**Implementation**:

To implement this project following steps are required:

1) Install the Oracle VirtualBox to local machine preferably running Windows. Using following link to download the virtual box.

* https://www.virtualbox.org/

2) Import the image of VM Mininet into the Virtual box using following instructions.

* Double-click on the Mininet\_VM.ova file → it should launch Oracle VM VirtualBox Manager.
* Should get prompted to import the virtual appliance, and so click “Import”.

3) Update the Virtual box Network setting to accept Bridge Adapter to enable SSH. Use following instructions to enable networking on Mininet.

* Launch the Mininet VM in the Oracle VirtualBox.
* On the left of the Ubuntu screen, there are a series of buttons. Click on button “Search your computer and online sources” (this is the top button).
* In the main window, a list of applications will be displayed. Click on “Terminal”.
* Once a Terminal window launches, at the command prompt, type “*ifconfig*” and click “Enter”
* Open a command prompt in the host computer. Try to ssh to the IP address that was obtained running ifconfig in the VM. Below, trying the “SSH” command results in a “Connection timed out” error. Doing a simple “ping” also results in a “Request timed out” error. If you see this, then you will need to configure the Oracle VirtualBox to allow connections to it. To connect to the Mininet VM, you will likely need to configure the Oracle VirtualBox to enable SSH connections. For the steps below, make sure that the Mininet VM is powered down.
* Click Settings in the Oracle VirtualBox.
* Click on “Network” in the left frame, and under the “Adapter 1” tab, select “Bridged Adapter” for the “Enabled Network Adapter”:
* Click on the “Adapter 2” tab and select “Host-only Adapter” for “Enable Network Adapter”.
* Go back to the Oracle VirtualBox main window and click File → Preferences.
* Click on “Network” from the left frame.
* In the previous step, there is no “Nat Network” listed in the main frame. To add a Nat Network, click on the green ‘+’ button to the upper right of the main frame. A Nat Network will appear and will be configured (confirm by double-clicking on it.
* From File, select “Host Network Manager”.
* Confirm the Host-Only is configured with your local host machine IP address by double-clicking on “VirtualBox Host-Only Ethernet Adapter” to view the configurations.

4) Other required software can be downloaded from below link.

http://www.nyu.edu/classes/jcf/CSCI-GA.2262-001/demos/Mininet\_VM.ova

5) Login into the VM (Username: mininet, Pwd: mininet). Launch the terminal and using *ifconfig* command find the IP address of the VM Mininet.

6) Launch terminal on Windows. Using SSH, VM IP and credentials in 5th step login into the Mininet.

7) Go to the project directory. Compile the Flood Light and SDN application using *ant* command.

If build status is successful, following jar file will be produced after compilation.

*FloodlightWithApps.jar*

8) Launch the SDN and Shortest path Applications using following commands.

*java —jar FloodlightWithApps.jar -cf l3routing.prop*

*java —jar FloodlightWithApps.jar -cf shortestPathSwitching.prop*

9) Start the Mininet using following command.

*sudo ./run\_mininet.py <topology>,<number of switches>*

Example*: sudo ./run\_mininet.py tree,3*

10) Run the following commands to test the network.

*pingall*

*h1 ping h3*

11) To run the Load balancer application, use the following command.

*FloodlightWithApps.jar -cf loadbalancer.prop*

12) Using below command incoming messages on a host can be traced.

*tcpdump -v -n -I hN-eth0*

*(N – Host Number)*

**Coding**:

Following pieces of code completes the project. They have been placed in TODO’s of the *Loadbalancer.java and ShortestPathSwitching.java* files. The codes are references from the Github repository- <https://github.com/zqf0722/Software-Defined-Networking-Application>.

Loadbalancer.java

public void createswitch\_rules(IOFSwitch sch, String mark){

for(long ct:this.instances.keySet()){

OFMatch om = new OFMatch();

ArrayList<OFMatchField> field\_list = new ArrayList<OFMatchField>();

OFMatchField etype;

OFMatchField match;

if(mark.equals("arp")){

log.info("ARP switch rules added");

etype = new OFMatchField(OFOXMFieldType.ETH\_TYPE, Ethernet.TYPE\_ARP);

match = new OFMatchField(OFOXMFieldType.ARP\_TPA, ct);

}

else if(mark.equals("ip")){

log.info("IP switch rules added");

etype = new OFMatchField(OFOXMFieldType.ETH\_TYPE, Ethernet.TYPE\_IPv4);

match = new OFMatchField(OFOXMFieldType.IPV4\_DST, ct);

}

else{

log.info("Wrong packet type");

return;

}

field\_list.add(etype);

field\_list.add(match);

om.setMatchFields(field\_list);

OFActionOutput output\_of = new OFActionOutput();

output\_of.setPort(OFPort.OFPP\_CONTROLLER);

ArrayList<OFAction> action\_list = new ArrayList<OFAction>();

action\_list.add(output\_of);

OFInstructionApplyActions actions = new OFInstructionApplyActions(action\_list);

ArrayList<OFInstruction> instruction\_lists = new ArrayList<OFInstruction>();

instruction\_lists.add(actions);

SwitchCommands.installRule(sch, this.table, SwitchCommands.DEFAULT\_PRIORITY,

om, instruction\_lists);

}

}

public void createmore(IOFSwitch sch){

log.info("Other switch rules added.");

OFMatch om = new OFMatch();

OFInstructionGotoTable instruct\_tab = new OFInstructionGotoTable();

instruct\_tab.setTableId(ShortestPathSwitching.table);

ArrayList<OFInstruction> instruction\_list = new ArrayList<OFInstruction>();

instruction\_list.add(instruct\_tab);

SwitchCommands.installRule(sch, this.table, (short)(SwitchCommands.DEFAULT\_PRIORITY-1), om, instruction\_list);

}

public void replyArp(Ethernet eth\_packet, OFPacketIn packet\_in, IOFSwitch sch){

ARP arp\_packt = (ARP) eth\_packet.getPayload();

long ip\_virtual = IPv4.toIPv4Address(arp\_packt.getTargetProtocolAddress());

boolean mark = false;

for(int validip:instances.keySet()){

if(validip==ip\_virtual){

mark = true;

break;

}

}

if(mark&&(arp\_packt.getOpCode() == ARP.OP\_REQUEST)) {

byte[] vh\_mac = instances.get(ip\_virtual).getVirtualMAC();

log.info("Arp reply process!");

arp\_packt.setOpCode(ARP.OP\_REPLY);

arp\_packt.setTargetHardwareAddress(arp\_packt.getSenderHardwareAddress());

arp\_packt.setTargetProtocolAddress(arp\_packt.getSenderProtocolAddress());

arp\_packt.setSenderHardwareAddress(vh\_mac);

arp\_packt.setSenderProtocolAddress(ip\_virtual);

eth\_packet.setDestinationMACAddress(eth\_packet.getSourceMACAddress());

eth\_packet.setSourceMACAddress(vh\_mac);

log.info("Arp reply packet sending: "+arp\_packt.toString());

SwitchCommands.sendPacket(sch, (short) packet\_in.getInPort(), eth\_packet);

}

}

public void reWrite(Ethernet ethrnt\_packt, OFPacketIn packt\_in, IOFSwitch sch){

IPv4 ip\_packt = (IPv4) ethrnt\_packt.getPayload();

if(ip\_packt.getProtocol() != IPv4.PROTOCOL\_TCP) return;

TCP tcp\_packt = (TCP) ip\_packt.getPayload();

long ip\_virtual = ip\_packt.getDestinationAddress();

if(tcp\_packt.getFlags() == TCP\_FLAG\_SYN){

log.info("TCP SYNs rewriting.");

long ip\_src = ip\_packt.getSourceAddress();

boolean mark = false;

for(int ip\_valid:instances.keySet()){

if(ip\_valid==ip\_virtual){

mark = true;

break;

}

}

if(!mark) return;

long port\_src = tcp\_packt.getSourcePort();

long port\_dst = tcp\_packt.getDestinationPort();

long ip\_host = instances.get(ip\_virtual).getNextip\_host();

byte[] mac\_host = getmac\_hostAddress(ip\_host);

log.info(String.format("Rewriting the IP address to %s, rewriting the MAC address to %s"

, IPv4.fromIPv4Address(ip\_host), MACAddress.valueOf(mac\_host).toString()));

OFMatch om;

ArrayList<OFMatchField> field\_list;

ArrayList<OFAction> action\_list;

ArrayList<OFInstruction> instruction\_list;

for(long j=0; j<2; j++){

field\_list = new ArrayList<OFMatchField>();

field\_list.add(new OFMatchField(OFOXMFieldType.ETH\_TYPE, Ethernet.TYPE\_IPv4));

field\_list.add(new OFMatchField(OFOXMFieldType.IPV4\_SRC, ip\_src));

field\_list.add(new OFMatchField(OFOXMFieldType.IPV4\_DST, ip\_virtual));

field\_list.add(new OFMatchField(OFOXMFieldType.IP\_PROTO, IPv4.PROTOCOL\_TCP));

if(j == 1){

field\_list.add(new OFMatchField(OFOXMFieldType.TCP\_SRC, port\_src));

field\_list.add(new OFMatchField(OFOXMFieldType.TCP\_DST, port\_dst));

action\_list = new ArrayList<OFAction>();

action\_list.add(new OFActionSetField(OFOXMFieldType.ETH\_DST, mac\_host));

action\_list.add(new OFActionSetField(OFOXMFieldType.IPV4\_DST, ip\_host));

}

else{

field\_list.add(new OFMatchField(OFOXMFieldType.TCP\_SRC, port\_dst));

field\_list.add(new OFMatchField(OFOXMFieldType.TCP\_DST, port\_src));

action\_list = new ArrayList<OFAction>();

action\_list.add(new OFActionSetField(OFOXMFieldType.ETH\_SRC, instances.get(ip\_virtual).getVirtualMAC()));

action\_list.add(new OFActionSetField(OFOXMFieldType.IPV4\_SRC, ip\_virtual));

}

om = new OFMatch();

om.setMatchFields(field\_list);

OFInstructionApplyActions acts = new OFInstructionApplyActions(action\_list);

OFInstructionGotoTable instrct\_got\_tab = new OFInstructionGotoTable();

instrct\_got\_tab.setTableId(ShortestPathSwitching.table);

instruction\_list = new ArrayList<OFInstruction>();

instruction\_list.add(acts);

instruction\_list.add(instrct\_got\_tab);

SwitchCommands.installRule(sch, this.table, (short) (SwitchCommands.DEFAULT\_PRIORITY + 1),

om, instruction\_list, SwitchCommands.NO\_TIMEOUT, (short)IDLE\_TIMEOUT);

}

}

else{

log.info("TCP reset rewriting.");

tcp\_packt.setSourcePort(tcp\_packt.getDestinationPort());

tcp\_packt.setDestinationPort(tcp\_packt.getSourcePort());

final byte tcprst\_flag = 0x04;

tcp\_packt.setFlags((short) tcprst\_flag);

tcp\_packt.setSequence(tcp\_packt.getAcknowledge() );

tcp\_packt.setWindowSize((short) 0);

tcp\_packt.setChecksum((short) 0);

tcp\_packt.serialize();

ip\_packt.setPayload(tcp\_packt);

long destIp = ip\_packt.getSourceAddress();

long ip\_src = ip\_packt.getDestinationAddress();

ip\_packt.setDestinationAddress(destIp);

ip\_packt.setSourceAddress(ip\_src);

ip\_packt.setChecksum((short) 0);

ip\_packt.serialize();

ethrnt\_packt.setPayload(ip\_packt);

byte[] destMac = ethrnt\_packt.getSourceMACAddress();

byte[] srcMac = ethrnt\_packt.getDestinationMACAddress();

ethrnt\_packt.setDestinationMACAddress(destMac);

ethrnt\_packt.setSourceMACAddress(srcMac);

SwitchCommands.sendPacket(sch, (short) packt\_in.getInPort(), ethrnt\_packt);

}

}

ShortestPathSwitching.java:

public HashMap<IOFSwitch, HashMap<IOFSwitch, IOFSwitch>> bf\_shortest\_path() {

Collection<IOFSwitch> swtchs = getSwitches().values();

HashMap<IOFSwitch, HashMap<IOFSwitch, IOFSwitch>> shortest\_path = new HashMap<IOFSwitch, HashMap<IOFSwitch, IOFSwitch>>();

for(IOFSwitch x : swtchs) {

HashMap<IOFSwitch, Integer> hm = new HashMap<IOFSwitch, Integer>();

HashMap<IOFSwitch, IOFSwitch> prdcsr = new HashMap<IOFSwitch, IOFSwitch>();

for(IOFSwitch z : swtchs) {

hm.put(z, Integer.MAX\_VALUE - 1);

prdcsr.put(z, null);

}

hm.put(x, 0);

for(int i=0;i<swtchs.size()-1;i++){

for(Link link : getLinks()) {

IOFSwitch src = getSwitches().get(link.getSrc());

IOFSwitch dst = getSwitches().get(link.getDst());

if(hm.get(src) +1 < hm.get(dst)) {

hm.put(dst, hm.get(src)+1);

prdcsr.put(dst, src);

}else if(hm.get(dst) +1 < hm.get(src)){

hm.put(src, hm.get(dst)+1);

prdcsr.put(src, dst);

}

}

}

shortest\_path.put(x, prdcsr);

}

return shortest\_path;

}

private void log\_data() {

StringBuilder msg = new StringBuilder();

msg.append("\n##################### LOG DATA #######################################");

msg.append(get\_sp\_string(this.paths));

log.info(msg.toString());

}

private String get\_sp\_string(HashMap<IOFSwitch, HashMap<IOFSwitch, IOFSwitch>> shortest\_path) {

StringBuilder msg = new StringBuilder();

msg.append("\n#############ShortestPaths#############\n");

for (Map.Entry<IOFSwitch, HashMap<IOFSwitch, IOFSwitch>> inr : shortest\_path.entrySet()) {

Iterator<Map.Entry<IOFSwitch, IOFSwitch>> itr2 = inr.getValue().entrySet().iterator();

msg.append(inr.getKey().getStringId()).append(": {");

msg.append("\n");

while (itr2.hasNext()) {

Map.Entry<IOFSwitch, IOFSwitch> inr2 = itr2.next();

msg.append("{ ");

if (inr2.getKey() != null && inr2.getKey().getStringId() != null) {

msg.append(inr2.getKey().getStringId());

} else {

msg.append("null");

}

msg.append(" : ");

if (inr2.getValue() != null) {

msg.append(inr2.getValue().getStringId());

} else {

msg.append("null");

}

msg.append(", ");

msg.append("}\n");

}

msg.append("}\n");

}

return msg.toString();

}

public void set\_flow(Host hst) {

if(hst.isAttachedToSwitch()) {

IOFSwitch s\_hst = hst.getSwitch();

OFMatch om = new OFMatch();

ArrayList<OFMatchField> listofField = new ArrayList<OFMatchField>();

OFMatchField etype = new OFMatchField(OFOXMFieldType.ETH\_TYPE, Ethernet.TYPE\_IPv4);

OFMatchField mac = new OFMatchField(OFOXMFieldType.ETH\_DST, Ethernet.toByteArray(hst.getMACAddress()));

listofField.add(etype);

listofField.add(mac);

om.setMatchFields(listofField);

for(IOFSwitch shc : getSwitches().values()) {

OFActionOutput out\_ofa = new OFActionOutput();

if(shc.getId() == s\_hst.getId()) {

out\_ofa.setPort(hst.getPort());

} else {

if(this.paths.containsKey(s\_hst)&&this.paths.get(s\_hst).containsKey(s)) {

IOFSwitch prd = this.paths.get(s\_hst).get(s);

for (Link lnk : getLinks()) {

if ((prd.getId() == lnk.getDst()) && (shc.getId() == lnk.getSrc())) {

out\_ofa.setPort(lnk.getSrcPort());

}

}

}

}

ArrayList<OFAction> action\_list = new ArrayList<OFAction>();

ArrayList<OFInstruction> instruction\_list = new ArrayList<OFInstruction>();

action\_list.add(out\_ofa);

instruction\_list.add(new OFInstructionApplyActions(action\_list));

SwitchCommands.installRule(s, table, SwitchCommands.DEFAULT\_PRIORITY, om, instruction\_list,

SwitchCommands.NO\_TIMEOUT, SwitchCommands.NO\_TIMEOUT);

}

}

}

public void create\_flow\_tabs() {

for(Host hst : getHosts()) {

set\_flow(hst);

}

}

public void del\_flow(Host hst) {

OFMatch om = new OFMatch();

ArrayList<OFMatchField> field\_list = new ArrayList<OFMatchField>();

OFMatchField etype = new OFMatchField(OFOXMFieldType.ETH\_TYPE, Ethernet.TYPE\_IPv4);

OFMatchField dst\_m = new OFMatchField(OFOXMFieldType.ETH\_DST, Ethernet.toByteArray(hst.getMACAddress()));

OFMatchField src\_m = new OFMatchField(OFOXMFieldType.ETH\_SRC, Ethernet.toByteArray(hst.getMACAddress()));

field\_list.add(etype);

field\_list.add(dst\_m);

field\_list.add(src\_m);

om.setMatchFields(field\_list);

for(IOFSwitch shc : getSwitches().values()) {

SwitchCommands.removeRules(shc, table, om);

}

}

public void del\_flow\_tabs() {

for(Host hst : getHosts()) {

del\_flow(hst);

}

}