CSC/ECE 573 (002) – Internet Protocols
Final Report

Implementation of Firewall and Server Load balancing in Software Defined Network
Fall 2016

Submission date : 12/02/2016

Project Team

Muppalla Yaswanth Kumar	ymuppal@ncsu.edu
Dhananjay Sathe	dssathe@ncsu.edu
Ami Sanghavi	asangha@ncsu.edu
Darshit Pandit	dupandit@ncsu.edu

Introduction :

The objective of this project is to Implement load balancing in Software Defined Network using OpenFlow protocol to increase the performance significantly by reducing load on the server and to implement firewall which will block the direct communication between servers and the client.

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers. Modern high-traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost-effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers, which is economically not feasible.Software Defined Networking (SDN) offers a cost-effective and flexible approach in implementing a load balancer. Software-Defined Network (SDN) is seen as one of the most promising paradigm. By using this technique the network becomes directly programmable and agile. Moving away from traditional hardware-based networking approach, this project implements load balancing with the help of openflow. The new SDN approach reduces the cost, offers flexibility in configuration, reduces time to deploy, provides automation and facilitates building a network without requiring the knowledge of any vendor-specific software/hardware.

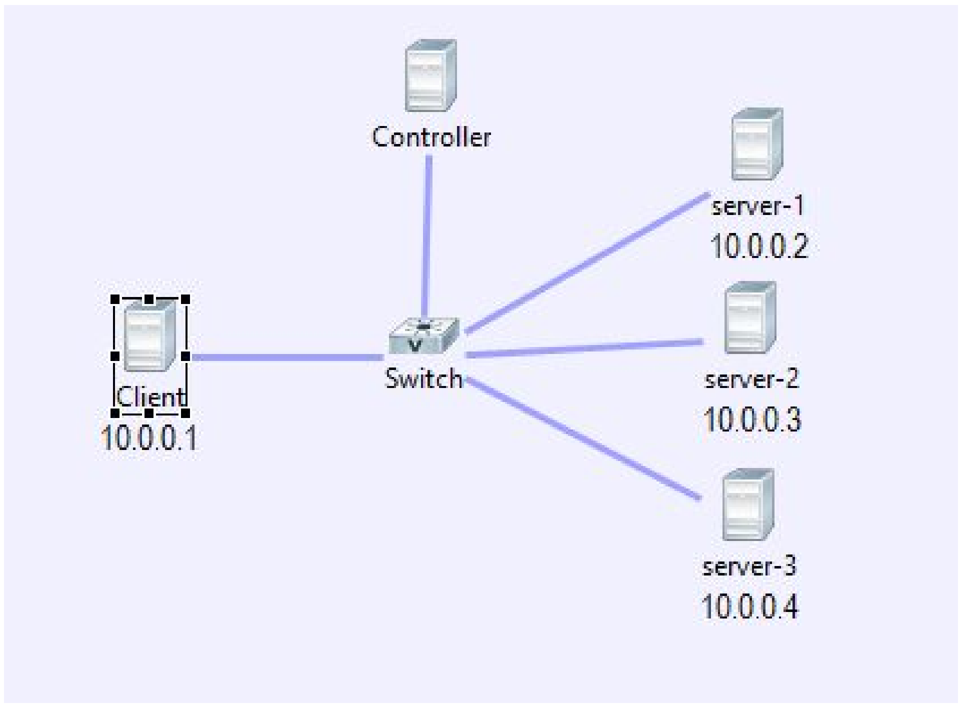


Fig a:Topology that we are implementing in-order to perform load balancing and firewall services.

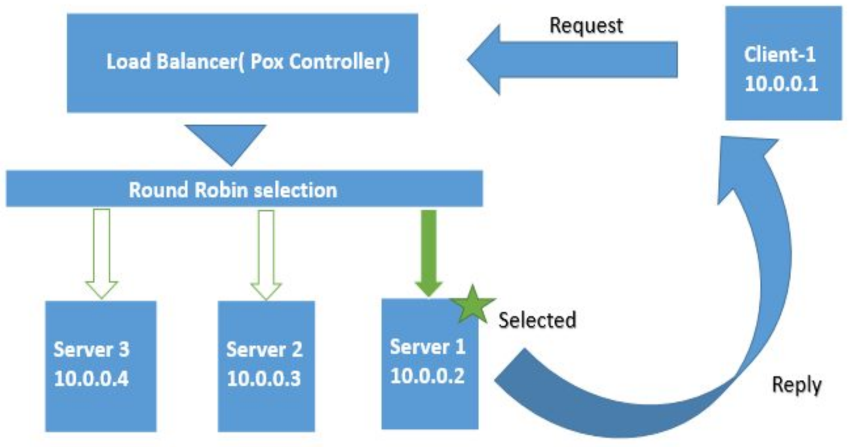


Fig b : an overview of our system topology how servers will be selected in round robin

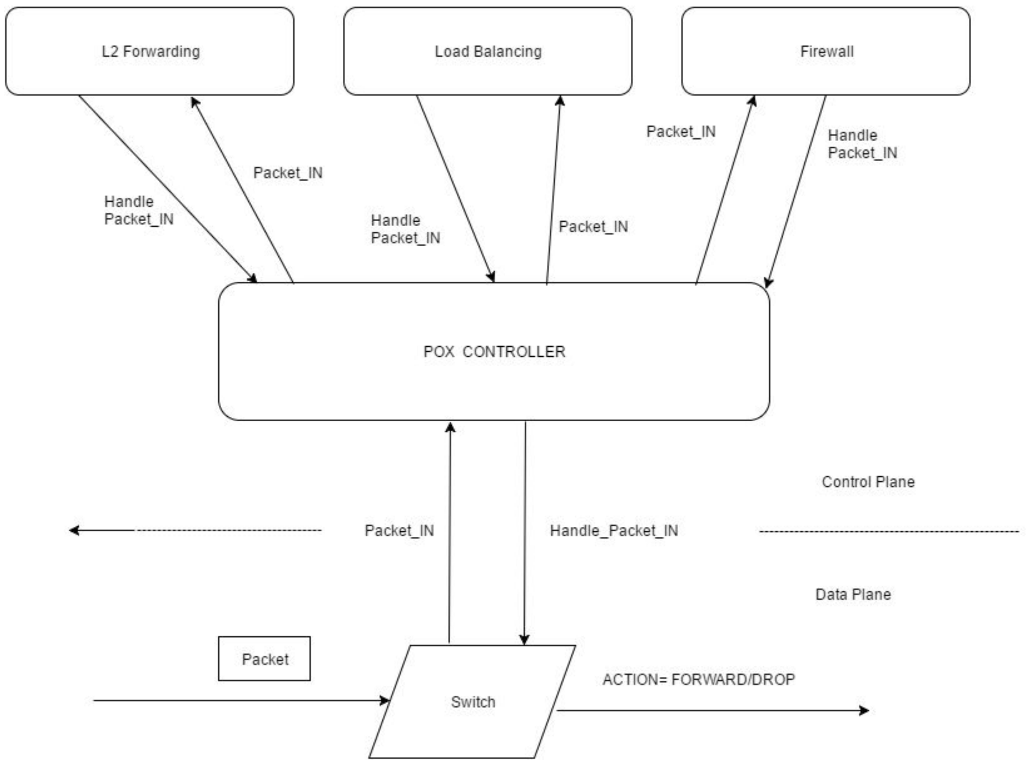


Fig c :Switch controlled by the POX controller Explanation in next page

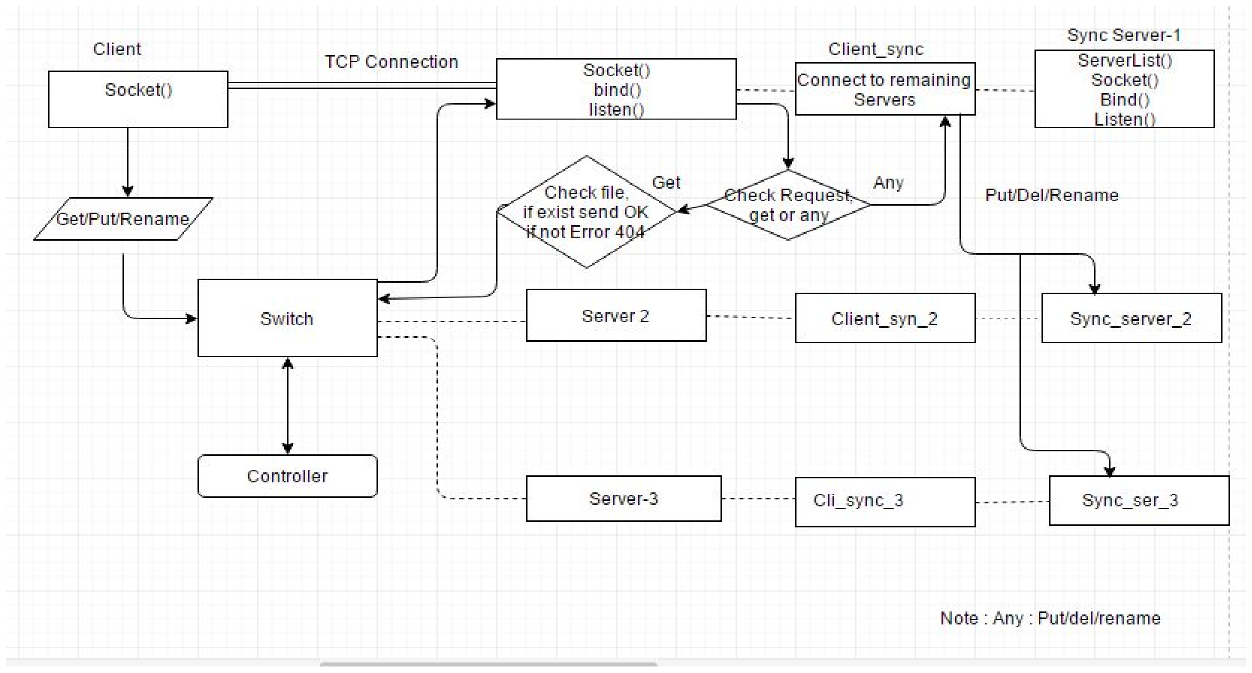
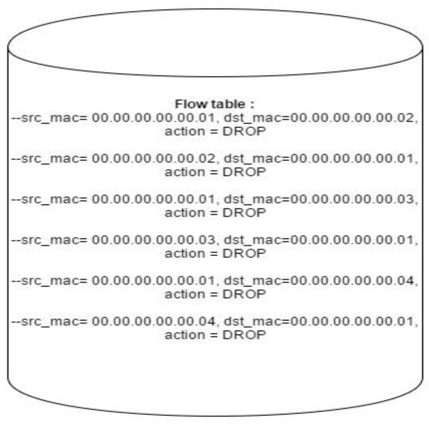


Fig d : Overview of How the Client request served and the file is synchronized between the servers.


Flow table :
--src_mac= 00.00.00.00.00.01, dst_mac=00.00.00.00.00.02, action = DROP
--src_mac= 00.00.00.00.00.02, dst_mac=00.00.00.00.00.01, action = DROP
--src_mac= 00.00.00.00.00.01, dst_mac=00.00.00.00.00.03, action = DROP
--src_mac= 00.00.00.00.00.03, dst_mac=00.00.00.00.00.01, action = DROP
--src_mac= 00.00.00.00.00.01, dst_mac=00.00.00.00.00.04, action = DROP
--src_mac= 00.00.00.00.00.04, dst_mac=00.00.00.00.00.01, action = DROP

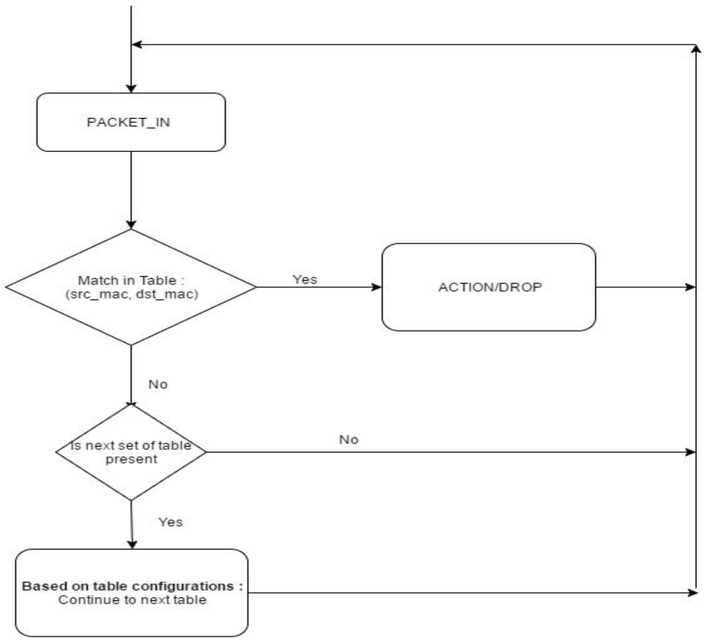

PACKET_IN
Match in Table : (src_mac, dst_mac)
Yes
ACTION/DROP
No
Is next set of table present
No
Yes
Based on table configurations : Continue to next table

Fig e : Firewall algorithm

Algorithm explanation

Fig C :

This figure shows the generic datapath for a packet.

1) A packet arrives at a switch with destination IP address as virtual IP address.
2) The Pox controller inspect the packet.
3) The Pox controller implements three different modules viz. L2 forwarding, Load balancing and firewall.
4) Accordingly flow tables are inserted in the switch which determine the output port

Fig D :

Here there are three VMS/actual servers. Each will have server process running in them requesting for connection from user. Suppose user wants to delete file from a server, he makes request to virtual ip and by round robin fashion will get served by some server say server2. This deletion needs to be synced across all servers. (All servers should have same content).
Similarly this goes for put and rename requests. So there are additional three sync server processes running and three sync client processes running in it.

Algorithm steps

1) Sync servers are running on every server.
2) Whenever client makes a put/delete/rename request according to controller logic (load balancer) gets served by one of the three server
3) The server performs the request operation (delete/put/rename)
4) It sees that this request needs to be synced across all other servers
5) Therefore it invokes sync client which will make request to other servers (Provided in serverlist.csv file)
6) All other syncserver request accept the connection and eventually all servers have same content
7) If the request is get or list file , the request is confined to one server only.
8) The file which is to be synced utilises the server-switch-server interface. Hence client link is never utilised which is very efficient.

Implementation

Programming language : Python,
Tools : Mininet,POX

- **Client Server Code Implementation**

Our code is enough to demonstrate the round robin functionality of the network.
Instead of making complex requests such as
GET ../ ADD../ etc we have a developed a very interactive UI which will ask the user what action needs to be performed [Retrieve file,Put file,List all files,Rename file, Delete file]
This is because we are concentrating on the load balancing aspect of SDN and not on the actual FTP protocol details
Two status codes and corresponding phrases are defined:
• 200 OK
• 404 Not Found
For each of the cases (Except Put and List All files), there is manipulation of files at the server side.
If the client requests for file not present in the server
404 Not Found will be displayed to the client.
If the file was successfully retrieved, 200 will be returned to the client.

Working:
So initially all servers will have same set of RFCs.
Client 1 requests RFC abc, Request gets served by Server1
Client 1 requests RFC xyz, Request gets served by Server2
Client 2 requests RFC xyz, Request gets served by Server3
And so on

Explanation of code : (Server,Syncserver and Client)
Server : There is a server listening at predefined port. Whenever a certain request comes from client, server analyses what request it is (Get,put,Delete etc).So we have developed various functions for each request.
If the request is get,delete or rename the server first checks if the file is present or not and returns appropriate Status Codes (200/404).
If the file is present, necessary operations are performed.
If the client wants the file to be retrieved,the code will read data byte by byte and sent it to the socket buffer.Client will write this data into the file and thus file transfer will take place.
For the put request, exact opposite will take place. Here data also needs to be synced across the servers.(Explained in the sync server code section)
Clientsync : Whenever a put,delete or rename request comes, this needs to be synced with the other servers. So the server code call clientsync code to make a put,delete or rename request to the sync servers listening on other networks.

Syncserver: This is to basically sync all files across the servers. It maintains a list of all servers present in the network
When i put some file into server, by roundrobin it will be uploaded only to one server.
This should not be the case.So there is a syncserver listening for any sync request from the server who got the file and will respond accrodlingly(To put,rename and delete respectively)

Client code : Simple UI based which asks user what operation needs to be performed

- **SDN Code Implementation :**

a) **Load balancing Code Implementation :**

When a request is received on a virtual IP address of the load balancer, it accepts the connection and changes the destination IP to the IP address of one of the servers provided by the load balancing algorithm (round robin). After receiving a request from the load balancer, assigned server sends a reply with the destination IP address as the IP address of the load balancer. Then, the load balancer intercepts the return packet from the server and changes the source IP address to match the virtual IP address and sends to the host.

When packet comes to the switch, the switch will sends the packet to the Load balancer (controller), here the load balancer invoke the Round robin function, In the round robin function, it will first check whether the packet is ipv4 and has destination ip as virtual ip , if so then it will select the next server according to round robin from the list,get the respective servers ip, mac and their matching switch port .After this it will invoke function setup_route_to_server which will setup route to the selected server and connection will be established.Once packet reach the destination server, Now setup_route_from_server will send the packets with destination IP , MAC of
Load balancer, Since switch is enabled with l2 learning functionality ,it will have the table entry for virtual IP, So it will forward to the load balancer, and load balancer replace the source ip of server with its IP and destination IP as Client IP. Now switch will forward the packet to the client, and the Load balancer will close the connection.

b) Firewall Code Implementation :

For this implementation, we had multiple options for type of flows that can be added to switch to implement the functionality. Type of flows is categorized based on ‘deny' or ‘accept' rule and source and destination ‘IP or MAC' addresses. Now selecting IP addresses for filtering is not an efficient option as, if we consider client IP address in our flow, then that metric will never remain constant as customers at client side can anytime statically change ip address. So we select MAC based flow. We create a policy file with each row representing every possible communication between client and server in our topology. With two clients and four servers there are total 8 rows that are added which will be used to be implement a ‘deny' or ‘allow' rule in the firewall code. Idea of this implementation is simple, we are just simple making every possible communication blocked statically. This code may not efficient in complex networks though.

Working:

As shown in the below command, here we add firewall code in the pox controller by providing the input policy file to firewall executable code.

We implement two scenarios here

a) Initiating communication between client and server through virtual ip.That is client requesting directly to the virtual ip which is forwarded to one of the servers based on the round-robin fashion as explained above (Scenario here is if client wants to access some files from the server then it will request to a virtual ip of the server which is created on the switch.) Communication must be successful.

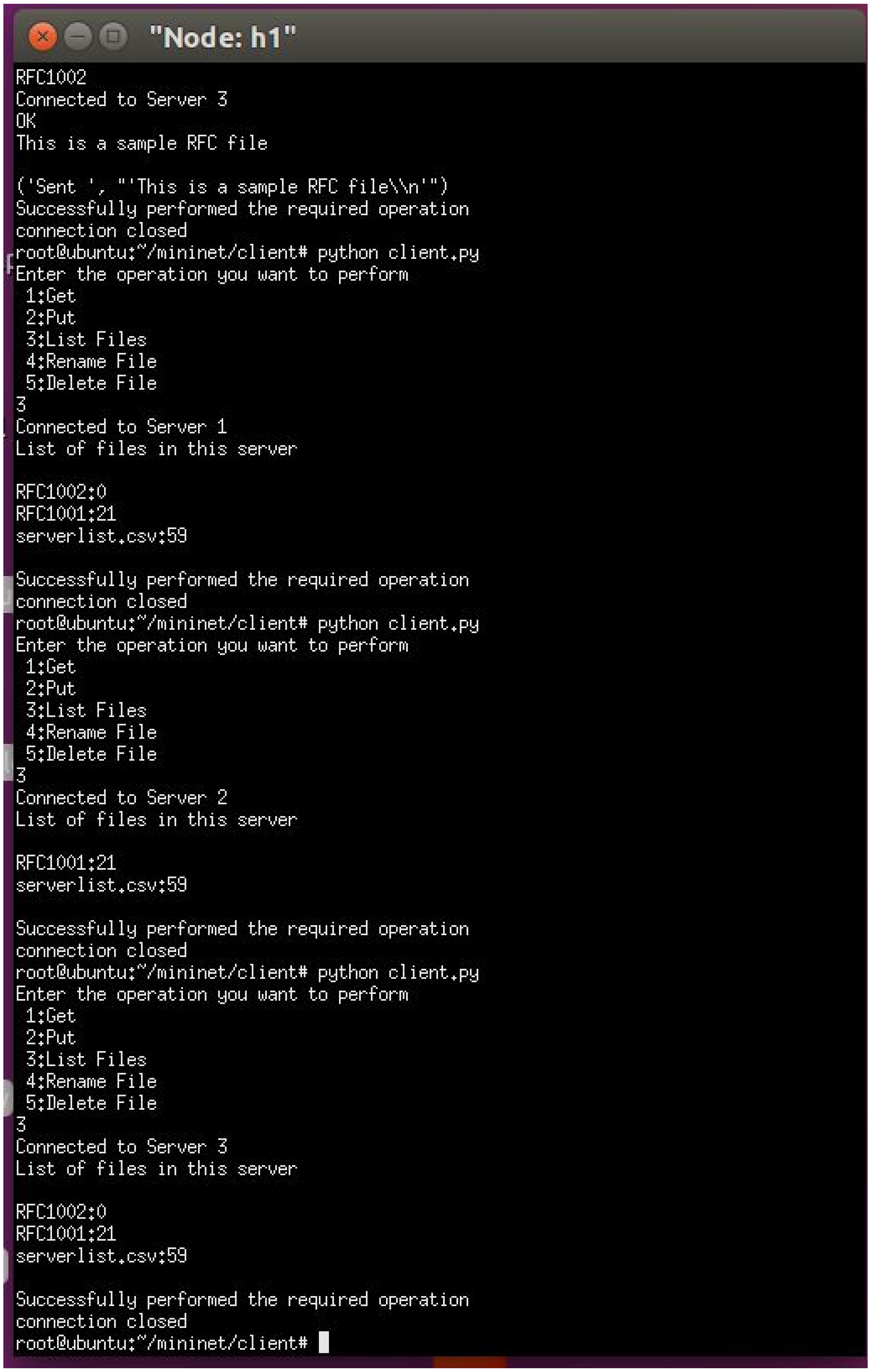
b) Now we implement the same code but making direct communication between client and server and not via virtual ip. Communication will be blocked at switch.

Now we can also verify the firewall code, by implementing (b) step first without adding firewall code to the pox controller and then by adding it.

Explanation of Firewall Code :

When created two main functions in class ‘Firewall’. In function ‘handle_ConnectionUp’ when packet comes to the switch, the switch will sends the packet to the Firewall (controller), match-action mechanism is performed based on two parameters (src_mac,dst_mac). The source and mac address of the new incoming is copied to standard set of source and destination variables used in function ofp_flow_mod(). The match-action functionality is performed here. In the second function ‘launch’ we actually pass the already written firewall policies ‘policies.csv’. This csv file contains ‘id, mac1,mac2’ in form of rows and columns. We define the global variable ‘denyList’ and for each row described in policy file, we append each possibility of connection that we want to avoid (direct client-server communication). At the end of this function rules will be added in the denyList with the action of drop.

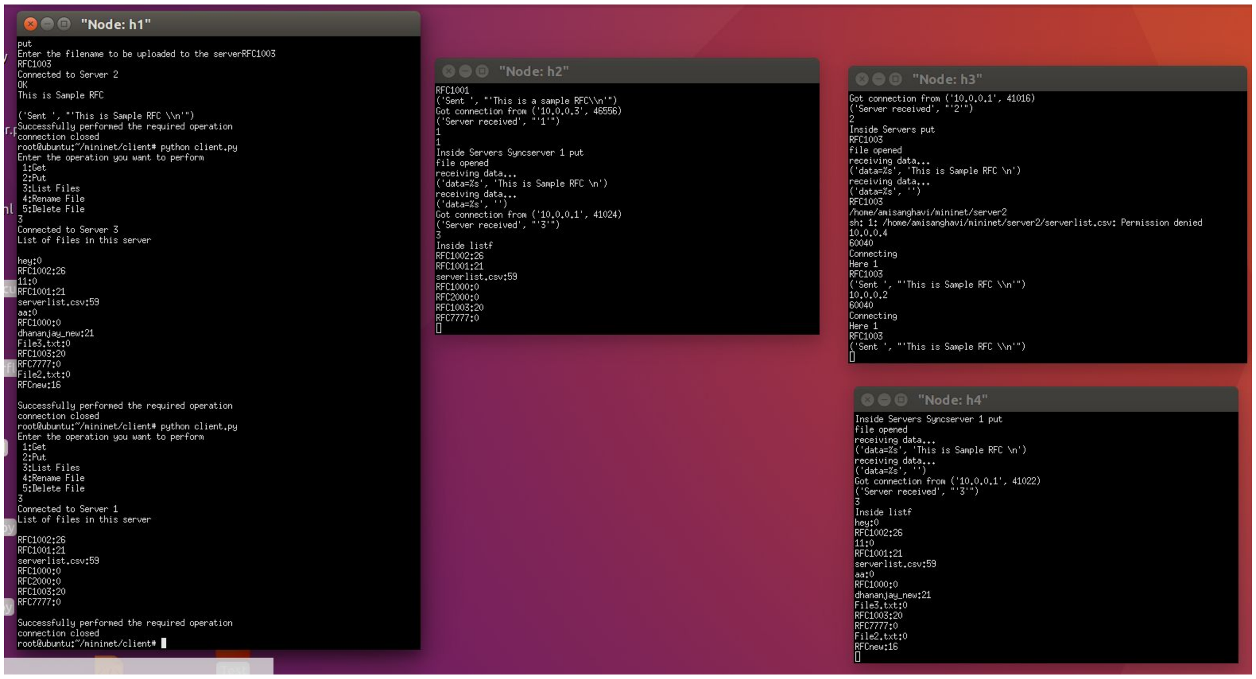
Experimental Observations



"Node: h1"

```
RFC1002
Connected to Server 3
OK
This is a sample RFC file

('Sent ', "'This is a sample RFC file\\n'")
Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client# python client.py
Enter the operation you want to perform
 1:Get
 2:Put
 3:List Files
 4:Rename File
 5:Delete File
3
Connected to Server 1
List of files in this server

RFC1002:0
RFC1001:21
serverlist.csv:59

Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client# python client.py
Enter the operation you want to perform
 1:Get
 2:Put
 3:List Files
 4:Rename File
 5:Delete File
3
Connected to Server 2
List of files in this server

RFC1001:21
serverlist.csv:59

Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client# python client.py
Enter the operation you want to perform
 1:Get
 2:Put
 3:List Files
 4:Rename File
 5:Delete File
3
Connected to Server 3
List of files in this server

RFC1002:0
RFC1001:21
serverlist.csv:59

Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client# 
```

Figure 1: List working in Round Robin Fashion . Can be seen from "Connected to"



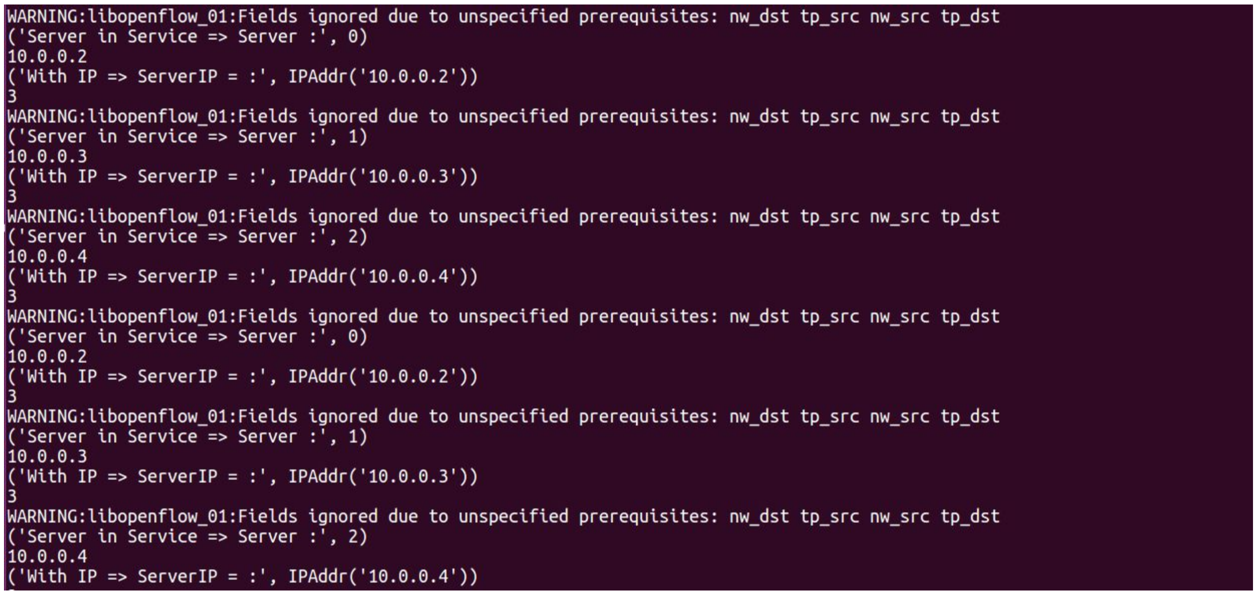
```
"Node: h1"
put
Enter the filename to be uploaded to the serverRFC1003
RFC1003
Connected to Server 2
OK
This is Sample RFC

('Sent ', "'This is Sample RFC \\n'")
Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client# python client.py
Enter the operation you want to perform
 1:Get
 2:Put
 3:List Files
 4:Rename File
 5:Delete File
3
Connected to Server 3
List of files in this server

hey:0
RFC1002:26
11:0
RFC1001:21
serverlist.csv:59
aa:0
RFC1000:0
dhananjay_new:21
File3.txt:0
RFC1003:20
RFC7777:0
File2.txt:0
RFCnew:16

Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client# python client.py
Enter the operation you want to perform
 1:Get
 2:Put
 3:List Files
 4:Rename File
 5:Delete File
3
Connected to Server 1
List of files in this server

RFC1002:26
RFC1001:21
serverlist.csv:59
RFC1000:0
RFC2000:0
RFC1003:20
RFC7777:0

Successfully performed the required operation
connection closed
root@ubuntu:~/mininet/client#
```

```
"Node: h2"
RFC1001
('Sent ', "'This is a sample RFC\\n'")
Got connection from ('10.0.0.3', 46556)
('Server received', "'1'")
1
1
Inside Servers Syncserver 1 put
file opened
receiving data...
('data=%s', 'This is Sample RFC \n')
receiving data...
('data=%s', '')
Got connection from ('10.0.0.1', 41024)
('Server received', "'3'")
3
Inside listf
RFC1002:26
RFC1001:21
serverlist.csv:59
RFC1000:0
RFC2000:0
RFC1003:20
RFC7777:0
```

```
"Node: h3"
Got connection from ('10.0.0.1', 41016)
('Server received', "'2'")
2
Inside Servers put
RFC1003
file opened
receiving data...
('data=%s', 'This is Sample RFC \n')
receiving data...
('data=%s', '')
RFC1003
/home/amisanghavi/mininet/server2
sh: 1: /home/amisanghavi/mininet/server2/serverlist.csv: Permission denied
10.0.0.4
60040
Connecting
Here 1
RFC1003
('Sent ', "'This is Sample RFC \\n'")
10.0.0.2
60040
Connecting
Here 1
RFC1003
('Sent ', "'This is Sample RFC \\n'")
```

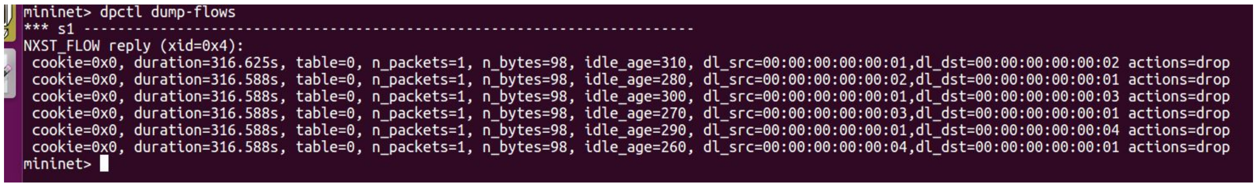
```
"Node: h4"
Inside Servers Syncserver 1 put
file opened
receiving data...
('data=%s', 'This is Sample RFC \n')
receiving data...
('data=%s', '')
Got connection from ('10.0.0.1', 41022)
('Server received', "'3'")
3
Inside listf
hey:0
RFC1002:26
11:0
RFC1001:21
serverlist.csv:59
aa:0
RFC1000:0
dhananjay_new:21
File3.txt:0
RFC1003:20
RFC7777:0
File2.txt:0
RFCnew:16
```

Figure 2 Round Robin Fashion: Put – Server 2 and Synced across all server 1 and 2 (Seen using list)



```
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: nw_dst tp_src nw_src tp_dst
('Server in Service => Server :', 0)
10.0.0.2
('With IP => ServerIP = :', IPAddr('10.0.0.2'))
3
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: nw_dst tp_src nw_src tp_dst
('Server in Service => Server :', 1)
10.0.0.3
('With IP => ServerIP = :', IPAddr('10.0.0.3'))
3
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: nw_dst tp_src nw_src tp_dst
('Server in Service => Server :', 2)
10.0.0.4
('With IP => ServerIP = :', IPAddr('10.0.0.4'))
3
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: nw_dst tp_src nw_src tp_dst
('Server in Service => Server :', 0)
10.0.0.2
('With IP => ServerIP = :', IPAddr('10.0.0.2'))
3
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: nw_dst tp_src nw_src tp_dst
('Server in Service => Server :', 1)
10.0.0.3
('With IP => ServerIP = :', IPAddr('10.0.0.3'))
3
WARNING:libopenflow_01:Fields ignored due to unspecified prerequisites: nw_dst tp_src nw_src tp_dst
('Server in Service => Server :', 2)
10.0.0.4
('With IP => ServerIP = :', IPAddr('10.0.0.4'))
```

Figure 4 POX log :6 requests serviced in round robin manner by loadbalance (10.0.0.2,10.0.0.3,10.0.0.4 … and so on)



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=316.625s, table=0, n_packets=1, n_bytes=98, idle_age=310, dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=drop
 cookie=0x0, duration=316.588s, table=0, n_packets=1, n_bytes=98, idle_age=280, dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=drop
 cookie=0x0, duration=316.588s, table=0, n_packets=1, n_bytes=98, idle_age=300, dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=drop
 cookie=0x0, duration=316.588s, table=0, n_packets=1, n_bytes=98, idle_age=270, dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=drop
 cookie=0x0, duration=316.588s, table=0, n_packets=1, n_bytes=98, idle_age=290, dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=drop
 cookie=0x0, duration=316.588s, table=0, n_packets=1, n_bytes=98, idle_age=260, dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=drop
mininet>
```

Figure 5 Direct connection to server blocked by firewall

5. **Experimental results:**

Initially all servers have RFC1000

Client/Operation	Server 1	Server2	Server3
Get file 'RFC1000'	**Served by Server1**	**-**	**-**
List all files in server	**-**	**Served by Server2**	**-**
Rename 'RFC1000' to 'RFC2000'	**'RFC1000' renamed to 'RFC2000' by Synchronization**	**'RFC1000' renamed to 'RFC2000' by Synchronization**	**Served by Server3 Synced to Server1 and Server2**
List all files in server	**Served by Server1 RFC 2000**	**-**	**-**
Delete file 'RFC 2000'	**RFC 2000 deleted by Synchronization**	**Served by Server2 Synced to Server1 and Server3**	**RFC 2000 deleted by Synchronization**
Put file 'RFC3432'	**RFC 3432 uploaded by Synchronization**	**RFC 3432 uploaded by Synchronization**	**Served by Server3 Synced to Server1 and Server2**
List all files in server	**Served by Server1 RFC 3432**	**-**	**-**

Figure 6 Experimental Results (One client and three servers)

Thus, we see that all requests are served in a round robin manner.

When a file is uploaded, deleted, or renamed by the client to the server, that server performs synchronization to the other servers in the network.

We even performed the experiment for 2 clients and 4 servers, but the screenshots were not visible due to 6 terminals working.

Observations on flow table were verified:

Here we can see that each request to virtual IP goes to controller. Controller decides the o/p port by implementing the round robin logic. Controller changes the destination IP to one of the actual server and inserts this flow into the switch. Controller will be contacted every time the request is made.

We also tested that client cannot directly connect to actual server without virtual IP due to firewall module implemented in controller.

References:

[1]http://mininet.org/walkthrough/

[2]http://www.colorado.edu/itp/sites/default/files/attached-files/70110-130943_-_harsh_kotak_-_apr_25_201[6_933_pm_-_team6_capstone_final_paper_resubmit.pdf

[3]https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch#Sending_OpenFlow_messages_with_POX

[4] http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

[5] L2_learning.py : Mininet Sample Files ,Custom Topology Implementaion : Mininet Sample File

[6] https://openflow.stanford.edu/display/ONL/POX+Wiki