

# **Open Flow Firewall Implementation**

**By Xiaoxin Lu**

Master of Science in Internetworking

University of Alberta, 2015

**Mentor: Pete Nanda**

## ABSTRACT

With rapid development of information technology, information security becomes more and more important in enterprises, governments and service providers networks. As we are transferring from traditional networks to SDN (Software Defined Network), the IT budget will be reduced dramatically due to the cutting cost of purchasing expensive security devices and deploying security services. In contrast with traditional networks, SDN offers centralised management, flexible deployment and programmability. Some SDN controllers were created to take the advantage of SDN's flexibility, but it is not flexible to add new modules with designated firewall functions on these controllers. A new network programming language, Pyretic, was developed to solve this problem. Programmers who use pyretic to develop SDN applications will focus on how to specify a network policy at a high level of abstraction instead of implementing it using low-level OpenFlow mechanisms [2]. This paper demonstrates the benefits of implementing firewall modules using pyretic language. The firewall modules are capable of some firewall functions, such as access control and denial of service prevention. This paper will also talk about a detailed implementation, test and analysis of firewall modules on Pyretic controller.

## Contents

1. INTRODUCTION.....	4
2. IMPLEMENTATION .....	6
2.1 Set up a SDN virtual network.....	6
2.2 Pyretic .....	7
2.2.1 Installing Pyretic.....	7
2.2.2 Running Pyretic .....	7
2.2.3 Using Match for Interested Packets.....	7
2.2.4 Query policies .....	9
2.2.5 Dynamic Policy .....	10
2.3 Network Diagram .....	11
2.4 Flow Chart .....	12
3. CONTROLLER TEST AND ANALYSIS.....	14
3.1 Connectivity Analysis .....	14
3.2 Denial of Service Test.....	19
4. FINDINGS.....	25
5. CONCLUSION AND FUTURE WORK .....	26
APPENDIX.....	27
Pyretic Code .....	27
Reference .....	32

## 1. INTRODUCTION

Pyretic is an open source network programming language. It uses modular programming techniques. In comparison with traditional OpenFlow programming, Pyretic can be used to write application modules independently. It also ensures that these modules will not interfere with one another. In addition, Pyretic is capable of creating a dynamic policy whose behavior will change over time, as specified by the programmer. Finally, Pyretic offers the ability to abstract the underlying network so programmers are dealing with high level abstractions when programming SDN controller.

Furthermore, Pyretic writes functions to generate network policies which are used to control underlying network forwarding behaviors. When Pyretic needs to do multiple tasks at same time, Pyretic policy compositions can be used to combine policies in series or in parallel. Pyretic writes complex policies easier and causes less conflicts between applications. Furthermore, Pyretic policy function can match any header field in packets and can change any header field value according to policies. Because using virtual packet head can create new headers, it makes forwarding methods more flexible.

Finally, Pyretic is running on top of a run-time system which is a SDN controller. The run-time system connects Pyretic and underlying network. It translates pyretic policies to open flows to install in the switches. The run-time system also mapping the underlying topology for pyretic's network virtualization purpose. Current pyretic version is using POX as run-time server. Due to this design aspect, Pyretic programmers are able to focus on codes from a high level abstract perspective. It offers some advantages like efficiency, less prone to errors and simplified design compared with other SDN controllers such as OpenDaylight or Trema.

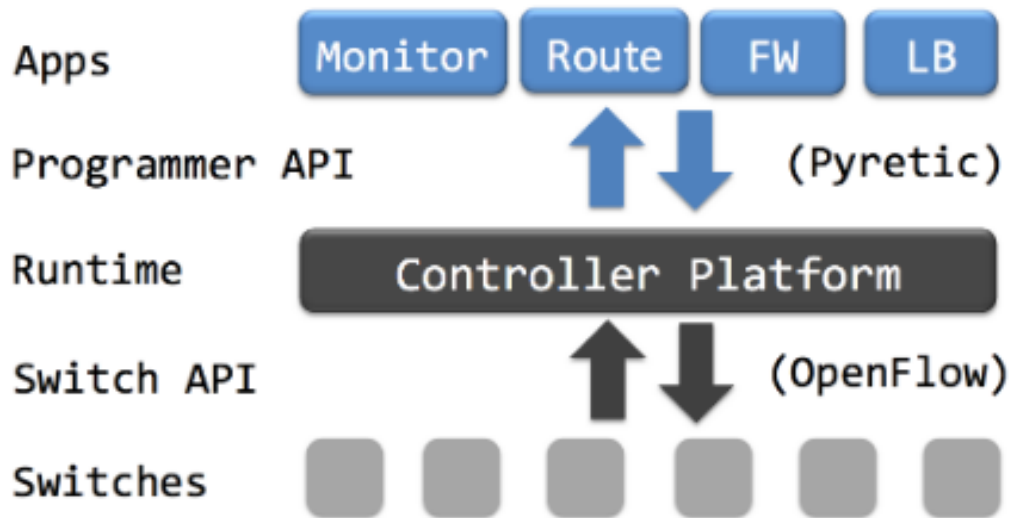
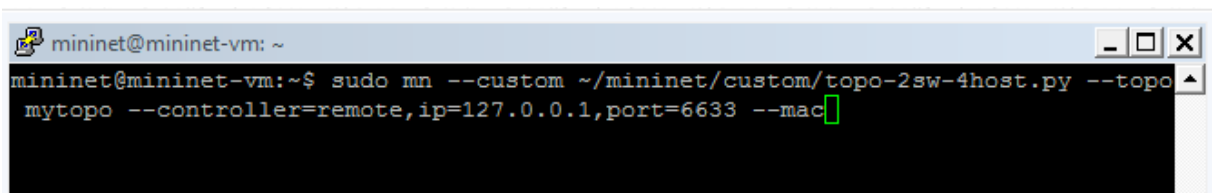


Figure 1: Software Defined Network [2]

## 2. IMPLEMENTATION

### 2.1 Set up a SDN virtual network

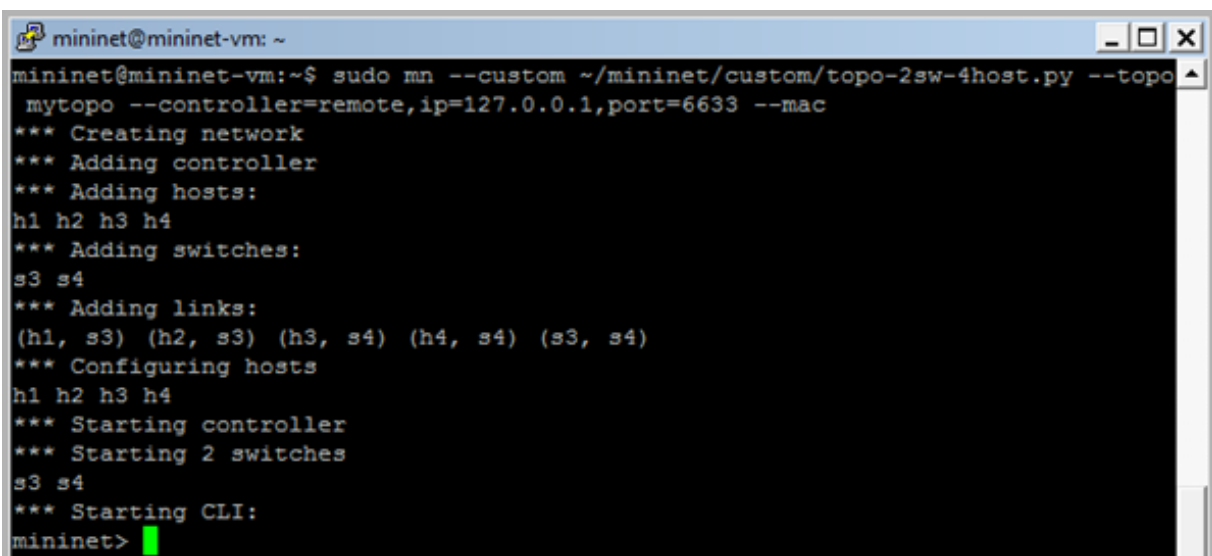
In this project Mininet is used to create a virtual network. Mininet is a network emulator which creates a network of virtual hosts, switches and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking [3]. Mininet official VM can be downloaded from Mininet official site. After running Mininet VM in Oracle VirtualBox, one line of command will create a virtual network (Figure 2.1.1).



```
mininet@mininet-vm: ~  
mininet@mininet-vm:~$ sudo mn --custom ~/mininet/custom/topo-2sw-4host.py --topo  
mytopo --controller=remote,ip=127.0.0.1,port=6633 --mac
```

Figure 2.1.1 Screenshot From Mininet VM

This virtual network includes four hosts, two switches. Two switches will connect to a remote controller at port 6633 which is Pyretic SDN controller (Figure 2.1.2). “—mac” option is used to assign static mac addresses for hosts.



```
mininet@mininet-vm: ~  
mininet@mininet-vm:~$ sudo mn --custom ~/mininet/custom/topo-2sw-4host.py --topo  
mytopo --controller=remote,ip=127.0.0.1,port=6633 --mac  
*** Creating network  
*** Adding controller  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s3 s4  
*** Adding links:  
(h1, s3) (h2, s3) (h3, s4) (h4, s4) (s3, s4)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
*** Starting 2 switches  
s3 s4  
*** Starting CLI:  
mininet>
```

Figure 2.1.2 Screenshot From Mininet VM

## 2.2 Pyretic

### 2.2.1 Installing Pyretic

There are two options to install Pyretic. One is to download pre-packaged Pyretic VM from Pyretic official website. The other is to manually install it inside the Mininet VM.

### 2.2.2 Running Pyretic

Pyretic is invoked by running “pyretic.py” followed by a mode and a module name. For example, the following command will invoke Pyretic and run a firewall module located inside the pyretic/modules directory using mode “Proactive”:

```
$ pyretic.py -m p0 pyretic.modules.fw
```

There are three pyretic runtime’s modes of operation.

- Interpreted (i) - every packet is processed in the controller runtime. Unsurprisingly slow, but useful for debugging [1].
- Reactive (r0) - rules are reactively pushed to switches based on the Pyretic policy and the packets seen [1].
- Proactive (p0) - rules are proactively pushed to switches based on the Pyretic policy. Generally the highest performant mode currently available [1].

### 2.2.3 Using Match for Interested Packets

Pyretic provides a lots of match fields to capture interested packets traversing the network [Figure 2.2.1]. The syntax is simple. For example, below is match packets whose field f has value v.

```
match (f=v)
```

Also several fields can be matched in one line using comma to separate them.

#match packets' source IP is 10.0.0.4 and destination port is 80  
**match(srcip=IPAddr("10.0.0.4"),dstport=80)**

Conjunction (&), disjunction (|), and negation (~) operators are very useful to compose more complicated matches.

#match packets from switch4 and source IP is 10.0.0.4 and destination port is 80  
**match(switch=4) & match(srcip=IPAddr("10.0.0.4"),dstport=80)**

#match packets whose source IP is 10.0.0.1 or 10.0.0.2 and protocol is 1  
**match(srcip=IPAddr('10.0.0.1'),protocol=1) | match(srcip=IPAddr('10.0.0.2'),protocol=1)**

#match packets whose destination mac address is not ff:ff:ff:ff:ff:ff  
**~match(dstmac=EthAddr('ff:ff:ff:ff:ff:ff'))**

FIELD	TYPE	EXAMPLE	NOTES
switch	int	4	
inport	int	3	
outport	int	2	
srcmac	EthAddr	EthAddr('00:00:00:00:00:01')	
dstmac	EthAddr	EthAddr('00:00:00:00:00:03')	
srcip	IPAddr or string	IPAddr('10.0.1.1'), '10.0.0.0/24'	
dstip	IPAddr or string	IPAddr('10.0.1.2'), '10.0.0.1/24'	
tos	int	0	
srcport	int	80	Requires the ethtype and protocol to be set
dstport	int	8080	
ethtype	int	1	
protocol	int	7	
vlan_id	int	0	
vlan_pcp	int	0	

Figure 2.2.1 Pyretic Match Fields [1]

## 2.2.4 Query policies

One of the most important tasks in the network management is network traffic monitoring. Pyretic offers a method called query policies to collect traffic statistics and install flows on the switches according to a statistic result. Query policies can also combine with other policies to compose a restricted policy. There are three different kinds of policies available in Pyretic. One is used to monitor raw packets. The other two are to count packets or bytes of packets.

For example, below codes will start a new query for each unique combination of source IP and destination IP for the first packet. “Self.query” also registers a callback function called “self.query\_action” to handle interested packets.

```
self.query = packets(1,['srcip','dstip'])  
self.query.register_callback(self.query_action)
```

Another one counts every packet and run callback function every 2 seconds grouped by source IP and destination IP.

```
self.query = count_packets(2,['srcip','dstip'])  
self.query.register_callback(self.query_action)
```

Count\_bytes function is doing the same thing as previous one, but instead of counting the packets, it counts the every byte of a packet received.

```
self.query = count_bytes(2,['srcip','dstip'])  
self.query.register_callback(self.query_action)
```

Last several lines of codes below composes a combination of query policy with match policy (self.query\_pkt >> self.query). It restricts what packets can be sent to “self.query” using a Sequential composition “>>”.

```
self.query_pkt= match(srcip=IPAddr('10.0.0.2'),dstip=IPAddr('10.0.0.4'))  
self.query = count_packets(2,['srcip','dstip'])  
self.query.register_callback(self.query_action)  
self.query_pkt >> self.query
```

### 2.2.5 Dynamic Policy

Dynamic policy is a very handy policy function to deal with policies changes at runtime. Query policies are often used to trigger these changes in dynamic policy. Dynamic policy defines a “self.policy” attribute which is dynamic and can take the changes from programmer's specification. For example, below class Access\_control uses class DynamicPolicy as a super class. It uses a query policy to run “query\_action” when new packets with unique source and destination IPs are coming. Method “update\_policy’s” behavior is to assign “self.policy” a new value. So all future packet with these source and destination IPs will follow the new policies’ rules.

```
class Access_control(DynamicPolicy):  
  
    def start_query(self):  
        self.query =packets(1,['srcip','dstip'])  
        self.query.register_callback(self.query_action)  
  
    def query_action(self):  
        self.update_policy()  
  
    def update_policy(self):  
        self.policy= self.whitelist + self.query
```

## 2.3 Network Diagram

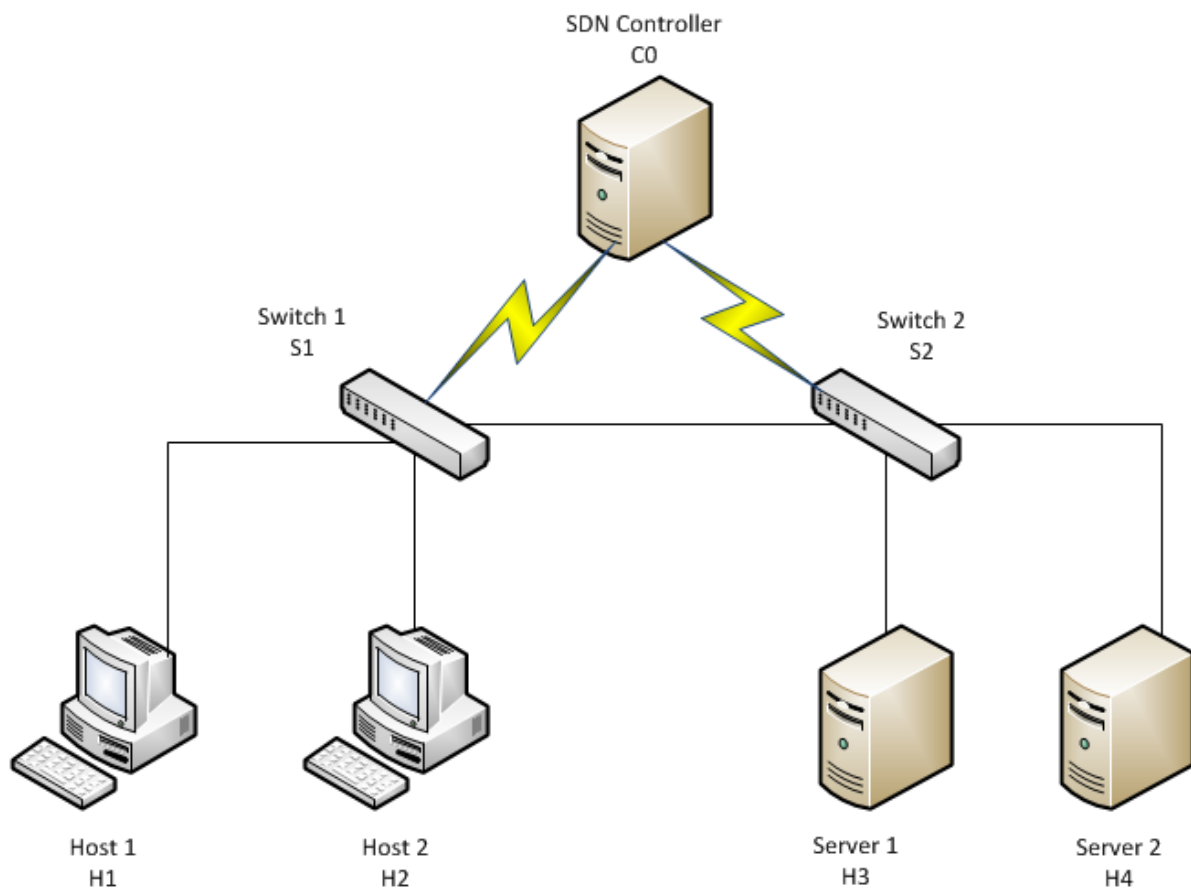


Figure 2.3 Network Diagram

Host Name	Mac Address	IP Address	Note
H1	00:00:00:00:00:01	10.0.0.1	
H2	00:00:00:00:00:02	10.0.0.2	
H3	00:00:00:00:00:03	10.0.0.3	
H4	00:00:00:00:00:04	10.0.0.4	
C0	n/a	127.0.0.1	port 6633

Table 2.3 IP and Mac Address Table

## 2.4 Flow Chart

The Flow chart gives a detailed picture of how the controller will do when packets start to come in. The controller has three main modules. These are ACL, DOS and Switch modules. ACL module checks whether packets are allowed to pass through switches. First it checks ACL white list which is preconfigured in the module to find a match. Then, if a match is found, ACL assigns dynamic self.policy with a new policy. In the new policy, these allowed packets are sent to the Switch module. Switch module's function is to find a route for these allowed packets. For Dos module, it keeps polling packets' statistics from switches. If numbers exceed the maximum, Dos will change the self.policy and remove relative flows from it.

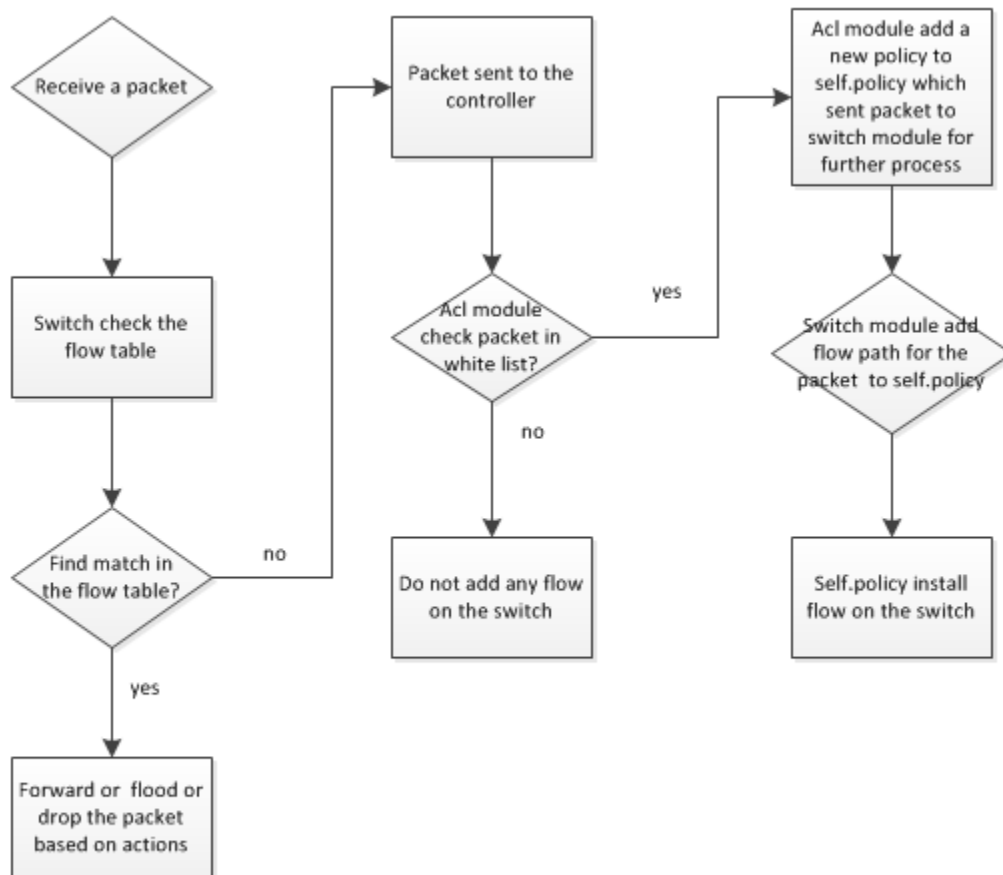


Figure: 2.4.1 ACL and Switch Module

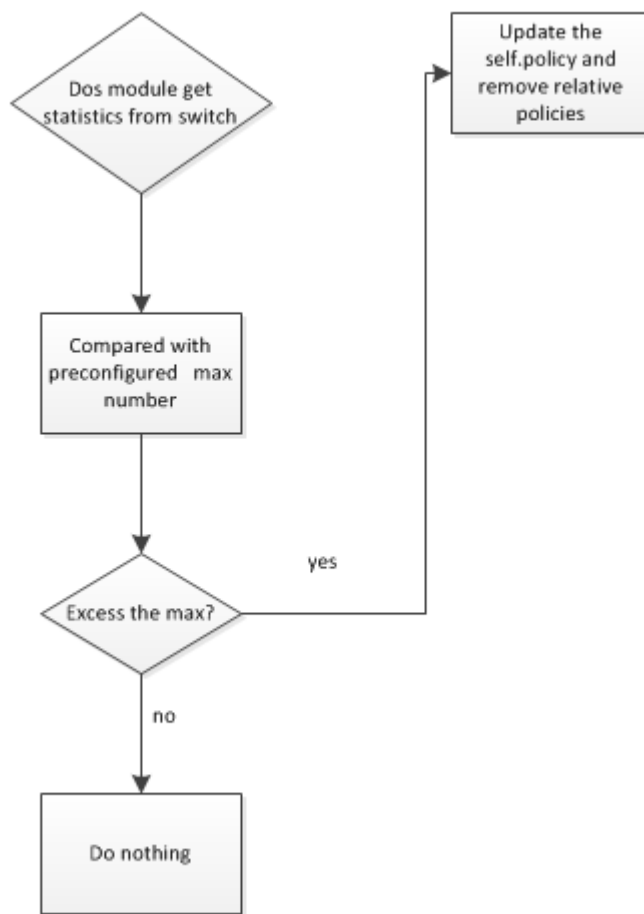


Figure: 2.4.2 DOS Module

### 3. CONTROLLER TEST AND ANALYSIS

#### 3.1 Connectivity Analysis

Firewall module loads ACLs' from a CSV file. These ACLs have some allows and denies (Figure 3.1).

Host	HTTP	Telnet	Deny of Service Attack
Host 1 (H1)	Deny	Allow	Deny
Host 2 (H2)	Allow	Deny	Deny

Figure: 3.1.1 ACL Configuration

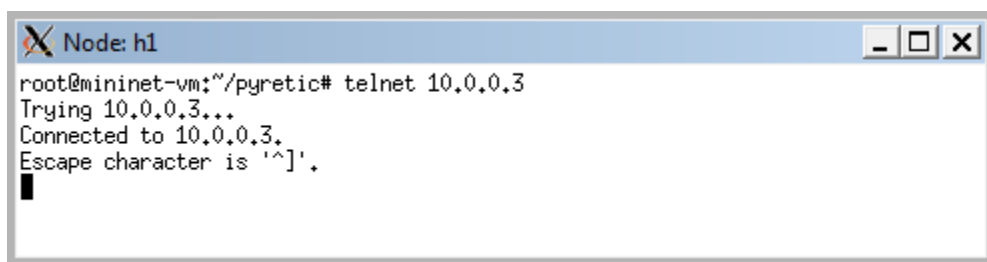
There are two hosts (h3 and h4) which are running network services. H3 is used as a Telnet server and H4 is a web server.

H1 can telnet into H3, but cannot access H4's webpage.



```
Node: h3
root@mininet-vm:~/pyretic# python -m SimpleHTTPServer 23
Serving HTTP on 0.0.0.0 port 23 ...
```

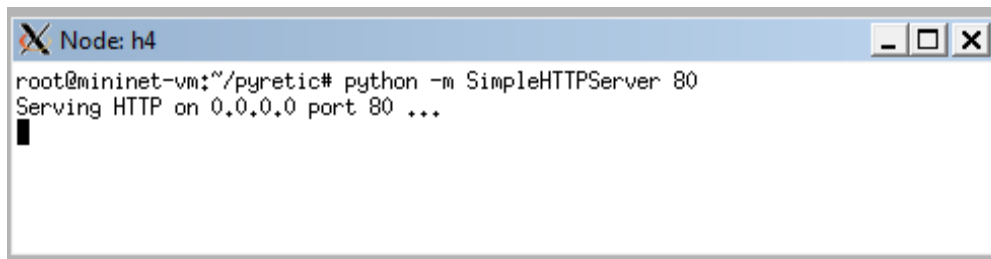
Figure: 3.1.2 Telnet Service on H3



```
Node: h1
root@mininet-vm:~/pyretic# telnet 10.0.0.3
Trying 10.0.0.3...
Connected to 10.0.0.3.
Escape character is '^]'.

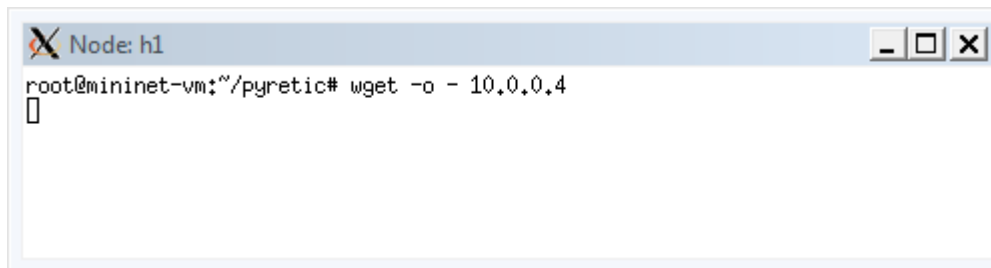
```

Figure: 3.1.3 H1 Telnet into H3



```
Node: h4
root@mininet-vm:~/pyretic# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
█
```

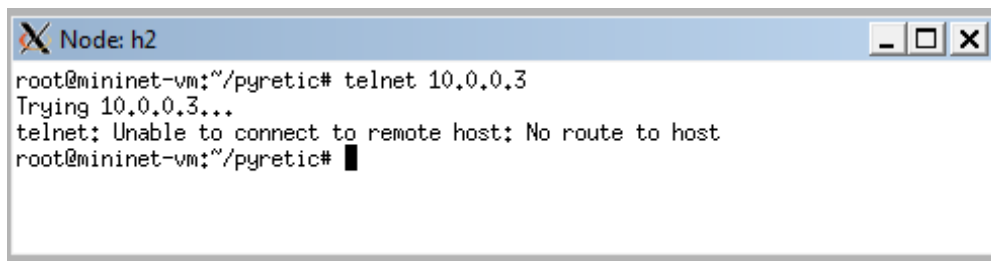
Figure: 3.1.4 Web Service on H4



```
Node: h1
root@mininet-vm:~/pyretic# wget -o - 10.0.0.4
█
```

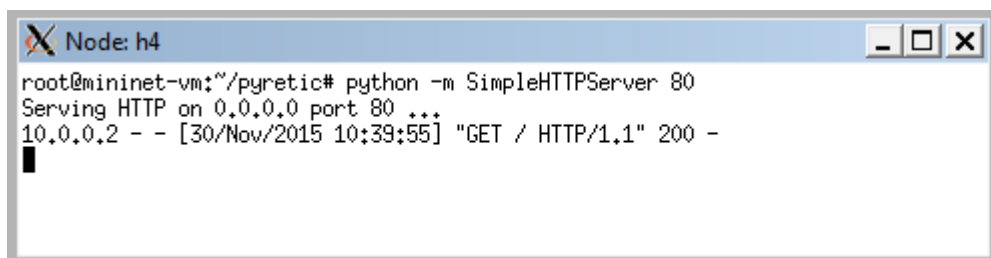
Figure: 3.1.5 H1 Failed to access H4 Web Server

On the contrary, H2 cannot telnet into H3, but can access H4's webpage.



```
Node: h2
root@mininet-vm:~/pyretic# telnet 10.0.0.3
Trying 10.0.0.3...
telnet: Unable to connect to remote host: No route to host
root@mininet-vm:~/pyretic# █
```

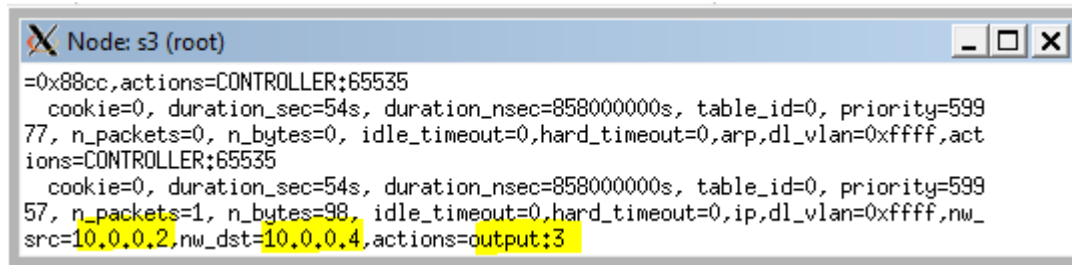
Figure: 3.1.6 H2 Failed to telnet H3



```
Node: h4
root@mininet-vm:~/pyretic# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.2 - - [30/Nov/2015 10:39:55] "GET / HTTP/1.1" 200 -
█
```

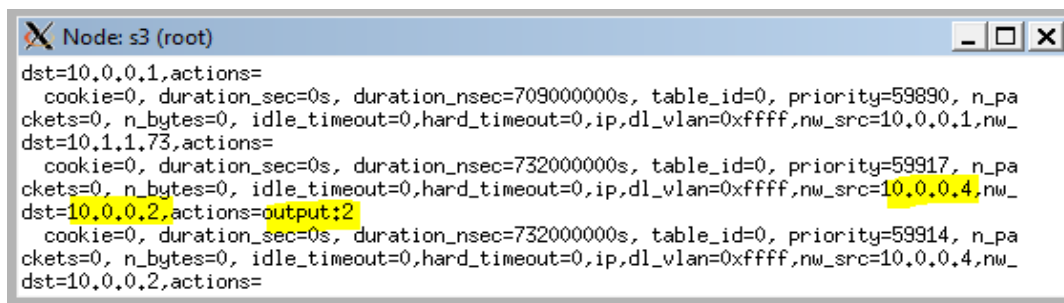
Figure: 3.1.7 H2 Connect to H4 Web Server

After using command (dptcl dump-flows tcp: 127.0.0.1:6634) on switch3 (S3), the flow table of S3 shows the controller installed flows with output actions. Each flow includes source IP, destination IP and action fields. The actions point out the egress ports for each pair of IPs.



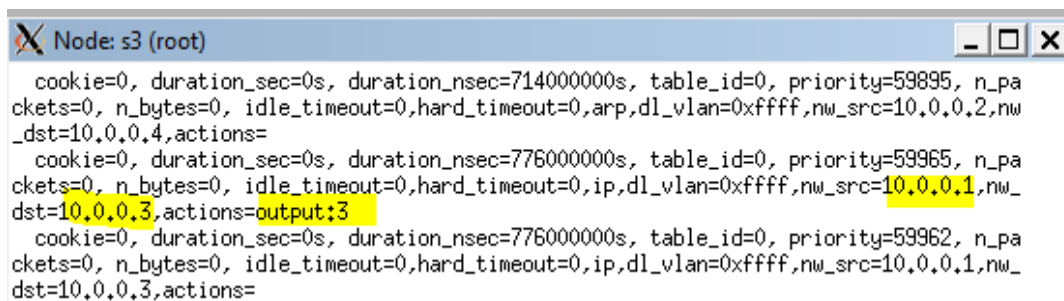
```
Node: s3 (root)
=0x88cc,actions=CONTROLLER;65535
  cookie=0, duration_sec=54s, duration_nsec=858000000s, table_id=0, priority=59977, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, arp, dl_vlan=0xffff, actions=CONTROLLER;65535
  cookie=0, duration_sec=54s, duration_nsec=858000000s, table_id=0, priority=59957, n_packets=1, n_bytes=98, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_src=10.0.0.2, nw_dst=10.0.0.4, actions=output:3
```

Figure: 3.1.8 S3's Flow Entry from Src 10.0.0.2 to Dst 10.0.0.4



```
Node: s3 (root)
dst=10.0.0.1,actions=
  cookie=0, duration_sec=0s, duration_nsec=709000000s, table_id=0, priority=59890, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_src=10.0.0.1, nw_dst=10.1.1.73,actions=
  cookie=0, duration_sec=0s, duration_nsec=732000000s, table_id=0, priority=59917, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_src=10.0.0.4, nw_dst=10.0.0.2,actions=output:2
  cookie=0, duration_sec=0s, duration_nsec=732000000s, table_id=0, priority=59914, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_src=10.0.0.4, nw_dst=10.0.0.2,actions=
```

Figure: 3.1.9 S3's Flow Entry from Src 10.0.0.4 to Dst 10.0.0.2



```
Node: s3 (root)
  cookie=0, duration_sec=0s, duration_nsec=714000000s, table_id=0, priority=59895, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, arp, dl_vlan=0xffff, nw_src=10.0.0.2, nw_dst=10.0.0.4,actions=
  cookie=0, duration_sec=0s, duration_nsec=776000000s, table_id=0, priority=59965, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_src=10.0.0.1, nw_dst=10.0.0.3,actions=output:3
  cookie=0, duration_sec=0s, duration_nsec=776000000s, table_id=0, priority=59962, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_src=10.0.0.1, nw_dst=10.0.0.3,actions=
```

Figure: 3.1.10 S3's Flow Entry from Src 10.0.0.1 to Dst 10.0.0.3

```
Node: s3 (root)
02, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_s
rc=10.0.0.1, nw_dst=10.0.0.3, actions=
  cookie=0, duration_sec=54s, duration_nsec=818000000s, table_id=0, priority=599
33, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_s
rc=10.0.0.3, nw_dst=10.0.0.1, actions=output:1
  cookie=0, duration_sec=54s, duration_nsec=818000000s, table_id=0, priority=599
30, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_s
```

Figure: 3.1.11 S3's Flow Entry from Src 10.0.0.3 to Dst 10.0.0.1

Issue the same command on switch4 (S4).

```
Node: s4 (root)
  cookie=0, duration_sec=0s, duration_nsec=395000000s, table_id=0, priority=5997
5, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, arp, dl_vlan=0xffff, acti
ons=CONTROLLER:65535
  cookie=0, duration_sec=0s, duration_nsec=351000000s, table_id=0, priority=5995
5, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.2, nw_dst=10.0.0.4, actions=output:3
  cookie=0, duration_sec=0s, duration_nsec=351000000s, table_id=0, priority=5995
2, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.2, nw_dst=10.0.0.4, actions=
  cookie=0, duration_sec=0s, duration_nsec=317000000s, table_id=0, priority=5989
```

Figure: 3.1.12 S4's Flow Entry from Src 10.0.0.2 to Dst 10.0.0.4

```
Node: s4 (root)
0, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.1, nw_dst=10.1.1.73, actions=
  cookie=0, duration_sec=0s, duration_nsec=339000000s, table_id=0, priority=5991
7, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.4, nw_dst=10.0.0.2, actions=output:2
  cookie=0, duration_sec=0s, duration_nsec=337000000s, table_id=0, priority=5991
4, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.4, nw_dst=10.0.0.2, actions=
  cookie=0, duration_sec=0s, duration_nsec=315000000s, table_id=0, priority=5989
4, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.4, nw_dst=10.0.0.2, actions=
```

Figure: 3.1.13 S4's Flow Entry from Src 10.0.0.4 to Dst 10.0.0.2

```
Node: s4 (root)
rc=10.0.0.2, nw_dst=10.0.0.4, actions=
  cookie=0, duration_sec=0s, duration_nsec=316000000s, table_id=0, priority=5989
5, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, arp, dl_vlan=0xffff, nw_s
rc=10.0.0.2, nw_dst=10.0.0.4, actions=
  cookie=0, duration_sec=0s, duration_nsec=351000000s, table_id=0, priority=5996
5, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.1, nw_dst=10.0.0.3, actions=output:3
  cookie=0, duration_sec=0s, duration_nsec=351000000s, table_id=0, priority=5996
2, n_packets=0, n_bytes=0, idle_timeout=0, hard_timeout=0, ip, dl_vlan=0xffff, nw_sr
c=10.0.0.1, nw_dst=10.0.0.3, actions=
```

Figure: 3.1.14 S4's Flow Entry from Src 10.0.0.1 to Dst 10.0.0.3

```

Node: s4 (root)
c=10.0.0.1,nw_dst=10.1.1.100,actions=
  cookie=0, duration_sec=0s, duration_nsec=395000000s, table_id=0, priority=5997
7, n_packets=0, n_bytes=0, idle_timeout=0,hard_timeout=0,arp,d_l_vlan=0xffff,nw_s
rc=10.0.0.1,nw_dst=10.1.1.100,actions=
  cookie=0, duration_sec=0s, duration_nsec=312000000s, table_id=0, priority=5989
1, n_packets=0, n_bytes=0, idle_timeout=0,hard_timeout=0,arp,d_l_vlan=0xffff,nw_s
rc=10.0.0.1,nw_dst=10.1.1.100,actions=
  cookie=0, duration_sec=0s, duration_nsec=351000000s, table_id=0, priority=5993
1, n_packets=0, n_bytes=0, idle_timeout=0,hard_timeout=0,ip,d_l_vlan=0xffff,nw_sr
c=10.0.0.3,nw_dst=10.0.0.1,actions=output:1

```

Figure: 3.1.15 S4's Flow Entry from Src 10.0.0.3 to Dst 10.0.0.1

Also according to wireshark captured packets, h1 can telnet into host 3 and H2 can get web access on h4.

No.	Time	Source	Destination	Protocol	Length	Info
279	352.2723870	10.0.0.1	10.0.0.3	TCP	74	38189 > telnet [SYN] Seq=0 Win=14600 Len=...
280	352.2734360	10.0.0.3	10.0.0.1	TCP	74	telnet > 38189 [SYN, ACK] Seq=0 Ack=1 Win=...
281	352.2736100	10.0.0.1	10.0.0.3	TCP	66	38189 > telnet [ACK] Seq=1 Ack=1 Win=1484...
284	352.2724840	10.0.0.1	10.0.0.3	TCP	74	38189 > telnet [SYN] Seq=0 Win=14600 Len=...
285	352.2733860	10.0.0.3	10.0.0.1	TCP	74	telnet > 38189 [SYN, ACK] Seq=0 Ack=1 Win=...
286	352.2736190	10.0.0.1	10.0.0.3	TCP	66	38189 > telnet [ACK] Seq=1 Ack=1 Win=1484...
288	352.2723450	10.0.0.1	10.0.0.3	TCP	74	38189 > telnet [SYN] Seq=0 Win=14600 Len=...
289	352.2735470	10.0.0.3	10.0.0.1	TCP	74	telnet > 38189 [SYN, ACK] Seq=0 Ack=1 Win=...
290	352.2735910	10.0.0.1	10.0.0.3	TCP	66	38189 > telnet [ACK] Seq=1 Ack=1 Win=1484...
293	352.2723770	10.0.0.1	10.0.0.3	TCP	74	38189 > telnet [SYN] Seq=0 Win=14600 Len=...
294	352.2734450	10.0.0.3	10.0.0.1	TCP	74	telnet > 38189 [SYN, ACK] Seq=0 Ack=1 Win=...
295	352.2736020	10.0.0.1	10.0.0.3	TCP	66	38189 > telnet [ACK] Seq=1 Ack=1 Win=1484...
296	352.2764770	10.0.0.1	10.0.0.3	TELNET	93	Telnet Data ...
297	352.2764640	10.0.0.1	10.0.0.3	TELNET	93	[TCP Retransmission] Telnet Data ...
298	352.2764280	10.0.0.1	10.0.0.3	TELNET	93	[TCP Retransmission] Telnet Data ...
299	352.2764530	10.0.0.1	10.0.0.3	TELNET	93	[TCP Retransmission] Telnet Data ...
300	352.2767480	10.0.0.3	10.0.0.1	TCP	66	telnet > 38189 [ACK] Seq=1 Ack=28 Win=148...
301	352.2767310	10.0.0.3	10.0.0.1	TCP	66	[TCP Dup ACK 300#1] telnet > 38189 [ACK]
302	352.2767730	10.0.0.3	10.0.0.1	TCP	66	[TCP Dup ACK 300#2] telnet > 38189 [ACK]
303	352.2767600	10.0.0.3	10.0.0.1	TCP	66	[TCP Dup ACK 300#3] telnet > 38189 [ACK]
312	357.4000210	10.0.0.1	10.0.0.3	TELNET	69	Telnet Data ...
313	357.4002940	10.0.0.3	10.0.0.1	TCP	66	telnet > 38189 [ACK] Seq=1 Ack=31 Win=148...
314	357.4012230	10.0.0.3	10.0.0.1	TELNET	332	Telnet Data ...
315	357.4012430	10.0.0.1	10.0.0.3	TCP	66	38189 > telnet [ACK] Seq=31 Ack=267 Win=1...
316	357.4013610	10.0.0.3	10.0.0.1	TCP	66	telnet > 38189 [FIN, ACK] Seq=267 Ack=31...
317	357.4016430	10.0.0.1	10.0.0.3	TCP	66	38189 > telnet [FIN, ACK] Seq=31 Ack=268...

Figure: 3.1.16 Wireshark Captured Telnet Packets from h1 to h3

No.	Time	Source	Destination	Protocol	Length	Info
348	370.499086000	10.0.0.1	10.0.0.4	TCP	74	39972 > http [SYN] Seq=0 Win=14600 Len=0
349	371.495067000	10.0.0.1	10.0.0.4	TCP	74	39972 > http [SYN] Seq=0 Win=14600 Len=0
350	373.499576000	10.0.0.1	10.0.0.4	TCP	74	39972 > http [SYN] Seq=0 Win=14600 Len=0
351	377.511835000	10.0.0.1	10.0.0.4	TCP	74	39972 > http [SYN] Seq=0 Win=14600 Len=0
352	385.536097000	10.0.0.1	10.0.0.4	TCP	74	39972 > http [SYN] Seq=0 Win=14600 Len=0
364	401.591073000	10.0.0.1	10.0.0.4	TCP	74	39972 > http [SYN] Seq=0 Win=14600 Len=0
374	405.758508000	10.0.0.2	10.0.0.4	TCP	74	45102 > http [SYN] Seq=0 Win=14600 Len=0
375	405.758540000	10.0.0.4	10.0.0.2	TCP	74	http > 45102 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
376	405.758555000	10.0.0.2	10.0.0.4	TCP	74	45102 > http [SYN] Seq=0 Win=14600 Len=0
377	405.758571000	10.0.0.4	10.0.0.2	TCP	74	http > 45102 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
378	405.758817000	10.0.0.2	10.0.0.4	TCP	66	45102 > http [ACK] Seq=1 Ack=1 Win=14848 Len=0
379	405.758910000	10.0.0.2	10.0.0.4	TCP	66	[TCP Dup ACK 378#1] 45102 > http [ACK] Seq=1 Ack=1 Win=14848 Len=0
380	405.758950000	10.0.0.2	10.0.0.4	HTTP	172	GET / HTTP/1.1
381	405.758969000	10.0.0.4	10.0.0.2	TCP	66	http > 45102 [ACK] Seq=1 Ack=107 Win=14848 Len=0
383	405.758342000	10.0.0.2	10.0.0.4	TCP	74	45102 > http [SYN] Seq=0 Win=14600 Len=0
384	405.758346000	10.0.0.2	10.0.0.4	TCP	74	45102 > http [SYN] Seq=0 Win=14600 Len=0
385	405.758694000	10.0.0.4	10.0.0.2	TCP	74	http > 45102 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
386	405.758784000	10.0.0.2	10.0.0.4	TCP	66	45102 > http [ACK] Seq=1 Ack=1 Win=14848 Len=0
387	405.758863000	10.0.0.4	10.0.0.2	TCP	74	http > 45102 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
388	405.758882000	10.0.0.2	10.0.0.4	TCP	66	45102 > http [ACK] Seq=1 Ack=1 Win=14848 Len=0
389	405.758925000	10.0.0.2	10.0.0.4	HTTP	172	[TCP Retransmission] GET / HTTP/1.1
390	405.758996000	10.0.0.4	10.0.0.2	TCP	66	http > 45102 [ACK] Seq=1 Ack=107 Win=14848 Len=0
392	405.758400000	10.0.0.2	10.0.0.4	TCP	74	45102 > http [SYN] Seq=0 Win=14600 Len=0
393	405.758426000	10.0.0.2	10.0.0.4	TCP	74	45102 > http [SYN] Seq=0 Win=14600 Len=0
394	405.758602000	10.0.0.4	10.0.0.2	TCP	74	http > 45102 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
395	405.758650000	10.0.0.4	10.0.0.2	TCP	74	http > 45102 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0

Figure: 3.1.16 Wireshark Captured HTTP Packets from H2 to H4

## 3.2 Denial of Service Test

Because Mininet creates a virtual network, in order to simulate a DoS attack, I am using the ping command with the -i option which manipulates the number of pings for one second. In the Dos module, a static threshold of 20 has been set, so that when the number of packets per second exceeds the threshold, it triggers the controller to build a new self.policy to drop any further packets from the attacker. The legitimate traffic from other hosts will not be affected. Figure 3.1.1 and 3.2.2 will show the difference of two self.policies before and after attack. When the attack meets the threshold, the match policies related to the attack have been removed from self.policy.

```
putty - Notepad
File Edit Format View Help
===== PUTTY log 2015.12.01 10:39:58 =====
mininet> h2 ping h4 -i .05
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_req=1 ttl=64 time=0.227 ms
64 bytes from 10.0.0.4: icmp_req=2 ttl=64 time=0.140 ms
64 bytes from 10.0.0.4: icmp_req=3 ttl=64 time=0.187 ms
64 bytes from 10.0.0.4: icmp_req=4 ttl=64 time=0.105 ms
64 bytes from 10.0.0.4: icmp_req=5 ttl=64 time=0.105 ms
64 bytes from 10.0.0.4: icmp_req=6 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=7 ttl=64 time=0.112 ms
64 bytes from 10.0.0.4: icmp_req=8 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=9 ttl=64 time=0.122 ms
64 bytes from 10.0.0.4: icmp_req=10 ttl=64 time=0.088 ms
64 bytes from 10.0.0.4: icmp_req=11 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=12 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=13 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=14 ttl=64 time=0.130 ms
64 bytes from 10.0.0.4: icmp_req=15 ttl=64 time=0.095 ms
64 bytes from 10.0.0.4: icmp_req=16 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=17 ttl=64 time=0.088 ms
64 bytes from 10.0.0.4: icmp_req=18 ttl=64 time=0.184 ms
64 bytes from 10.0.0.4: icmp_req=19 ttl=64 time=0.183 ms
64 bytes from 10.0.0.4: icmp_req=20 ttl=64 time=0.092 ms
64 bytes from 10.0.0.4: icmp_req=21 ttl=64 time=0.090 ms
64 bytes from 10.0.0.4: icmp_req=22 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=23 ttl=64 time=0.088 ms
64 bytes from 10.0.0.4: icmp_req=24 ttl=64 time=0.088 ms
64 bytes from 10.0.0.4: icmp_req=25 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=26 ttl=64 time=0.096 ms
64 bytes from 10.0.0.4: icmp_req=27 ttl=64 time=0.118 ms
64 bytes from 10.0.0.4: icmp_req=28 ttl=64 time=0.156 ms
64 bytes from 10.0.0.4: icmp_req=29 ttl=64 time=0.094 ms
64 bytes from 10.0.0.4: icmp_req=30 ttl=64 time=0.141 ms
64 bytes from 10.0.0.4: icmp_req=31 ttl=64 time=0.098 ms
```

Figure: 3.2.1 H2 Attacks H4 Web Server

```

dos - Notepad
File Edit Format View Help
===== PuTTY log 2015.12.01 10:16:59 =====
('*****self.policy before attack*****', parallel:
  [DynamicPolicy]
  parallel:
    sequential:
      if
        match: ('ethype', 2054)
      then
        identity
      else
        if
          union:
            match: ('srcip', '10.0.0.1') ('dstip', '10.0.0.3') ('dstport', 23)
            match: ('srcip', '10.0.0.2') ('dstip', '10.0.0.4') ('dstport', 80)
            match: ('srcip', '10.0.0.1') ('dstip', '10.0.0.3') ('protocol', 1)
            match: ('srcip', '10.0.0.2') ('dstip', '10.0.0.4') ('protocol', 1)
            match: ('srcip', '10.0.0.3') ('srcport', 23) ('dstip', '10.0.0.1')
            match: ('srcip', '10.0.0.4') ('srcport', 80) ('dstip', '10.0.0.2')
            match: ('srcip', '10.0.0.3') ('dstip', '10.0.0.1') ('protocol', 1)
            match: ('srcip', '10.0.0.4') ('dstip', '10.0.0.2') ('protocol', 1)
          then
            identity
          else
            drop
        parallel:
          sequential:
            match: ('dstip', 10.0.0.1)
            parallel:
              sequential:
                match: ('switch', 4)
                fwd 1
              sequential:
                match: ('switch', 3)
                fwd 1
            sequential:
              match: ('dstip', 10.0.0.2)
              parallel:
                sequential:
                  match: ('switch', 4)
                  fwd 1
                sequential:
                  match: ('switch', 3)
                  fwd 2
            sequential:
              match: ('dstip', 10.0.0.3)
              parallel:
                sequential:
                  match: ('switch', 4)
                  fwd 2
                sequential:
                  match: ('switch', 3)
                  fwd 3
            sequential:
              match: ('dstip', 10.0.0.4)
              parallel:
                sequential:
                  match: ('switch', 4)
                  fwd 3
                sequential:
                  match: ('switch', 3)
                  fwd 3
          packets
          sequential:
            LimitFilter
            negate:
              union:
                match: ('srcip', 10.0.0.1) ('dstip', 10.0.0.3)
                match: ('srcip', 10.0.0.3) ('dstip', 10.0.0.1)
                match: ('srcip', 10.0.0.2) ('dstip', 10.0.0.4)
                match: ('srcip', 10.0.0.4) ('dstip', 10.0.0.2)
              FwdBucket
            sequential:
              match: ('srcip', 10.0.0.2) ('dstip', 10.0.0.4) ('protocol', 1)

```

Figure: 3.2.2 Self.policy Before attack

```

dos - Notepad
File Edit Format View Help
('*****self.policy after attack*****', parallel:
[DynamicPolicy]
parallel:
sequential:
if
match: ('ethype', 2054)
then
identity
else
union:
match: ('srcip', '10.0.0.1') ('dstip', '10.0.0.3') ('dstport', 23)
match: ('srcip', '10.0.0.1') ('dstip', '10.0.0.3') ('protocol', 1)
match: ('srcip', '10.0.0.3') ('srcport', 23) ('dstip', '10.0.0.1')
match: ('srcip', '10.0.0.3') ('dstip', '10.0.0.1') ('protocol', 1)
parallel:
sequential:
match: ('dstip', 10.0.0.1)
parallel:
sequential:
match: ('switch', 4)
fwd 1
sequential:
match: ('switch', 3)
fwd 1
sequential:
match: ('dstip', 10.0.0.2)
parallel:
sequential:
match: ('switch', 4)
fwd 1
sequential:
match: ('switch', 3)
fwd 2
sequential:
match: ('dstip', 10.0.0.3)
parallel:
sequential:
match: ('switch', 4)
fwd 2
sequential:
match: ('switch', 3)
fwd 3
sequential:
match: ('dstip', 10.0.0.4)
parallel:
sequential:
match: ('switch', 4)
fwd 3
sequential:
match: ('switch', 3)
fwd 3
packets
sequential:
LimitFilter
negate:
union:
match: ('srcip', 10.0.0.1) ('dstip', 10.0.0.3)
match: ('srcip', 10.0.0.3) ('dstip', 10.0.0.1)
match: ('srcip', 10.0.0.2) ('dstip', 10.0.0.4)
match: ('srcip', 10.0.0.4) ('dstip', 10.0.0.2)
FwdBucket
sequential:
match: ('srcip', 10.0.0.2) ('dstip', 10.0.0.4) ('protocol', 1)
FwdBucket)

```

Figure: 3.2.3 Self.policy after attack

After changes made on the self.policy, the controller sends new flows to switches. So when H2 ping H4 again, there is no action associated with the flow. Packets from H2 will be dropped by switches.

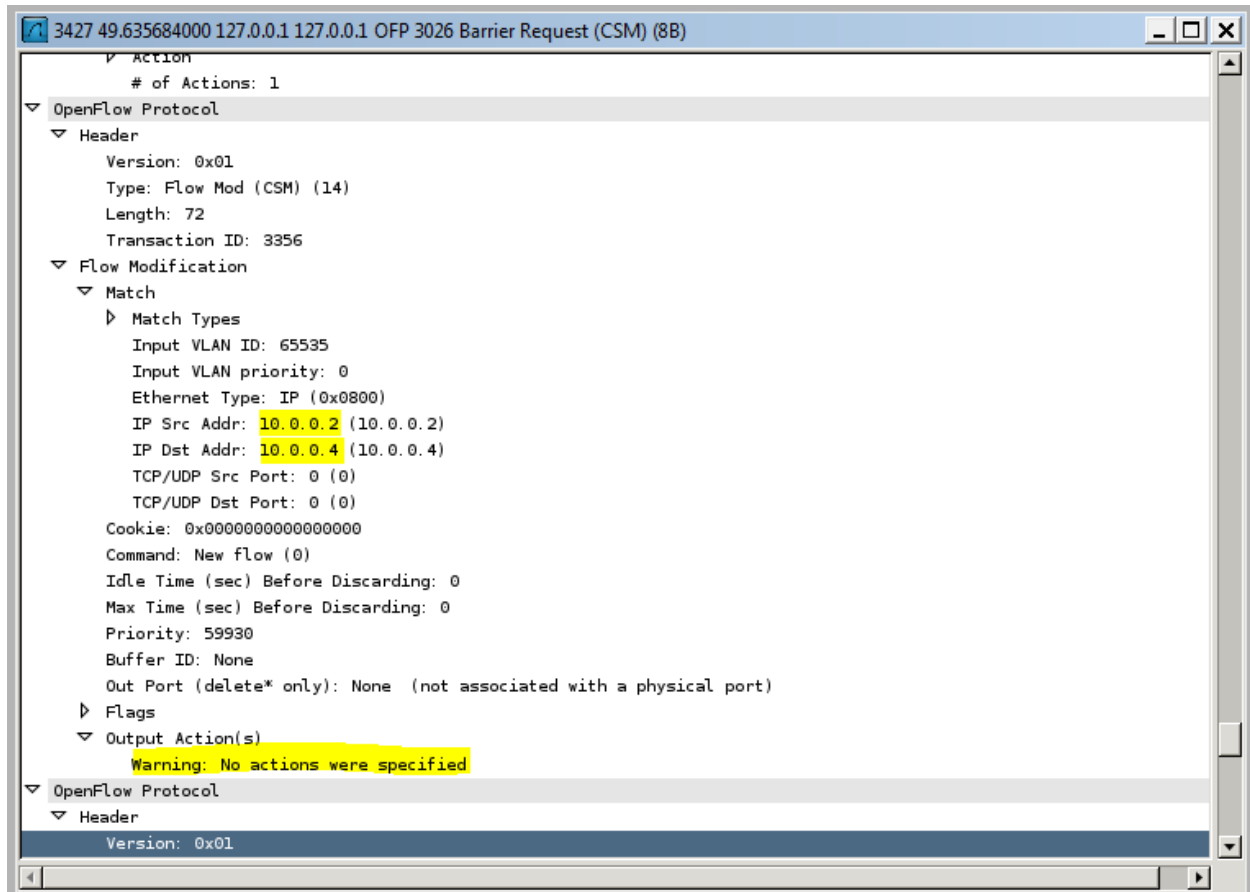


Figure: 3.2.4 Wireshark Captured Open Flow packets

H1 can still reach h3.

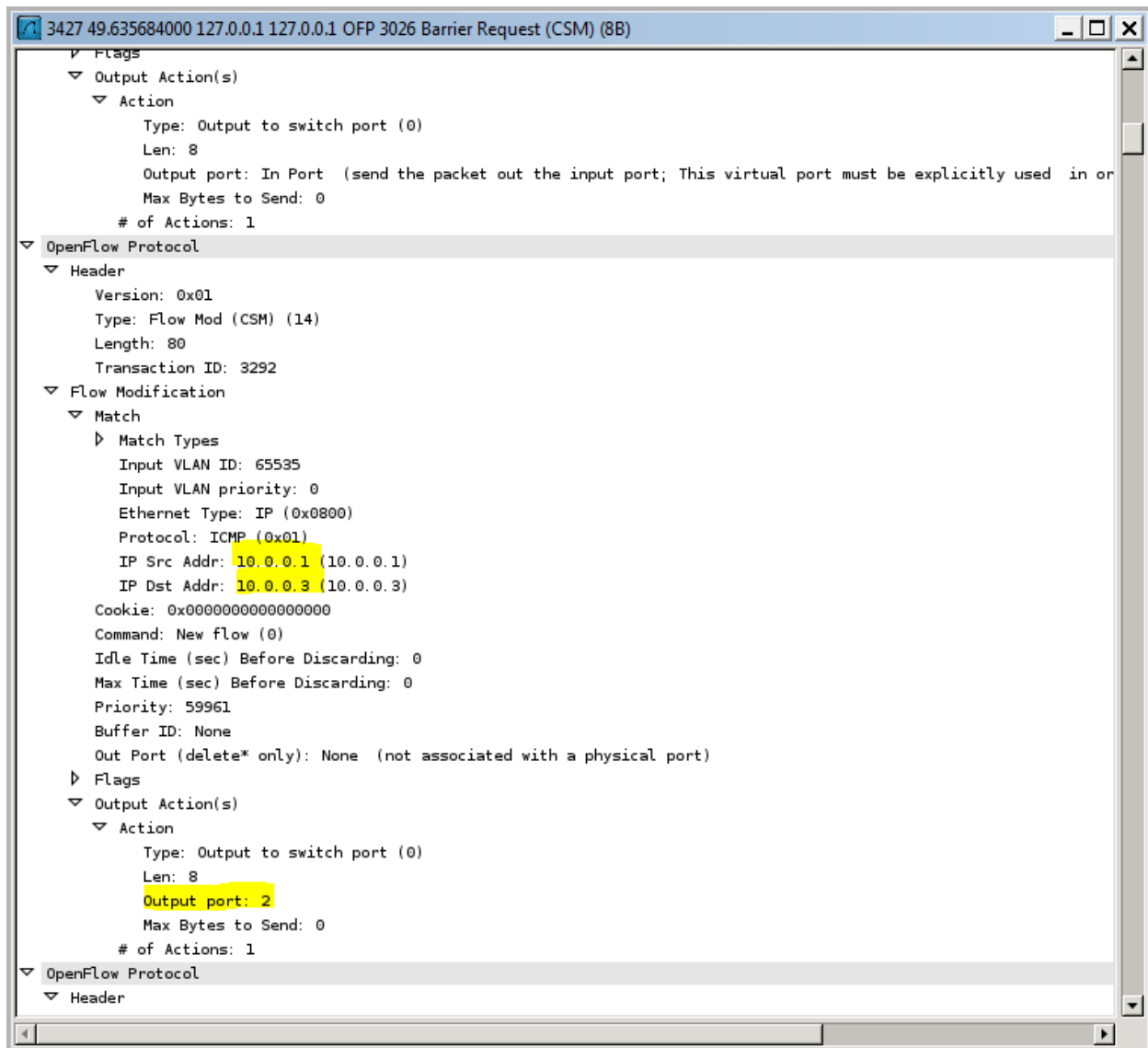


Figure: 3.2.5 Wireshark Captured Open Flow packets

## 4. FINDINGS

This paper demonstrates that Pyretic is a powerful and concise SDN programming language. It has the following main features:

- using function to realize network policy
- using policy composition to build one module on top of another
- flexible packet header

These features make Pyretic an ideal language to build SDN Northbound applications. With its simple syntax, programmers can write a powerful application using less effort compared with other methods. It also reduces the risk of the policies conflicts with each other and ensures that it easy to debug for programmers. Furthermore, with flexible packet header feature, new virtual fields can be added to the packet header. These new information can be used to manipulate network paths of packets, ensure QOS, deploy load balancing, etc. For example, a programmer may want a packet to follow a special path through a network. After adding the packet with a new head field, it makes it easier to forward the packet to destination based on the new field in the path.

This paper also shows that it is possible to build a firewall module using pyretic. The pyretic controller with a firewall module can replace expensive traditional hardware firewalls.

Although pyretic used POX as a runtime system to handle underlay network behaviors, this implementation proved that the firewall module can perform well in manipulating network traffic compared with a traditional firewall.

## 5. CONCLUSION AND FUTURE WORK

In this paper, I have demonstrated several pyretic firewall modules. With the development of Pyretic modules, it is safe to say Pyretic is capable of creating modular SDN application. Pyretic as a comprehensive framework is one of a solutions for developing complex SDN firewall. In addition, the current version of Pyretic can support a POX SDN controller as run-time system. With the contribution from Pyretic developers, the Pyretic runtime could run on top of any Openflow controller platform in the future. Currently, Pyretic only provides very limited access to packet-level information in the controller. Therefore, there is improvement needed to support statefull packet inspection in Pyretic. It also need to point out that this implementation scope is limited by pyretic system. Some of advanced firewall functions cannot be simulated by the firewall module, such as anti-virus, malware detection, VPN tunnel, etc. There are still a lots of work to do for pyretic contributors in the future. For future work, it is hoped to develop robust security modules in the pyretic framework to support the advanced firewall functions for SDNs.

## APPENDIX

### Pyretic Code

#### switch.py

**# Switch module is used to select network path for allowed packets**

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

h1=IPAddr('10.0.0.1')
H2=IPAddr('10.0.0.2')
h3=IPAddr('10.0.0.3')
H4=IPAddr('10.0.0.4')

def switch():

    return(
        (match(dstip=h1)>>((match(switch=4)>>fwd(1))+(match(switch=3)>>fwd(1))))+
        (match(dstip=H2)>>((match(switch=4)>>fwd(1))+(match(switch=3)>>fwd(2))))+
        (match(dstip=h3)>>((match(switch=4)>>fwd(2))+(match(switch=3)>>fwd(3))))+
        (match(dstip=H4)>>((match(switch=4)>>fwd(3))+(match(switch=3)>>fwd(3)))))
```

#### fw.py

**# fw module imports acl and dos modules. Pyretic run this module to start the controller.**

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.modules.dos import Dos
from pyretic.modules.acl import Access_control

def main():

    print ('Firewall is runing')
    acl=Access_control()
    return Dos(acl)
```

#### dos.py

**# dos monitor real time statistics from switch and make changes on self.policy when the number of packets per second exceeds the threshold.**

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

```
class Dos(DynamicPolicy):
```

```
    def __init__(self,acl):
```

```
        super(Dos,self).__init__()
        self.query_pkt= match(srcip=IPAddr('10.0.0.2'),dstip=IPAddr('10.0.0.4'),protocol=1)
        self.r=0
        self.launch()
```

```
    def start_query(self):
```

```
        self.query = count_packets(1,['srcip','dstip'])
        self.query.register_callback(self.query_action)
```

```
    def update_policy(self):
```

```
        self.policy=self.acl + (self.query_pkt >> self.query)
        #print ('dos.self.policy',self.policy)
```

```
    def query_action(self,stats):
```

```
        #print('dos.stats-->',stats)
        for (p,c) in stats.items():
```

```
            print ('*****p=', p, '-->c=',c,'*****')
            if c>20 and self.r==0:
```

```
                del self.acl.rules_p[str(p.map['srcip']),str(p.map['dstip'])]
```

```
                del self.acl.rules_t[str(p.map['srcip']),str(p.map['dstip'])]
                del self.acl.revers_rules_t[str(p.map['dstip']),str(p.map['srcip'])]
                del self.acl.revers_rules_p[str(p.map['dstip']),str(p.map['srcip'])]
                #print ('self.acl.rules_p',self.acl.rules_p)
```

```

        #print ('self.acl.rules_t',self.acl.rules_t)
        #print ('self.acl.revers_rules_t',self.acl.revers_rules_t)
        #print ('self.acl.revers_rules_p',self.acl.revers_rules_p)
        self.r=1
        self.acl.whitelist=self.acl.allow(self.acl.rules_p,self.acl.rules_t)|
self.acl.allow_revers(self.acl.revers_rules_p,self.acl.revers_rules_t)
        self.acl.whitelist=if_(match(ethtype=2054),passthrough, self.acl.whitelist)
        self.acl.update_policy()
        self.update_policy()
    else:
        self.acl.update_policy()
        self.update_policy()

def launch(self):

    self.policy=self.acl + (self.query_pkt >> self.query)

```

## acl.py

**#acl reads while list from whitelist.csv and forward allowed traffic to switch module**

```

import os
import csv
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
from pyretic.modules.switch import switch
wl_file = "%s/pyretic/pyretic/examples/whitelist.csv" % os.environ[ 'HOME' ]

class Access_control(DynamicPolicy):

    def __init__(self):

        super(Access_control,self).__init__()
        self.switch=switch()
        self.rules_p={}
        self.rules_t={}
        self.revers_rules_p={}
        self.revers_rules_t={}
        self.openfile()
        self.whitelist=if_(match(ethtype=2054),passthrough,drop)

```

```
self.run()
```

```
def openfile(self):
```

```
    self.f=open(wl_file, 'r')
    self.f.readline()
    self.lines=csv.reader(self.f)
    for s,d,p,t in self.lines:
        if t=='6':
            self.rules_p[(s,d)]=p
        else:
            self.rules_t[(s,d)]=t

    #print ('self.rules_p', self.rules_p)
    #print ('self.rules_t', self.rules_t)
```

```
def allow(self,arg1,arg2):
```

```
    p=union([match(srcip=s,dstip=d,dstport=int(arg1[s,d])) for (s,d) in arg1.keys()])
    t=union([match(srcip=s,dstip=d,protocol=int(arg2[s,d])) for (s,d) in arg2.keys()])
    pa=p | t
    return pa
```

```
def allow_revers(self,arg1,arg2):
```

```
    p=union([match(srcip=s,dstip=d,srcport=int(arg1[s,d])) for (s,d) in arg1.keys()])
    t=union([match(srcip=s,dstip=d,protocol=int(arg2[s,d])) for (s,d) in arg2.keys()])
    pa=p | t
    return pa
```

```
def start_query(self):
```

```
    self.query =packets(1,['srcip','dstip'])
    self.query.register_callback(self.query_action)
```

```
def update_policy(self):
```

```
    self.policy= (self.whitelist >> self.router) + self.query

    #print ('Update_policy')
```

```

def query_action(self, p):

    #print ('*****acl_p=', p, '*****')
    if (str(p['srcip']),str(p['dstip'])) in self.rules_p.keys():

        self.revers_rules_p[(str(p['dstip']),str(p['srcip']))]=self.rules_p[(str(p['srcip']),str(
p['dstip']))]

        self.revers_rules_t[(str(p['dstip']),str(p['srcip']))]=self.rules_t[(str(p['srcip']),str(
p['dstip']))]
        print ('Access allowed from %s to %s' % (p['srcip'],p['dstip']))
        print ('Access allowed from %s to %s' % (p['dstip'],p['srcip']))
        self.whitelist=self.allow(self.rules_p,self.rules_t)
        self.allow_revers(self.revers_rules_p,self.revers_rules_t)
        self.whitelist=if_(self.whitelist, passthrough,drop)
        self.whitelist=if_(match(ethtype=2054),passthrough, self.whitelist)
        self.update_policy()
    else:
        if (str(p['srcip']),str(p['dstip'])) not in self.revers_rules_p.keys():
            print ('Access denied from %s to %s' % (p['srcip'],p['dstip']))

def run(self):
    self.start_query()
    self.policy= (self.whitelist >> self.switch )+ self.query

```

## Whitelist.csv

#pre-configured white list, read by acl

```

scrip,dstip,port,protocol-type
10.0.0.1,10.0.0.3,23,6
10.0.0.2,10.0.0.4,80,6
10.0.0.1,10.0.0.3,0,1
10.0.0.2,10.0.0.4,0,1

```

## Reference

- [1] Pyretic Reference Implementation  
<http://www.frenetic-lang.org/pyretic>
- [2] Modular SDN Programming with Pyretic  
<http://frenetic-lang.org/publications/pyretic-login13.pdf>
- [3] Mininet official website  
<http://mininet.org/overview/>
- [4] Programming SDNs: Module 6.4: Pyretic  
<http://vk5tu.livejournal.com/41153.html>
- [5] The Python Tutorial  
<https://docs.python.org/2/tutorial/>
- [6] Software-Defined Networking (SDN) Definition  
<https://www.opennetworking.org/sdn-resources/sdn-definition>
- [7] The Pyretic language and runtime system  
<https://github.com/frenetic-lang/pyretic>
- [8] Composing Software-Defined Networks  
Joshua Reich, Nate Foster, Jennifer Rexford, David Walker  
<ps://www.usenix.org/system/files/conference/nsdi13/nsdi13-final232.pdf>
- [9] Opendaylight Use Cases  
<https://www.opendaylight.org/example-use-cases>
- [10] G. Bianchi, M. Bonola, A. Capone, and C. Cascone.  
OpenState: programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Computer Communication Review, 2014.
- [11] FOSTER, N., HARRISON, R., FREEDMAN, M. J., MONSANTO, C., REXFORD, J., STORY, A., AND WALKER, D. Frenetic: A Network Programming Language. In *ACM ICFP* (Sep. 2011), pp. 279–291.
- [12] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn and Ziming Zhao  
FLOWGUARD: Building Robust Firewalls for Software-Defined Networks  
<http://people.cs.clemson.edu/~hongxih/papers/HotSDN2014.pdf>
- [13] KELLER, E., AND REXFORD, J. The 'Platform as a Service' Model for Networking. In *IMN/WREN* (Apr. 2010).
- [14] NICIRA. It's time to virtualize the network, 2012.  
<http://nicira.com/en/network-virtualization-platform>

[15] S. Shirali-Shahreza and Y. Ganjali. Flexam  
Flexible sampling extension for monitoring and security applications in openflow.

[16] J. Wang, Y. Wang, H. Hu, Q. Sun, H. Shi, and L. Zeng.  
Towards a security-enhanced firewall application for openflow networks. In  
Cyberspace Safety and Security, 2013.