University of Victoria

Department of Computer Science

CSC 586A

Software Defined Networking

Course Project

Pyretic-based Firewall and QoS Research

Zhang, Zeyang(Sylar)

Student Number: V00869738

ngjtxyh@aliyun.com/zeyangz@uvic.ca

CONTENT

ABSTRACT

Software-Defined Networking (SDN) helps organization accelerate application deployment and delivery, dramatically reducing costs through policy-enabled work-flow automation. As a result, SDN technology has became more and more popular. This project focused on one of Frenetic language - Pyretic. A Pyretic-based layer 2 firewall was built and worked in SDN. Then, in order to improve the network efficiency, Quality of Service(QoS) was also get tried through Pyretic. As a course project, this project report contains relatively details in environment setup, tool installation and application implementation.

1. Background

(1) Software Defined Networking

Due to increasing demand of Cloud computing, Mobile traffic, Big data and Internet of Things(IoT), network providers and users have to re-evaluate traditional approaches to network architecture. Luckily, network transmission technologies has also been increased, new technologies are no longer restricted by the performance aged network devices, because with progressive devices, everyone can handle more changeable design.

As a relatively new approach to computer networking, Software defined networking(SDN) allows network administrator to initialize, control, change and manage network behavior dynamically through open interfaces and abstraction of lower-level functionality programmatically.

The SDN architecture is directly programmable, agile:, centrally managed, programmatically configured and open standards-based and vendor-neutral

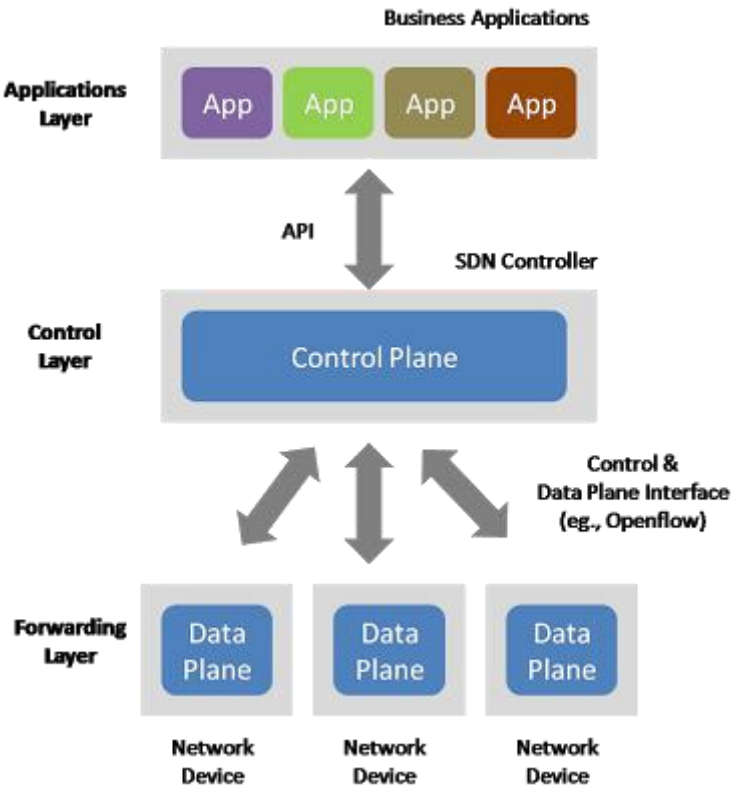

Fig 1.1 SDN architecture[1]

This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. And the OpenFlow protocol is a foundational element for building SDN solutions.[2]

(2) Frenetic

Frenetic is a Software defined networking(SDN) umbrella project focused on language design and compiler implementation. It is an open-source SDN controller platform designed to make SDN programming easy, modular, and semantically correct.[3]

The programming language of Frenetic with the following essential features[4]:
• High-level abstractions that give programmers direct control over the network, allowing them to specify what they want the network to do without worrying about how to implement it.
• Modular constructs that facilitate compositional reasoning about programs.
• Portability, allowing programs written for one platform to be reused with different devices.
• Rigorous semantic foundations that precisely document the meaning of the language and provide a solid platform for mechanical program analysis tools.

Table below compared two programming language for Frenetic. Pyretic(Python-based) and Frenetic Ocaml. Ocaml(Used to be called Objective Caml). Here, in this project, because of the course experience and current programming language popularity, Pyretic was applied.

	Pyretic	**Frenetic Ocaml**
Target Community	Systems and Networking	Programming Languages
Primary Dev. Location	Princeton	Cornell
Host Language	Python	OCaml

Table 1.1 Pyretic and Frenetic Ocaml

Pyretic encourages programmers to focus on how to specify a network policy at a high level of abstraction, rather than how to implement it using low-level OpenFlow mechanisms. In particular, instead of implementing a policy by incrementally installing physical rule after physical rule on switch after switch, a Pyretic policy is specified for the entire network at once, via a function from an input located packet (i.e., a packet and its location) to an output set of located packets. The output packets can have modified fields and usually end up at new locations—this is how packet forwarding occurs. The programmer does not need to worry about which OpenFlow rules are used to move packets from place to place.

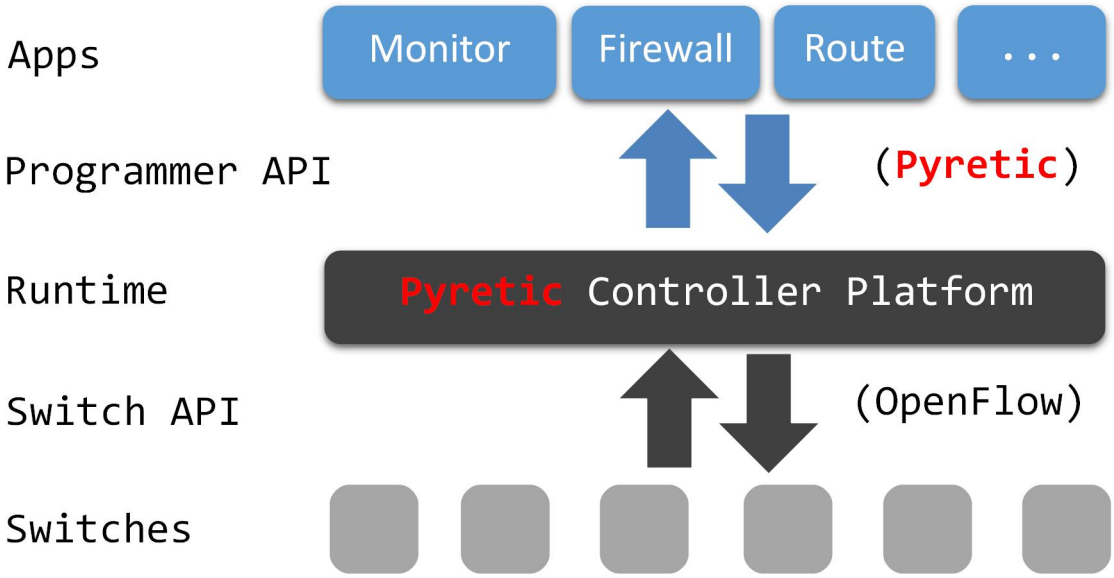

Fig. 1.2 Pyretic working in SDN architecture[5]

2. Environment Setup

(1) Pyretic installation

Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine. Pyretic-VM is a virtual machine that inherits Mininet with pre-installed POX controller. Then get latest version of Pyretic-VM(0.2.2) from http://frenetic-lang.org/pyretic/. After that, use VirtualBox to load it with an extra adapter - host-only adapter.

(2) Remote terminal - MobaXterm

The reason we use MobaXterm as remote terminal instead of using original VM terminal is that original VM doesn't support multi-window and only available for one small page information. While, MobaXterm has inner Xterm that support multi-window, and its user interface is also friendly, easy to use.

In Pyretic-VM, configure adapter's IP address and allocate IP addresses for DHCP server.

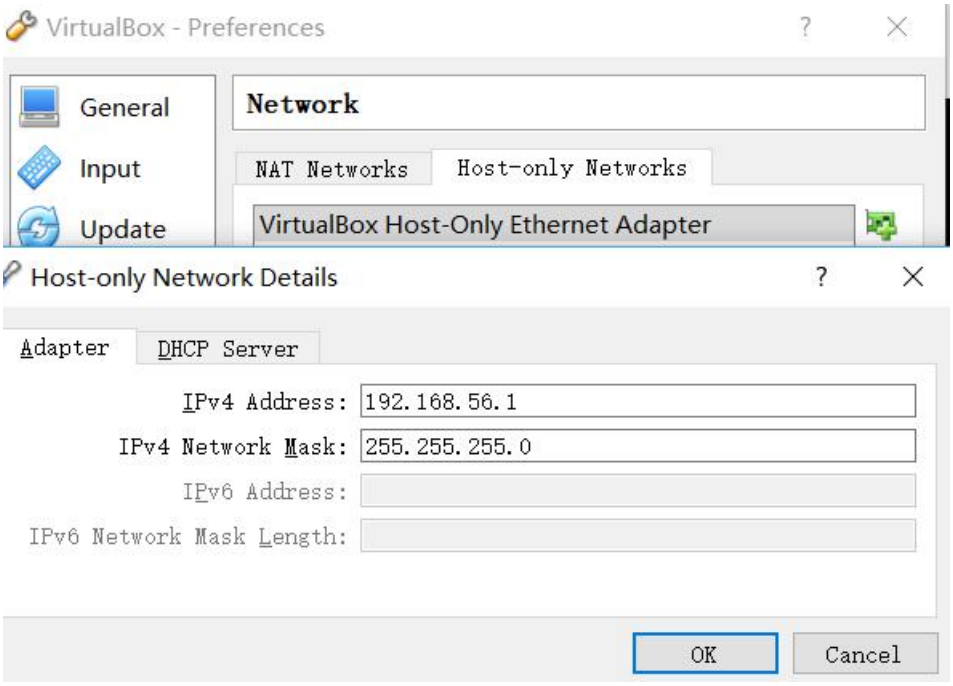

Fig. 2.1 Adapter IP

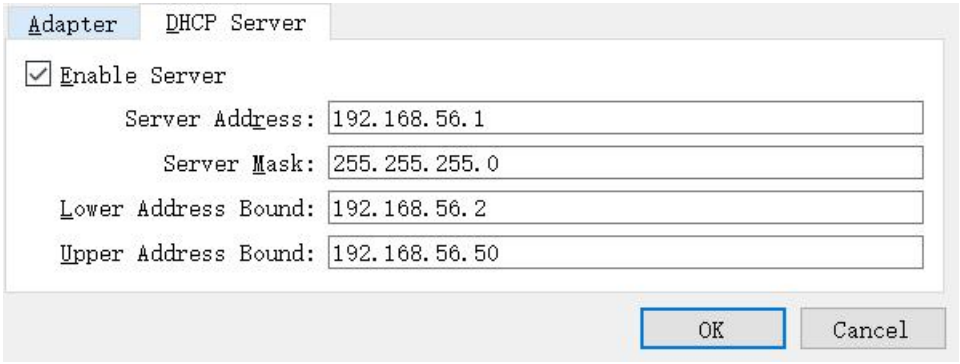

Fig. 2.2 DHCP Server

Therefore, any IP addresses between 192.168.56.2 and 192.168.56.50 can be our Pyretic-VM's IP address. However, unlike Mininet-VM, Pyretic-VM has to be set IP address manually. Use the following command to set up IP address,

```
~$ sudo ifconfig eth1 192.168.56.2
```

Then, try "ifconfig" command to see the change of IP address,

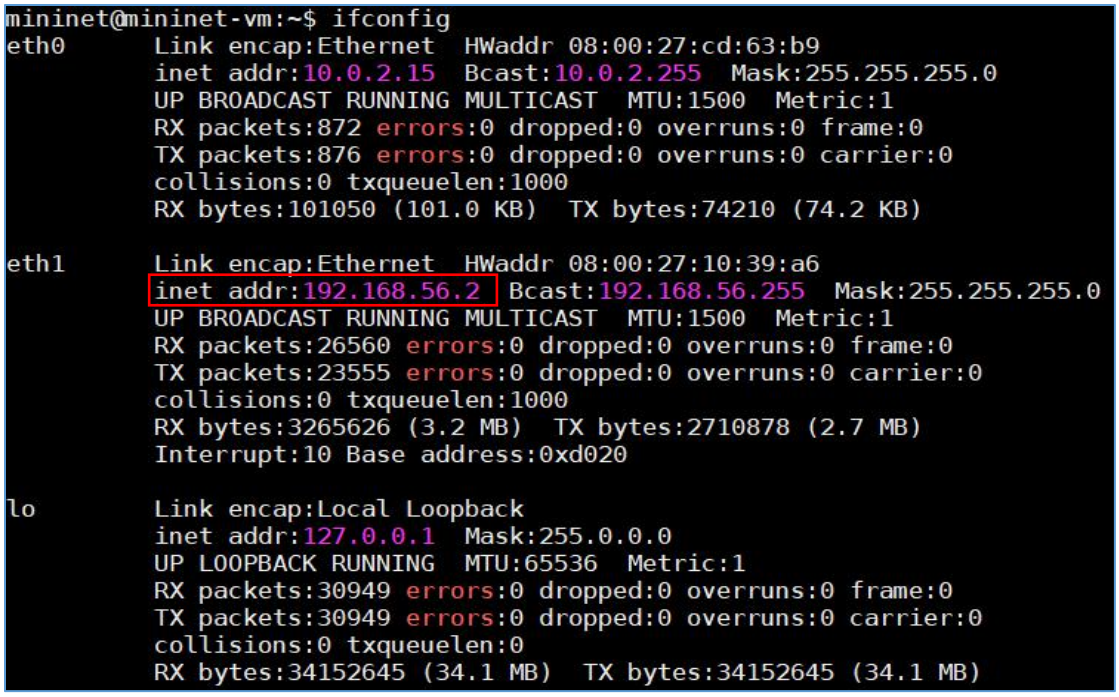

Fig. 2.3 IP address for Pyretic-VM

Download MobaXterm from http://mobaxterm.mobatek.net/download.html, and type in IP address 192.168.56.2 and choose “Ssh(Secure shell)” session, log in with user name: mininet. Now, the remote terminal has been set up.

3. Firewall Design

A Firewall is a network security system that is used to control the flow of ingress and egress traffic usually between a more secure local-area network (LAN) and a less secure wide-area network (WAN). The system analyses data packets for parameters like L2/L3 headers (i.e., MAC and IP address) or performs deep packet inspection (DPI) for higher layer parameters (like application type and services etc) to filter network traffic. A firewall acts as a barricade between a trusted, secure internal network and another network (e.g. the Internet) which is supposed to be not very secure or trusted.

In this project, a simple layer 2 firewall that runs alongside the MAC learning module on the Pyretic runtime has been applied. The firewall application is provided with a list of MAC address pairs i.e., access control list (ACLs). When a connection establishes between the controller and the switch, the application installs static flow rule entries in the OpenFlow table to disable all communication between each MAC pair.

The firewall was agnostic of the underlying topology. It took MAC pair list as input and installed it on the switches in the network. To make things simple, we will implement a less intelligent approach and will install rules on all the switches in the network.

Here, a basic simple topology was applied. Due to the code, this firewall can be also implement in any complicated topology such as data center topology.

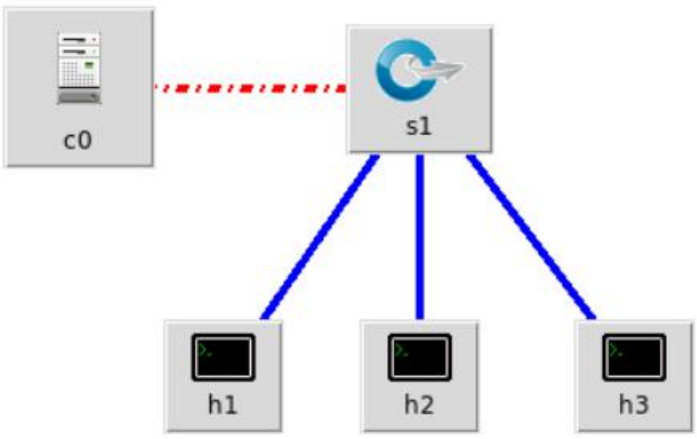

Fig. 3.4 Topology

“c0” refers to controller(POX, in this particular case), “s1” refers to OpenFlow switch, and “h1, h2, h3” means hosts 1 to 3.

4. Firewall Implementation

First, start controller with running application firewall that we programmed before by running the following command,

```
~$ pyretic.py pyretic.examples.firewall
```

And the screen shows the following information indicates that controller has successfully start up.

```
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.1.0 (betta) is up.
```

Use the second terminal to start a Mininet topology. Here, as we discussed before, a 1 switch, 3 hosts simple topology was applied.

```
~$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

And here is the information shows that network has been creating with 1controller, 3 hosts, 1 switch, and 3 links.

```
*** 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
```

Use “pingall” command to see if the firewall works. As it’s shown below, host 1 was not able to ping h2, vice versa. During test, h1 can’t ping h2 for sure due to the firewall policy, yet h2 also can’t ping h1, even the policy says blocked source address is h1 and destination is h2. This is because when h2 tries to ping h1, h2 will send a request to h1(handshake - hello), which has no problem for h1 to receive that, but h1 can’t reply and allow connection from h2. Therefore h2 will never get reply from h1 and it will have to wait until the maximum and give up then.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X h3
h3 -> h1 h2
*** Results: 33% dropped (4/6 received)
```

Meantime, since the ping was working, the controller screen shows a lot information. To be specific, here I divided “pingall” into several steps.
First, for 2 hosts were forbidden from each other, with command,

```
mininet> h1 ping -c1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

While controller terminal shows,

```
parallel:
    if
        match: ('switch', 1) ('dstmac', 00:00:00:00:00:02)
    then
        fwd 2
    else
        if
            match: ('switch', 1) ('dstmac', 00:00:00:00:00:03)
        then
            fwd 3
        else
            if
                match: ('switch', 1) ('dstmac', 00:00:00:00:00:01)
            then
                fwd 1
            else
                flood on:
                ------------------------------------------------------
                switch  |  switch edges    |   egress ports            |
                ------------------------------------------------------
                1       |                  |  1[2]---, 1[3]---, 1[1]---  |
                ------------------------------------------------------
    packets
    sequential:
        LimitFilter
        negate:
            union:
                match: ('switch', 1) ('srcmac', 00:00:00:00:00:01)
                match: ('switch', 1) ('srcmac', 00:00:00:00:00:03)
                match: ('switch', 1) ('srcmac', 00:00:00:00:00:02)
        FwdBucket
```

Fig. 4.1 Controller output information during pinging h1 and h2

To compare, try h1 and h3, which should ping successfully.

```
mininet> h1 ping -c1 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=213 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 213.252/213.252/213.252/0.000 ms
```

```
parallel:
    if
        match: ('switch', 1) ('dstmac', 00:00:00:00:00:01)
    then
        fwd 1
    else
        flood on:
        ------------------------------------------------------
        switch  |  switch edges    |   egress ports             |
        ------------------------------------------------------
        1       |                  |  1[2]---, 1[3]---, 1[1]---  |
        ------------------------------------------------------
    packets
    sequential:
        LimitFilter
        negate:
            union:
                match: ('switch', 1) ('srcmac', 00:00:00:00:00:01)
        FwdBucket
```

```
parallel:
    if
        match: ('switch', 1) ('dstmac', 00:00:00:00:00:03)
    then
        fwd 3
    else
        if
            match: ('switch', 1) ('dstmac', 00:00:00:00:00:01)
        then
            fwd 1
        else
            flood on:
            ------------------------------------------------------
            switch  |  switch edges    |   egress ports             |
            ------------------------------------------------------
            1       |                  |  1[2]---, 1[3]---, 1[1]---  |
            ------------------------------------------------------
    packets
    sequential:
        LimitFilter
        negate:
            union:
                match: ('switch', 1) ('srcmac', 00:00:00:00:00:01)
                match: ('switch', 1) ('srcmac', 00:00:00:00:00:03)
        FwdBucket
```

Fig. 4.2 Controller output information during pinging h1 and h3

5. Quality of Service

Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks that may use any or all of these underlying technologies. The primary goal of QoS is to provide priority including dedicated bandwidth, controlled jitter and latency (required by some real-time and interactive traffic), and improved loss characteristics. Also important is making sure that providing priority for one or more flows does not make other flows fail. QoS technologies provide the elemental building blocks that will be used for future business applications in campus, WAN, and service provider networks. [6]

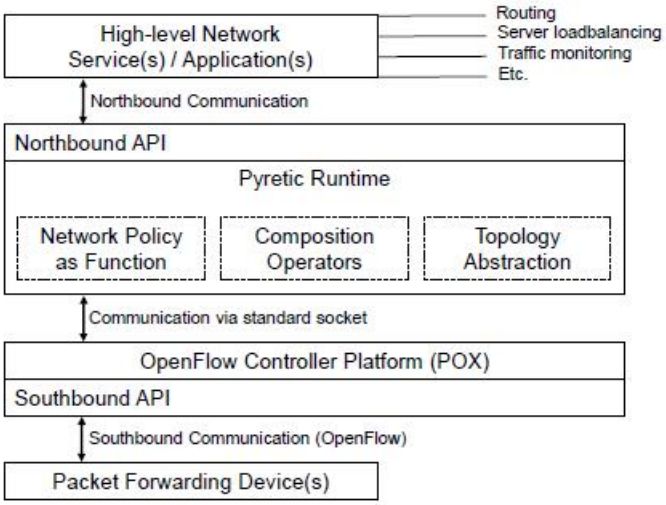

Fig. 5.1 Working QoS in Pyretic[7]

With Pyretic installed POX controller, QoS is still a major problem. Since Pyretic-VM only has development version, I used Miniedit from Mininet-VM and exported to .py file and then copy to Pyretic-VM to apply.

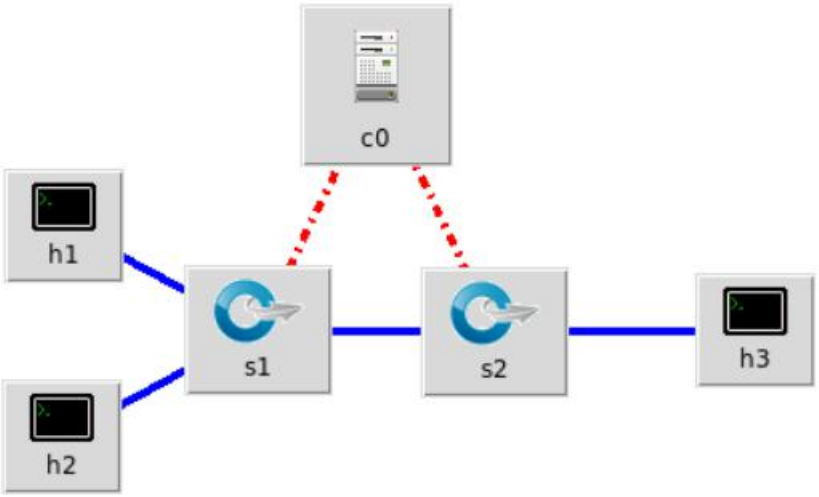

Fig. 5.1 Topology for QoS research

Open 3 hosts' terminals with,

```
mininet> xterm h1 h2 h3
```

In this 3 hosts and 2 switches network topology, I set up h1 as the server via command "iperf -s",

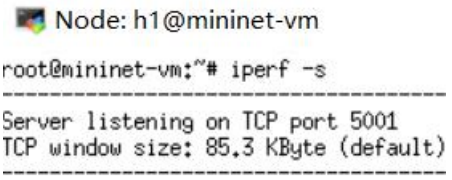

Fig. 5.2 Server-host

Which means h2 and h3 were clients. Applied "iperf -c 10.0.0.1 -r" to test bandwidth. Here, 10.0.0.1 was the IP address of server(h1), "-r" refers to round trip, both directions in sequence.

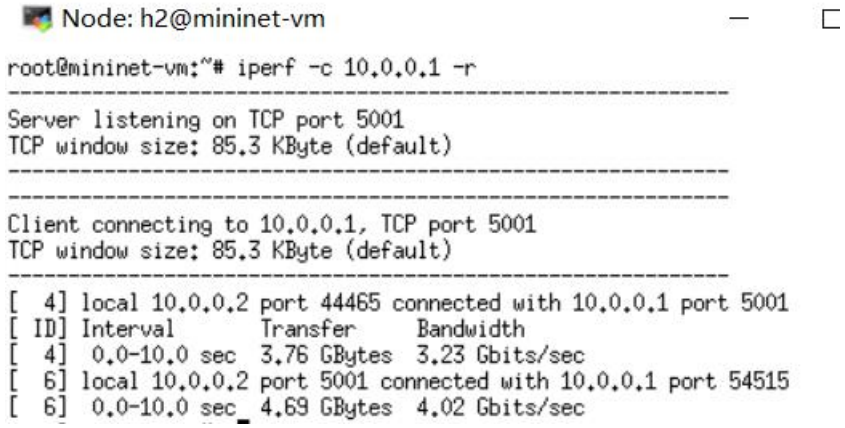

Fig. 5.3 Bandwidth to h2 before QoS

Similarly, client h3 with the following information after "iperf -c 10.0.0.1 -r"

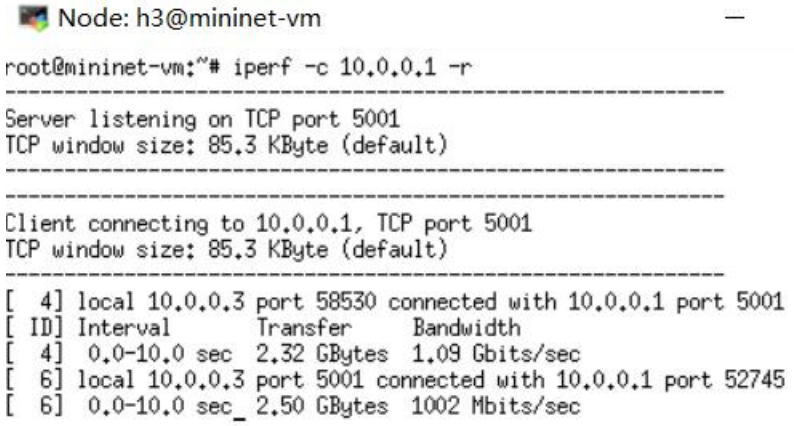

Fig. 5.4 Bandwidth to h3 before QoS

From the screen-shots above we can see that bandwidth between h1 and h2 were about 3.23 Gbits/s (h2 -> h1), 4.02 Gbits/s (h1 -> h2). And bandwidth between h1 and h3 were 1.09 Gbits/s (h3 -> h3), 1002 Mbits/s (h1 -> h3).

Now, in order to controller the bandwidth, a designed program was employed. It controller bandwidth through adjusting queue of switches. In this case, we need to control the bandwidth to be 30% of original only for h2.

This time, we run the controller with the module application - add_queue,

```
sudo pyretic.py pyretic.modules.add_queue
```

Then the controller with the application we need was up,

```
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.1.0 (betta) is up.
```

Use the same design and order of commands to set h1 to the server, and h2, h3 to clients as before. Now, use “iperf -c 10.0.0.1 -r” on both clients, we had the results below.

```
Node: h2@mininet-vm
root@mininet-vm:~# iperf -c 10.0.0.1 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size:  146 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.2 port 44411 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.19 GBytes  1.02 Gbits/sec
[  6] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 54461
[  6]  0.0-10.0 sec  1.14 GBytes   982 Mbits/sec
```

Fig. 5.5 Bandwidth to h3 after QoS

```
Node: h3@mininet-vm
root@mininet-vm:~# iperf -c 10.0.0.1 -r
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.3 port 58476 connected with 10.0.0.1 port 5001
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   985 MBytes   826 Mbits/sec
[  6] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 52691
[  6]  0.0-10.0 sec  1012 MBytes   849 Mbits/sec
```

Fig. 5.6 Bandwidth to h3 after QoS

Right now, bandwidth between h1 and h2 were decreased to 1.02 Gbits/s (h2 -> h1), 982 Mbits/s (h1 -> h2),. And bandwidth between h1 and h3 were 826 Mbits/s (h3 -> h3), 849 Mbits/s (h1 -> h3), which basically remains the same.

Before			After			Decrease (%)
h2-h1	h1-h2	Average	h2-h1	h1-h2	Average	
3.23 Gbits/s	4.02 Gbits/s	**3.625 Gbits/s**	1.02 Gbits/s	982 Mbits/s	**0.989 Gbits/s**	**27.28%**
h3-h1	h1-h3	Average	h3-h1	h1-h3	Average	/
1.09 Gbits/s	1002 Mbits/s	**1.03 Gbits/s**	826 Mbits/s	849 Mbits/s	**0.818 Gbits/s**	**79.42%**

Table 5.1 Comparison before and after applying QoS module

6. Conclusion

In this project, I have learned a lot. At beginning, I was working with Frenetic with programming language Ocaml (i.e with virtual machine - Frenetic-VM). However, there were at least 3 official of Frenetic-Ocaml-VM available online, which caused confusion with its instruction manual. At the end, I reported bugs and mistakes that happened during my experiment. Anyway, I had a basic knowledge of Frenetic-Ocaml and Frenetic structure.

Then, Pyretic as a substitution in my project, it worked perfectly and successfully showed the properties of Software Defined Network(SDN), which I used to code a layer 2 firewall as an application that run in SDN system. After that, I also did research on Quality of Service(QoS) in SDN. Unfortunately, I haven't get so far due to my background. Eventually, however, I still controlled data flow by adding extra queue.

This project gave me a perfect opportunity for getting familiar with Frenetic which is a popular project under SDN and working for it. Then, as a self-learner of QoS, based on the network environment I set, I also achieved the method to control the bandwidth.

*Reference

[1] SDN Architecture,The Tech, 2015
[2] Open Networking Foundation
https://www.opennetworking.org/sdn-resources/sdn-definition
[3] Modular SDN Programming with Pyretic, Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, David Walker, 2013
[4] Frenetic, http://frenetic-lang.org/
[5] Modular SDN Programming w/ Pyretic,Joshua Reich
Princeton University
[6] Quality of Service Networking, Cisco, 2013
[7] OpenFlow for QoS-Monitoring, Technische Universitat Munchen, Germany, Georg Carle

*Code

(1) firewall.py

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.modules.pyretic_switch import ActLikeSwitch
from csv import DictReader
from collections import namedtuple
import os

policy_file = "%s/pyretic/pyretic/examples/firewall-policies.csv" % os.environ[ 'HOME' ]
Policy = namedtuple('Policy', ('mac_0', 'mac_1'))

def main():
    # Read in the policies from the .csv file
    def read_policies (file):
        with open(file, 'r') as f:
            reader = DictReader(f, delimiter = ",")
            policies = {}
            for row in reader:
                policies[row['id']] = Policy(MAC(row['mac_0']), MAC(row['mac_1']))
        return policies

    policies = read_policies(policy_file)

    # Start with a policy that doesn't match any packets
    not_allowed = none

    # Add traffic that isn't allowed
    for policy in policies.itervalues():
        not_allowed = not_allowed | match(srcmac=policy.mac_0, dstmac=policy.mac_1) | match(srcmac=policy.mac_1, dstmac=policy.mac_0)

    # Allowed traffic in terms of not_allowed
    allowed = ~not_allowed

    # Only send allowed traffic to the mac learning (act_like_switch) logic
    return allowed >> ActLikeSwitch()
```

(2) pyretic_switch.py

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *

class ActLikeSwitch(DynamicPolicy):
    def __init__(self):
        super(ActLikeSwitch, self).__init__()
        # Set up the initial forwarding behavior for your mac learning switch to flood all packets
        self.forward = flood()

        # Set up a query that will receive new incoming packets
        self.query = packets(limit=1,group_by=['srcmac','switch'])
        # Set the initial internal policy value (each dynamic policy has a member 'policy'
        # when this member is assigned, the dynamic policy updates itself)
        self.policy = self.forward + self.query

        self.query.register_callback(self.learn_from_a_packet)

    def learn_from_a_packet(self, pkt):
        # Set the forwarding policy
        self.forward = if_(match(dstmac=pkt['srcmac'],
                                 switch=pkt['switch']), fwd(pkt['inport']),
                           self.forward)    # hint use 'match', '&', 'if_', and 'fwd'
        # Update the policy
        self.policy = self.forward + self.query # hint you've already written this
        print self.policy

def main():
    return ActLikeSwitch()
```

(3) firewall-policies.csv

```
id,mac_0,mac_1
1,00:00:00:00:00:01,00:00:00:00:00:02
```

(4) add_queue.csv

```
import os
import sys
import time
import subprocess

def find_all(a_str, sub_str):
    start = 0
    b_starts = []
    while True:
        start = a_str.find(sub_str, start)
        if start == -1: return b_starts
        b_starts.append(start)
        start += 1


if os.getuid() != 0:
        print "Root permissions required"
```

```
        exit()


cmd = "ovs-vsctl show"
p = os.popen(cmd).read()

brdgs = find_all(p, "Bridge")
print brdgs

switches = []
for bn in brdgs:
        sw =    p[(bn+8):(bn+10)]
        switches.append(sw)

ports = find_all(p,"Port")
print ports

prts = []
for prt in ports:
        prt = p[(prt+6):(prt+13)]
        if "" not in prt:
                print prt
                prts.append(prt)
config_strings = {}
for i in range(len(switches)):
        str = ""
        sw = switches[i]
        for n in range(len(prts)):
                #Verify the correct order
                if switches[i] in prts[n]:
                        port_name = prts[n]
                        str = str+" -- set port %s qos=@defaultqos" % port_name
        config_strings[sw] = str

#Build queues for every switch
print config_strings
for sw in switches:
        queuecmd = "sudo ovs-vsctl %s -- --id=@defaultqos create qos type=linux-htb
other-config:max-rate=1000000000 queues=0=@q0,1=@q1,2=@q2 -- --id=@q0 create queue
other-config:min-rate=1000000000 other-config:max-rate=1000000000 -- --id=@q1 create queue
other-config:max-rate=20000000 -- --id=@q2 create queue other-config:max-rate=1000000
other-config:min-rate=1000000" % config_strings[sw]
    q_res = os.popen(queuecmd).read()
        print q_res
        print queuecmd
```