Building Firewall over the Software-Defined Network Controller

Michelle Suh, Sae Hyong Park, Byungjoon Lee, Sunhee Yang

SDN Research Section, ETRI (Electronics and Telecommunications Research Institute), Korea

{michsuh1, justin.labry, byungjoon.lee}@ gmail.com, shyang@etri.re.kr

***Abstract*— Many have recognized the need to restructure the current internetwork into a much more dynamic networking environment. It is difficult for today's inflexible infrastructure to cope with the fast changing demands of the users. As a result, Software-Defined Network (SDN) was introduced around 2005 to transform today's network to have centralized management, rapid innovation, and programmability by decoupling the control and data planes. This study focuses on developing a firewall application that runs over an OpenFlow-based SDN controller to show that most of the firewall functionalities are able to be built on software, without the aid of a dedicated hardware. Among many OpenFlow controllers that already exist for the public, we have chosen POX written in Python for the experiment; and to create the SDN network topology, we have used VirtualBox and Mininet. In this study, we cover the implementation detail of our firewall application, as well as the experimentation result.**

***Keywords*— SDN, Openflow, Firewall, Controller, IRIS**

I. INTRODUCTION

The current network infrastructure known as the 'Internet' has settled for more than a decade and has rooted deeply in our society. However, enterprises and carriers are starting to realize the limitations of current state with the rapidly evolving network technologies and the growing demands of the users; the current network is too inflexible compared to the much dynamic server environment. Right now, most of the forwarding decisions are determined at the routers based on packet headers. And in order to switch around virtual machines or hardware devices, vendors must reconfigure multiple routers, switches, firewalls, etc. Simply put, change in network is too much of a hassle with the need to touch each hardware entity, which also makes it time consuming and cost inefficient.

To overcome this limitation, Software-Defined Network (SDN) was suggested around 2005. SDN is a newly rising networking concept that decouples the control plane and data plane to centralize network intelligence in the controller. The idea of separating control and data plane rose with the intention of developing each part independent from one another, so that the software will not be constrained by the limitation of the hardware. And users can control from high-level software programs to make it easier to manage the logic [10]. In SDN, data plane entities are 'dummy' switching devices that make reactive forwarding decisions by querying the control plane. This has unveiled the unprecedented possibility to program the data plane with network-controlling business logics from the control plane. Thus, a lot of people are expecting that SDN will reduce network management cost and enhance programmability, advancing the network to be easily configured and managed [3]. One of the examples where SDN is most useful is Data Center. Data Centers carry tens of thousands of virtual machines that migrate over the underlying topology. Instead of having to configure each switch in accordance to the new virtual network, they can employ programmable switches that the central database can control [5].

The most important component of SDN is a communication protocol between the control plane and the data plane. OpenFlow Protocol [8] has been introduced for that purpose, supported by a world-wide SDN organization called Open Networking Foundation (ONF). Now the OpenFlow switches acquire flexible packet handling by inquiring the controller on how to process the incoming data.

Currently, there are quite a few public SDN controllers implementing OpenFlow, such as NOX, POX, Ryu, and Floodlight [3]. Furthermore, there is a new controller named IRIS currently being developed at Electronics and Telecommunication Research Institute (ETRI) in Daejeon, South Korea. It is created with an attempt to overcome some of the limitations of Floodlight, and to improve the overall speed and scalability. The same OpenFlow Protocol can be applied for various purposes such as forwarding, flow switching, and firewall [7]. And the work specific to this paper involves creating a firewall over the SDN controllers using OpenFlow. Firewall is essential in keeping the network safe from outside attacks. Some of the current SDN controllers do carry firewalls along with their main forwarding faculties, but they either lack user interface or effectiveness in their functions. Therefore, with the current SDN controllers, the service providers would have to buy external firewall hardware from the vendors like Cisco and Juniper. The firewall presented in this paper is coded in the hopes of creating a sufficient logic and, importantly, a well-designed user interface. A well written user interface in SDN firewall can adopt all the benefits of Software-Defined Network. And being able to switch around the rule priorities as the user wishes is a very powerful quality that a firewall can acquire.

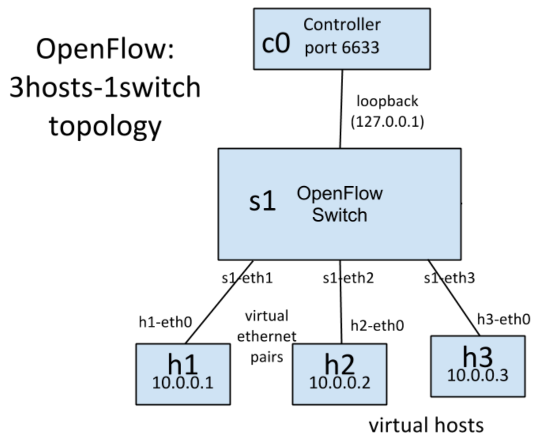


Figure 1. SDN network topology used for testing

II. Approach

There were two approaches considered in implementing the firewall: a) pre-installing the rules onto the switch's flow table and b) handling the packets directly as they come in. We chose to handle the incoming packets directly because of the flexibility in management. One downside of this method is that too many packets can be delivered to the controller and take up a large portion of its resources; it is a lot more efficient to block unnecessary packets at the switch level. To cope with this issue, the user can also decide to install a 'deny' flow modification on the switch to continue dropping similar packets for a certain time period.
The logic of this firewall is as follows: each packet headers are checked against the firewall rule from highest to lowest priority, and performs specified action once matching fields are found in the rule. Any unmatched packets are dropped. Installing firewall rules are possible from an external entity through a text-based user interface.

III. Implementation

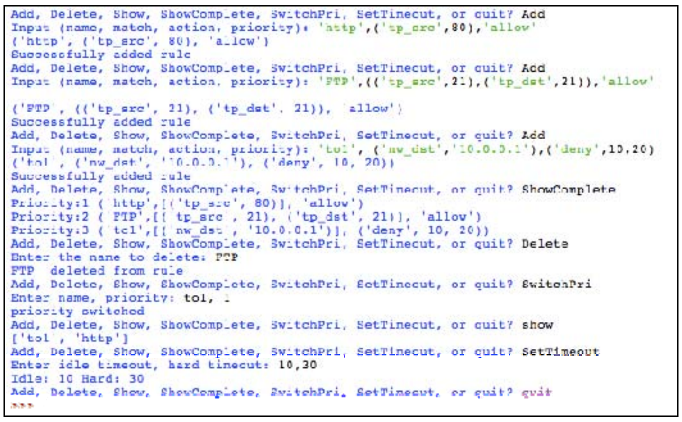
In order to test the functionality of this firewall, the following programs were used:

1. *VirtualBox - provides an environment for virtual network to be formed.*

2. *Mininet - provides virtual SDN network topology.*

3. *POX - SDN controller.*

The firewall was tested by attaching to a learning switch POX controller [9]. Learning switch uses a simple forwarding algorithm that updates the OpenFlow Switch by registering the packets' port, MAC, and IP addresses as they come in. The topology has a remote controller attached to an OpenFlow Switch, which connects three or four hosts (Figure 1).The Ethernet address of each host corresponds to its IP address for simplicity, meaning h1 has 00:00:00:00:00:01, h2 has 00:00:00:00:00:02, and so on. As mentioned above, the main focus in improving SDN firewalls is to implement a simply, understandable, and


```
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? Add
Input (name, match, action, priority): 'http',('tp_src',80),'allow'
('http', ('tp_src', 80), 'allow')
Successfully added rule
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? Add
Input (name, match, action, priority): 'FTP',(('tp_src',21),('tp_dst',21)),'allow'

('FTP', (('tp_src', 21), ('tp_dst', 21)), 'allow')
Successfully added rule
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? Add
Input (name, match, action, priority): 'tol', ('nw_dst','10.0.0.1'),('deny',10,20)
('tol', ('nw_dst', '10.0.0.1'), ('deny', 10, 20))
Successfully added rule
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? ShowComplete
Priority:1 ('http',[('tp_src', 80)], 'allow')
Priority:2 ('FTP',[('tp_src', 21), ('tp_dst', 21)], 'allow')
Priority:3 ('tol',[('nw_dst', '10.0.0.1')], ('deny', 10, 20))
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? Delete
Enter the name to delete: FTP
FTP  deleted from rule
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? SwitchPri
Enter name, priority: tol, 1
priority switched
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? show
['tol', 'http']
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? SetTimeout
Enter idle timeout, hard timeout: 10,30
Idle: 10 Hard: 30
Add, Delete, Show, ShowComplete, SwitchPri, SetTimeout, or quit? quit
>>>
```

Figure 2. A demonstration of the user interface

functional user interface. The firewall is made so that the user, most likely some sort of administrator, can manage the rules outside the controller. We tested the design through a server-client thread (Figure 2). The user can input exactly how the firewall asks in each stage. First, the UI gives six options to choose in managing firewall: Add, Delete, Show, ShowComplete, SwitchPri, and SetTimeout. The user should write the exact word as it appears for the six options, though it is case insensitive. The names give quite a good sense of their functions, but each will be explained in detail.

1) *SDN Firewall Commands*

A. Add:

this command takes in the parameters and add to the list of rules in a dictionary form. This command has following functions:

- if repeated name appears, it warns the user and doesn't add.
- the rule list is in the order of priority, and user can decide priority for each rule. If the user doesn't specify a priority, it is added to the end of the list.
- Timeout is an integer in unit of seconds
- Priority is optional. The highest priority that user can assign is one. When the indicated priority is already taken, the new rule gets placed at that priority and the rest is pushed back one priority.

This command has following syntax:

- input in the form of (name, match, action, priority (optional)
- anything non-integer are added with the quotation marks
- When putting multiple match fields, All the match pair need to be put inside a parenthesis (see Figure 2)
- Four options for the action field are: 'allow', 'deny', ('deny', timeout): when idle & hard have equal timeout. This installs the rule to switch that rejects all similar

packets for the time specified. ('deny', idle timeout, hard timeout)

B. Delete:

this command takes the rule name as an input, and deletes from the rule. This command has following functions:

- If indicated name doesn't exist, it warns the user

This command has following syntax:

- The name is case-sensitive, and does not need to be in quotation marks

C. Show:

this commands show the name of the rules. This command has following functions:

- The names of the rules are listed in the order of priority

This command has following syntax:

- No additional input necessary

D. ShowComplete:

this command Shows complete entry of the rules. This command has following functions:

- List each rule in the order of priority, name, match, allow

This command has following syntax:

- No additional input necessary

E. SwitchPri:

this command takes name and new priority as input, and switches list order. This command has following functions:

- The indicated rule gets placed at the newly assigned priority, and the rest of the rules gets pushed back one.

This command has following syntax:

- Input should be in order of name, priority separated by a comma.
- Name does not need to be in quotation marks

F. setTimeout:

this command sets general timeout to drop similar packets in the future. This command has following functions:

- Checks that inputs are valid integers

This command has following syntax:

- Input in the order of idle timeout, hard timeout separated by a comma

2) *SDN Firewall Attributes*

A. in_port:

this attribute denotes switch port number the packet arrived on. This attribute can be used as following example:

- Int: 30

B. dl_src:

this attribute indicates Ethernet source address. This attribute can be used as following example:

- String: '00:00:00:00:00:01'

C. dl_dst:

this attribute indicates Ethernet destination address. This attribute can be used as following example:

- String: '00:00:00:00:00:02'

D. dl_vlan:

this attribute indicates VLAN ID. This attribute can be used as following example:

- Int: 2

E. dl_vlan_pcp:

this attribute indicates VLAN priority. This attribute can be used as following example:

- Int: 0

F. dl_type:

this attribute indicates Ethertype. This attribute can be used as following example:

- IP_TYPE, ARP_TYPE, RARP_TYPE, VLAN_TYPE, LLDP_TYPE, JUMBO_TYPE, QINQ_TYPE

G. nw_tos:

this attribute indicates IP TOS/DS bits. This attribute can be used as following example:

- Int: 0

H. nw_proto:

this attribute indicates IP protocol. This attribute can be used as following example:

- ICMP_PROTOCOL, TCP_PROTOCOL, UDP_PROTOCOL

I. nw_src:

this attribute indicates IP source address. This attribute can be used as following example:

- String: '10.0.0.1'

J. nw_dst:

this attribute indicates IP destination address. This attribute can be used as following example:

- String: '10.0.0.2'

K. tp_src:

this attribute indicates TCP/UDP source port. This attribute can be used as following example:

- Int: 80

L. tp_dst:

this attribute indicates TCP/UDP destination port. This attribute can be used as following example:

- Int: 333

There are twelve match fields for OpenFlow Protocol that the users can specify [1]. As shown above, this firewall has seven 'dl_type' (ethertypes) to match the packet to, which is about half the amount of the total ethertype that POX has given an option to match with. Future developers can easily add more options to 'dl_type' by referring to the POX documentation [1] and codes [9]. This goes the same for 'nw_proto' (IP protocol). Other font types may be used if needed for special purposes.

IV. Experimental Results

The three tests below demonstrate specific Ethernet and IP addresses to pass or reject using PING between the three hosts. The rules are listed in order of priority in dictionary form, which is how it is stored in firewall. In the (key: value) structure, key is the field name and value the match element.

We first allowed all ARP requests as first priority, so that the hosts can understand their surrounding topology. Then any data transmissions to and from the IP address '10.0.0.1' (host1) are allowed, as well as the transmissions to and from the Ethernet address '00:00:00:00:00:02' (host 2).

A. Firewall rule:

- 'name': 'ARP', 'dl_type': 'ARP_TYPE', 'action': 'allow'
- 'name': 'to2', 'dl_dst': '00:00:00:00:00:02', 'action': 'allow'
- 'name': 'from2', 'dl_src': '00:00:00:00:00:02', 'action': 'allow'
- 'name': 'to1', 'nw_dst': '10.0.0.1', 'action': 'allow'
- 'name': 'from1', 'nw_src': '10.0.0.1', 'action': 'allow'

As expected, all PING are successful even without allowing host3 packet transmission since all interaction with h1 and h2 are permitted.

To check that 'deny' action works properly, the same exact rule as the first test was employed, except for rejecting any packets between host1 and host3. The 'deny' rule was placed before the 'allow' rules to make sure that denying has a higher priority than allowing. The added rule is in #2.

B. Firewall rule:

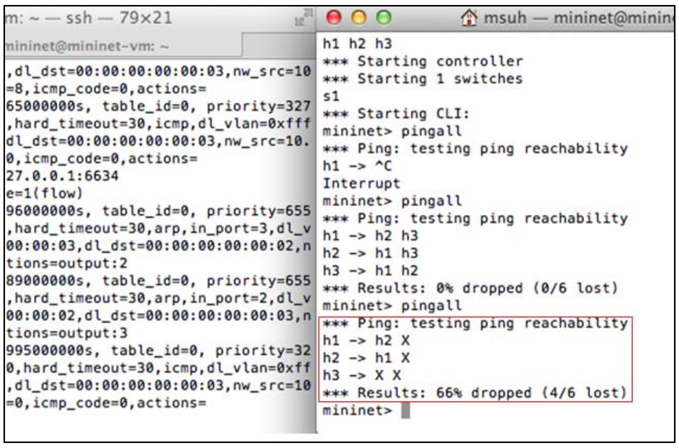
- 'name': 'ARP', 'dl_type': 'ARP_TYPE', 'action': 'allow'
- 'name': '1to3', 'nw_src': '10.0.0.1', 'nw_dst': '10.0.0.3', 'action': 'deny'
- 'name': 'to2', 'dl_dst': '00:00:00:00:00:02', 'action': 'allow'
- 'name': 'from2', 'dl_src': '00:00:00:00:00:02', 'action': 'allow'
- 'name': 'to1', 'nw_dst': '10.0.0.1', 'action': 'allow'
- 'name': 'from1', 'nw_src': '10.0.0.1', 'action': 'allow'

This test demonstrates rejecting a packet and installing a rule onto the OpenFlow switch to continue dropping similar packets for a certain amount of time. The firewall specifies to install the switch rule to drop all packets to host3 with idle and hard timeout of 20 and 30 seconds (rule #2). Then all other PING are allowed to pass (rule #3).

C. Firewall rule:

- 'name':'ARP', 'dl_type':'ARP_TYPE', 'action':'allow'
- 'name':'to3', 'dl_dst':'00:00:00:00:00:03', 'action':('deny',20,30)
- 'name':'PING', 'nw_proto':'ICMP_PROTOCOL', 'action':'allow'

The portion in the red box on the right window shows the expected PING result (Figure 3). And the dump-flows on the left window demonstrate that the switch installed the rules successfully as firewall demanded. The green box shows the proper 'deny' rule, which performs no action, idle_timeout of 20s, and hard_timeout of 30s when 'dl_dst' equals 00:00:00:00:00:03. The other switch rules with specified



```
m: ~ — ssh — 79×21
mininet@mininet-vm: ~
,dl_dst=00:00:00:00:00:03,nw_src=10
=8,icmp_code=0,actions=
65000000s, table_id=0, priority=327
,hard_timeout=30,icmp,dl_vlan=0xfff
dl_dst=00:00:00:00:00:03,nw_src=10.
0,icmp_code=0,actions=
27.0.0.1:6634
e=1(flow)
96000000s, table_id=0, priority=655
,hard_timeout=30,arp,in_port=3,dl_v
00:00:03,dl_dst=00:00:00:00:00:02,n
tions=output:2
89000000s, table_id=0, priority=655
,hard_timeout=30,arp,in_port=2,dl_v
00:00:02,dl_dst=00:00:00:00:00:03,n
tions=output:3
995000000s, table_id=0, priority=32
0,hard_timeout=30,icmp,dl_vlan=0xff
,dl_dst=00:00:00:00:00:03,nw_src=10
=0,icmp_code=0,actions=
```

```
msuh — mininet@minin
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> ^C
Interrupt
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (0/6 lost)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (4/6 lost)
mininet>
```

Figure 3. PING test result denying packets between host1 and host3

action ports are from the packets that were passed onto the controller to forward.

Testing TCP connection between hosts was trickier than simply using PING because neither particular TCP test between hosts nor ways for a host to connect to the 'real' internet exist. Thus, we used a simple server-client thread that imitates the firewall user interface. Host1 runs a program carrying a server port that the clients can connect to. Host2 acts as a client and runs a program that tries to bind to host1's port.

The contents of h1 and h2 interaction are irrelevant in what rules firewall actually have in these examples. This is just an attempt to show that the firewall can properly manage TCP port connections. The two cases below demonstrate when the server port was denied and allowed. h1 ran the server thread with server port number 3335, and h2 bind to the server port as a client. The difference in rule is marked with bolded action.

D. Server port 3335 allowed

- 'name': 'ARP', 'dl_type': 'ARP_TYPE', 'action': 'allow'
- 'name': 'thread', 'tp_dst': 3335, 'action': 'allow'
- 'name': 'thread', 'tp_src': 3335, 'action': 'allow'
- 'name': 'from1', 'dl_src': '00:00:00:00:00:01', 'action': 'allow'
- 'name': 'to1', 'dl_dst': '00:00:00:00:00:01', 'action': 'allow'

E. Server port 3335 denied

- 'name': 'ARP', 'dl_type': 'ARP_TYPE', 'action': 'allow'
- 'name': 'thread', 'tp_dst': 3335, 'action': 'deny'
- 'name': 'thread', 'tp_src': 3335, 'action': 'deny'
- 'name': 'from1', 'dl_src': '00:00:00:00:00:01', 'action': 'allow'
- 'name': 'to1', 'dl_dst': '00:00:00:00:00:01', 'action': 'allow'

V. CONCLUSIONS

The popularity of SDN in the IT world today is demonstrated by the numerous startups that proliferate in this area. Though there exist outstanding firewalls in the market, it is quite costly for companies to install numerous firewall hardware across the entire network to preserve high security. Also as mentioned earlier, replacing a firewall is a serious pain – from physically replace the firewall, reconfiguring each device related to the firewall to troubleshooting [4]. Besides the inconvenience, one wrong turn can put the entire network at risk. Thus, SDN is not only revolutionary in making the control flexible and manageable, but also for firewalls to achieve programmability by separating the firewall hardware and the control software. An OpenFlow-based firewall with a straightforward UI that integrates priority switching can bring another wave of innovation in the Internet world.

Currently, this design only looks at the packet header fields to determine the action. Perhaps, future developers can further improve this logic by incorporating SDN capacities to improve security by observing the entire network flow and efficiently block the network attacks in the early stage without having to perform deep packet inspection.

ACKNOWLEDGMENT

This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013

REFERENCES

[1] Al-Shabibi, A. & McCauley, M. (2013, August 4). POX Wiki. Retrieved from Stanford University, OpenFlow at Stanford : https://OpenFlow.stanford.edu/display/ONL/POX+

[2] JBavier. & Andy. et al. (2006). In VINI vertias: realistic and controlled network experimentation. ACM SIGCOMM Computer Communication Review, 36(4).

[3] Feamster, N. (2013). Software Defined Networking. Retrieved from coursera: https://class.coursera.org/sdn-001

[4] Ferro, G. (2013, Mar 18). SDN Use Case: Firewall Migration in the Enterprise. Retrieve from Ethereal Mind: etherealmind.com/sdn-use-case-firewall-migration-in-the-enterprise.

[5] Gashinsky, I. (2012 April 17). SDN in Warehouse-Scale Data Centers. Retrieved from: http://goo.gl/u9bWq

[6] Mininet Team. (2013). Retrieved from Mininet: mininet.org

[7] *OpenFlow Overview. (2010). Retrieved from Stanford University, Department of Computer Science: Yuba.stanford.edu/cs244wiki*

[8] OpenFlow Switch Specification: Version 1.1.0 Implemented. (2011, Feb 28). Open Networking Foundation. Retrieved from https://www.opennetworking.org/

[9] POX. (2013). Retrieved from Open Source control platforms for SDN: NOXRepo.org

[10] Software-Defined Networking: The New Norm for Networks. (2013, April 13). Open Networking Foundation. Retrieved from https://www.opennetworking.org/

[11] *Stevens, W.R. (1994). TCP/IP Illustrated, Volume1: The Protocols. Reading, MA: Addison-Wesley Publishing Company.*

[12] Van der merwe & Jacobus E., et al. (1998). The tempest-a practical framework for network programmability. Network, IEEE 12.3, 20-28.tevens.