [SwitchCommands:New I/O server worker #2-2] Forwarding packet:
arp
dl_vlan: untagged
dl_vlan_pcp: 0
dl_src: 00:00:00:00:00:01
dl_dst: 00:00:00:00:00:02
nw_src: 10.0.0.1
nw_dst: 10.0.0.2
```

Run mininet with single graph (3 hosts, 1 switch)

Note: Similar test methods and results for (linear,n), (tree,n), (assign1), (triangle), (mesh,n), (someloops).

```
mininet@mininet-VirtualBox:~/openflow$ sudo ./run_mininet.py single,3
[sudo] password for mininet:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** ARPing from host h1
*** Starting SimpleHTTPServer on host h1
*** ARPing from host h2
*** Starting SimpleHTTPServer on host h2
*** ARPing from host h3
*** Starting SimpleHTTPServer on host h3
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

- Distributed Load Balance Routing Application

  Start application with loadbalancer.prop

```
mininet@mininet-VirtualBox:~/openflow$ java -jar FloodlightWithApps.jar -cf loadbalancer.prop
16:23:25.400 INFO [n.f.c.m.FloodlightModuleLoader:main] Loading modules from file loadbalancer.prop
16:23:26.178 INFO [n.f.c.i.Controller:main] Controller role set to MASTER
16:23:26.197 INFO [n.f.c.i.Controller:main] Flush switches on reconnect -- Disabled
16:23:26.244 INFO [ArpServer:main] Initializing ArpServer...
16:23:26.245 INFO [ShortestPathSwitching:main] Initializing ShortestPathSwitching...
16:23:26.246 INFO [LoadBalancer:main] Initializing LoadBalancer...
16:23:26.249 INFO [LoadBalancer:main] Added load balancer instance: 10.0.100.1 00:00:01:00:00:01 10.0.0.2,1
0.0.0.3
16:23:26.250 INFO [LoadBalancer:main] Added load balancer instance: 10.0.110.1 00:00:01:10:00:01 10.0.0.4,1
0.0.0.6
16:23:28.000 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF
16:23:28.127 INFO [ArpServer:main] Starting ArpServer...
16:23:28.129 INFO [ShortestPathSwitching:main] Starting ShortestPathSwitching...
16:23:28.133 INFO [LoadBalancer:main] Starting LoadBalancer...
16:23:28.509 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local conf
iguration
16:23:28.510 INFO [o.s.s.i.SyncManager:main] [32767] Updating sync configuration ClusterConfig [allNodes={3
2767=Node [hostname=localhost, port=6642, nodeId=32767, domainId=32767]}, authScheme=NO_AUTH, keyStorePath=
null, keyStorePassword is unset]
16:23:28.667 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:
6642
16:23:29.075 INFO [n.f.c.i.Controller:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6633
16:23:38.533 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] New switch connection from /127.0.0
.1:60642
16:23:38.609 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-1] Disconnected switch [/127.0.0.1:606
42 DPID[?]]
16:23:41.415 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] New switch connection from /127.0.0
.1:60643
16:23:41.819 INFO [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Switch OFSwitchBase [/127.0.0.1:606
43 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]
16:23:41.837 INFO [n.f.c.OFSwitchBase:New I/O server worker #2-2] Clearing all flows on switch OFSwitchBase
 [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]]
16:23:41.853 WARN [n.f.c.i.C.s.notification:main] Switch 00:00:00:00:00:00:00:01 connected.
16:23:41.854 INFO [ShortestPathSwitching:main] Switch s1 added
16:23:41.856 INFO [LoadBalancer:main] Switch s1 added
16:23:42.110 INFO [ShortestPathSwitching:Topology Updates] Link s1:0 -> host updated
16:23:42.207 INFO [ShortestPathSwitching:Topology Updates] Link s1:-2 -> host updated
16:23:42.211 INFO [ShortestPathSwitching:Topology Updates] Link s1:1 -> host updated
16:23:42.215 INFO [ShortestPathSwitching:Topology Updates] Link s1:3 -> host updated
16:23:42.224 INFO [ShortestPathSwitching:Topology Updates] Link s1:2 -> host updated
16:23:42.242 ERROR [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Error OFPET_BAD_MATCH OFPBMC_BAD_P
REREQ from switch OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]] in state MASTER
16:23:42.253 WARN [n.f.c.i.Controller:New I/O server worker #2-2] Unhandled OF Message: OFError [type=OFPET
_BAD_MATCH, code=OFPBMC_BAD_PREREQ] from OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]]
16:23:42.254 ERROR [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Error OFPET_BAD_MATCH OFPBMC_BAD_P
REREQ from switch OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]] in state MASTER
16:23:42.255 WARN [n.f.c.i.Controller:New I/O server worker #2-2] Unhandled OF Message: OFError [type=OFPET
_BAD_MATCH, code=OFPBMC_BAD_PREREQ] from OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]]
16:23:42.255 ERROR [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Error OFPET_BAD_MATCH OFPBMC_BAD_P
REREQ from switch OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]] in state MASTER
16:23:42.260 WARN [n.f.c.i.Controller:New I/O server worker #2-2] Unhandled OF Message: OFError [type=OFPET
_BAD_MATCH, code=OFPBMC_BAD_PREREQ] from OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]]
16:23:42.261 ERROR [n.f.c.i.OFChannelHandler:New I/O server worker #2-2] Error OFPET_BAD_MATCH OFPBMC_BAD_P
REREQ from switch OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]] in state MASTER
16:23:42.262 WARN [n.f.c.i.Controller:New I/O server worker #2-2] Unhandled OF Message: OFError [type=OFPET
_BAD_MATCH, code=OFPBMC_BAD_PREREQ] from OFSwitchBase [/127.0.0.1:60643 DPID[00:00:00:00:00:00:00:01]]
16:23:42.517 INFO [ShortestPathSwitching:New I/O server worker #2-2] Host h1 added
16:23:42.528 INFO [LoadBalancer:New I/O server worker #2-2] Enter processArp method!!!
16:23:42.555 INFO [ShortestPathSwitching:New I/O server worker #2-2] Host h2 added
16:23:42.557 INFO [LoadBalancer:New I/O server worker #2-2] Enter processArp method!!!
16:23:43.559 INFO [LoadBalancer:New I/O server worker #2-2] Enter processArp method!!!
```

Run mininet with single graph (3 hosts, 1 switch)

Note: Similar test methods and results for (linear,n), (tree,n), (assign1), (triangle), (mesh,n), (someloops).

```
mininet@mininet-VirtualBox:~/openflow$ sudo ./run_mininet.py single,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** ARPing from host h1
*** Starting SimpleHTTPServer on host h1
*** ARPing from host h2
*** Starting SimpleHTTPServer on host h2
*** ARPing from host h3
*** Starting SimpleHTTPServer on host h3
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Conclusion

Through correct implementation for Shortest-Path Switching Routing Application and Distributed Load Balance Routing Application, various host in mininet can communicate with each other via switches, and new TCP connections be redirected to different hosts in a round-robin style.

Reference

[1] D. Erickson. The Beacon OpenFlow controller. In Proc. HotSDN, Aug. 2013.
[2] Floodlight OpenFlow Controller. http://floodlight.openflowhub.org/.
[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an operating system for networks. ACM SIGCOMM Computer Communication Review, 38(3):105–110, July 2008.
[4] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In OSDI, volume 10, pages 1–6, 2010.
[5] ON.Lab. ONOS: Open network operating system, 2013. http://tinyurl.com/pjs9eyw.
[6] POX. http://www.noxrepo.org/pox/about-pox/.
[7] A. Voellmy and P. Hudak. Nettle: Functional reactive programming of OpenFlow networks. In Proc. Workshop on Practical Aspects of Declarative Languages, pages 235–249, Jan. 2011.
[8] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In ACM SIGCOMM '07, 2007.
[9] A. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic access control in enterprise networks. In Proc. Workshop: Research on Enterprise Networking, Barcelona, Spain, Aug. 2009.
[10] N. Handigol, M. Flajslik, S. Seetharaman, N. McKeown, and R. Johari. Aster*x: Load-balancing as a network primitive. In ACLD '10: Architectural Concerns in Large Datacenters, 2010
[11] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In Hot-ICE, Mar. 2011.
[12] Nicira. It's time to virtualize the network, 2012. http://nicira.com/en/ network-virtualization-platform.
[13] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In Proc. 9th USENIX OSDI, Vancouver, Canada, Oct. 2010.
[14] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving energy in data center networks. Apr. 2010.
[15] D. Erickson et al. A demonstration of virtual machine mobility in an OpenFlow network, Aug. 2008. Demo at ACM SIGCOMM.
[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, and A. Vahdat. B4: Experience with a globally deployed software defined WAN. In ACM SIGCOMM, Aug. 2013.
[17] M. Caesar, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In Proc. 2nd USENIX NSDI, Boston, MA, May 2005.
[18] Open Networking Foundation. https://www.opennetworking.org/.
[19] Open Daylight. http://www.opendaylight.org/
[20] B. Anwer, M. Motiwala, M. bin Tariq, and N. Feamster. SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. In Proc. ACM SIGCOMM, New Delhi, India, Aug. 2010.

[21] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting parallelism to scale software routers. In Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP), Big Sky, MT, Oct. 2009.
[22] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. In Proc. ACM SIGCOMM, New Delhi, India, Aug. 2010.
[23] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. ACM Transactions on Computer Systems, 18(3):263–297, Aug. 2000.
[24] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA: An open platform for gigabit-rate network switching and routing. In IEEE International Conference on Microelectronic Systems Education, pages 160–161, 2007.
[25] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In Proc. SOSP, Dec. 2001.
[26] D. E. Taylor, J. S. Turner, J. W. Lockwood, and E. L. Horta. Dynamic hardware plugins: Exploiting reconfigurable hardware for high-performance programmable routers. Computer Networks, 38(3):295–310, 2002.