Implementing Openflow Based Distributed Firewall

Sukhveer Kaur, Karamjeet Kaur
Department Of Computer Science and Applications
AD College, Dharamkot
Moga, India
{bhullarsukh96, bhullar1991}@gmail.com

Vipin Gupta
U-Net Solutions
Moga, India
vipin2411@gmail.com

***Abstract*— SDN is an emerging technology which is going to drive next generation networks. Lot of companies and organizations has started using SDN applications. It is giving network administrators the flexibility in implementing their networks. But at the same time, it is bringing new security issues. To secure SDN networks, we need strong firewall application. Already some firewall applications are there but they suffer from certain shortcomings. One of the main drawbacks of existing firewall solutions is that they suffer from single point of failure due to their centralized nature and overloading of rules in single device. Other drawback of existing firewall is that they are mostly layer 2 firewalls. In this paper, we are implementing Distributed Firewall where every OpenFlow switch in a network can acts as a firewall. Plus this firewall will be capable of handling TCP, UDP and ICMP Traffic. We have tested this firewall using Mininet Emulator installed in Ubuntu 14.04 linux installed under VirtualBox virtualization solution. We are using python based POX controller. This work is extension of our earlier work on programmable firewalls.**

Keywords—Mininet; POX; Software Defined Networking; Firewall

I. INTRODUCTION

In Traditional networks, it is very difficult to dynamically modify the routers, switches, load balancers, IDS, IPS configurations according to the requirements of the organization. But SDN is changing all this. SDN decouples the forwarding plane from the control plane. SDN allows centralized controller to dynamically manage all the devices [1]. In traditional networks, each device has to be configured individually. Control plane and forward plane communicates using OpenFlow protocol as shown in Fig. 1.

The main function of the SDN is to determine what to do with the packet that has been received by OpenFlow switch by consulting its flow entry table [2, 3]. When the packets arrive at switch, packet header fields are matched against flow table entries. It there is a match, then action is performed according to specified in the flow entry. If there is no match, then packet will sent to the controller according to flow miss entry. This is also called Packet_In message. Now controller will decide what to do with the packet according to its application logic [4]. It may then instruct the switch to forward the packet or may add flow entry into switch flow table so that similar types of packets can be handled quickly.

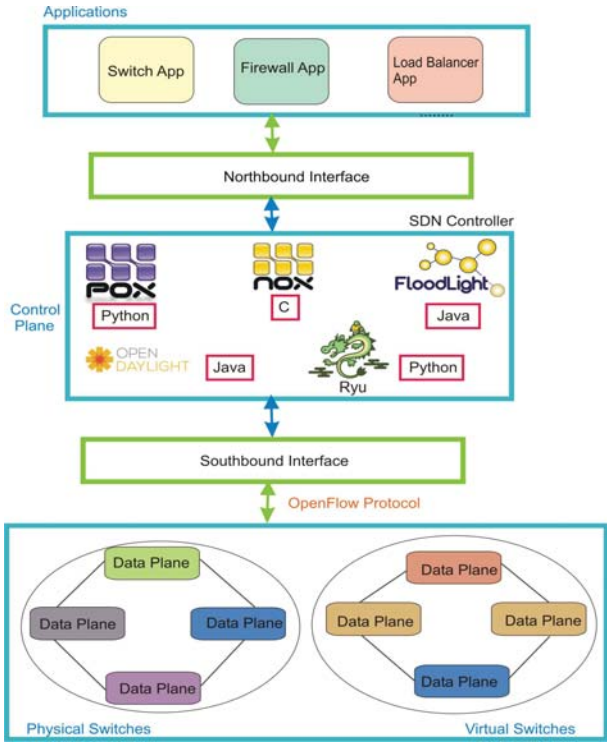

Fig. 1. Software defined networking architecture

II. RELATED WORK

Firewalls are primarily used for restricting unauthorized access. Firewalls can be of various types such as hardware/software firewalls, stateful/stateless firewalls, packet filtering/application level or centralized/distributed types. In our case we are implementing stateless distributed packet filtering firewall [5].

Tariq [6] implements a simple centralized Layer 2 firewall based on mac addresses. Kaur [7] implemented centralized firewall solution in which only 1 switch is configured as a firewall. It suffers from single point of failure and rules overload. Pena [8] developed Distributed Firewall that only checks icmp traffic. It does not check other header fields such as TCP, UDP.

III. Experiment Setup

We used Mininet emulator as our testbed software. You can set up any type of experiment labs in Mininet [9]. We used POX Controller based on Python as our SDN Controller which is very useful for research purposes [10, 11]. For our experiment, we created tree topology by using this command

- mn --topo tree,3 --controller remote

It created topology as shown in Fig. 2. Core "s1" is acting as centralized switch. If we are going to use only "s1" switch as firewall, then it will act as centralized firewall. "s1" is connected with distribution switches "s2" and "s5". "s2" is connected with access switches "s3" & "s4". "s5" is connected with access switches "s6" & "s7". Each of the access switches are connected with 2 hosts. Hosts are from h1 to h8 having ip addresses from 10.0.0.1 to 10.0.0.8. We have implemented the following servers or services.

- Web server on port 80 on host h4 by using python based "SimpleHTTPServer"
- UDP based server on port 53 on host h8 by using netcat "nc" utility.

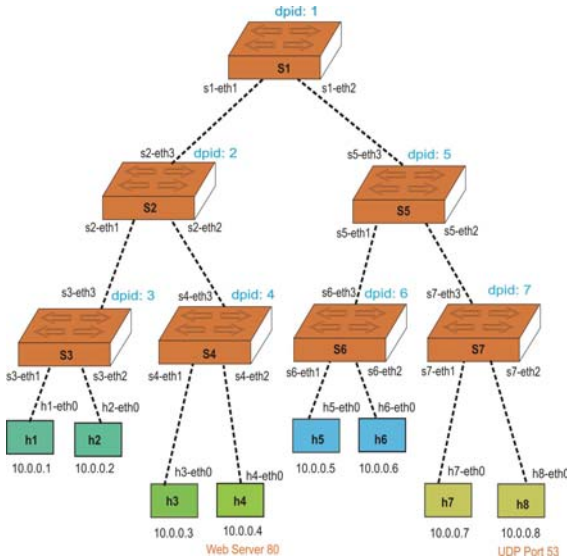

Fig. 2. Network topology

IV. Performance Evaluation

A. Firewall Scenario

We want to test our firewall for the following scenario. Our firewall should be able to perform all the following tasks.

Task 1: h1 should not be able to ping h2.

Task 2: h1 should not be able to ping h7.

Task 3: h1 should not be able access web server running on h4.

Task 4: h7 should not be able to access udp port 53 on h8.

What will happen if we are going to use centralized firewall implementation that means implementing firewall on "s1". It will only be able to control traffic mentioned in task 2 (ping between h1 to h7). It will not be able to control other traffic as required in our case. What if we are going to make "s2" as our firewall. It will able to perform task 2 and task 3 but not task1 and task 4. Why, because traffic from "h1" to "h2" and from "h7" to "h8" does not pass through switch "s2". What if we are going to make "s3" as our firewall? It will able to perform task 1, task 2 and task 3 but not task 4. Why, because traffic from "h7" to "h8" does not pass through switch "s2".

From the above discussion, it is crystal clear that we will not be able to implement all the tasks required by using single switch as a firewall. It means that centralized firewall is not suitable for all the scenarios. What we need is distributed firewall where every switch can acts as a firewall. Our distributed firewall has the capability of installing firewall rules in all the switches. There are many options in which our distributed firewall can handle the above scenario without overloading our switches with large no. of firewall rules.

Option 1: We can enter rules in "s3" and "s7" switches. Rules in "s3" will be able to handle tasks 1, task 2, task 3 and rules in "s7" will be able to handle task 4.

Option 2: We can enter rules in "s3", "s2" and "s7" switches. Rules in "s3" will handle task 1, rules in "s2" will handle task 2, 3 and rules in "s7" will handle task 4.

Option 3: We can enter rules in "s3", "s2", "s1" & "s7". Rules in "s3" will handle task 1, rules in "s2" will handle task 2, rules in "s1" will handle task 3 and rules in "s7" will handle task 4.

In our case, we are going to choose the Option 3, because this will allow us to distribute our rules to many switches to prevent overloading of switches with firewall rules.

Where we are going to place our rules is also dependent upon the granularity of rules. If we want to block only on the basis of source ip address, then it is recommended practice to place the rules in far away switch from where traffic has originated. If we want to block on basis of source ip, destination ip and destination port (that is we exactly know where the packet is going) then it is recommended to place the rules near to switch from where packet has originated. But in our case, we have not taken this factor into account.

The following flow rules were installed on different openflow switches as shown in Table 1.

TABLE I. Firewall Rules

Switch	Source IP	Destination IP	Protocol	Port
S3	10.0.0.1	10.0.0.2	1(ICMP)	-
S1	10.0.0.1	10.0.0.7	1(ICMP)	-
S2	10.0.0.1	10.0.0.4	6(TCP)	80
S7	10.0.0.7	10.0.0.8	17(UDP)	53

In Switch “s3”, we added the rule that block ping (icmp) traffic from host “10.0.0.1” (host h1) to “10.0.0.2” (host h2). In Switch “s1”, we added the rule that block ping (icmp) traffic from host “10.0.0.1” (host h1) to “10.0.0.7” (host h7).In Switch “s2”, we added the rule that block tcp traffic from host “10.0.0.1” (host h1) to “10.0.0.4” (host h4) going towards web server (port 80). In Switch “s7”, we added the rule that block udp traffic from host “10.0.0.7” (host h7) to “10.0.0.8” (host h8) going towards dns server (port 53).

We created simple user interface (UI) for installing, deleting & showing rules as shown in Fig. 3 for our distributed firewall. It shows the following options: A for adding rule, D for deleting the rule, S for showing the rules.

```
Enter the DPID of switch in which you want to insert, delete or show rules:  3
Enter 'A' for adding, 'D' for deleting, 'S' for showing the rules: 'A'
Enter any of the following matching fields
Source IP,Destination IP,Network Protocol,Destination port(UDP or TCP)
Source IP Address(Press Enter for None):  10.0.0.1
Destination IP address(Press Enter for None):  10.0.0.2
Enter Network Protocol(1 for ICMP,6 for TCP,17 for UDP):  1
('adding Rule:', (2048, IPAddr('10.0.0.1'), IPAddr('10.0.0.2'), 1))
Enter the DPID of switch in which you want to insert, delete or show rules:  1
Enter 'A' for adding, 'D' for deleting, 'S' for showing the rules:  'A'
Enter any of the following matching fields
Source IP,Destination IP,Network Protocol,Destination port(UDP or TCP)
Source IP Address(Press Enter for None):  10.0.0.1
Destination IP address(Press Enter for None):  10.0.0.7
Enter Network Protocol(1 for ICMP,6 for TCP,17 for UDP):  1
('adding Rule:', (2048, IPAddr('10.0.0.1'), IPAddr('10.0.0.7'), 1))

Enter the DPID of switch in which you want to insert, delete or show rules:  2
Enter 'A' for adding, 'D' for deleting, 'S' for showing the rules: 'A'
Enter any of the following matching fields
Source IP,Destination IP,Network Protocol,Destination port(UDP or TCP)
Source IP Address(Press Enter for None):  10.0.0.1
Destination IP address(Press Enter for None):  10.0.0.4
Enter Network Protocol(1 for ICMP,6 for TCP,17 for UDP):  6
Enter TCP or UDP destination port:  80
('adding Rule:', (2048, IPAddr('10.0.0.1'), IPAddr('10.0.0.4'), 6, 80))

Enter the DPID of switch in which you want to insert, delete or show rules:  7
Enter 'A' for adding, 'D' for deleting, 'S' for showing the rules: 'A'
Enter any of the following matching fields
Source IP,Destination IP,Network Protocol,Destination port(UDP or TCP)
Source IP Address(Press Enter for None):  10.0.0.7
Destination IP address(Press Enter for None):  10.0.0.8
Enter Network Protocol(1 for ICMP,6 for TCP,17 for UDP):  17
Enter TCP or UDP destination port:  53
('adding Rule:', (2048, IPAddr('10.0.0.7'), IPAddr('10.0.0.8'), 17, 53))
```

Fig. 3. Firewall user interface

B. Results

We testing the ICMP rules by running "pingall" command on our mininet prompt. As can be seen in Fig. 4, h1(10.0.0.1) is not able to ping h2(10.0.0.2) and h7 (10.0.0.2).

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 h4 h5 h6 X h8
h2 -> X h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> X h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 7% dropped (52/56 received)
```

Fig. 4. Ping test between hosts

We tested the web traffic by using "curl" command on h1 & tried to access the web server that is running on h4. The host h1 is not able to access web server as seen in Fig. 5.

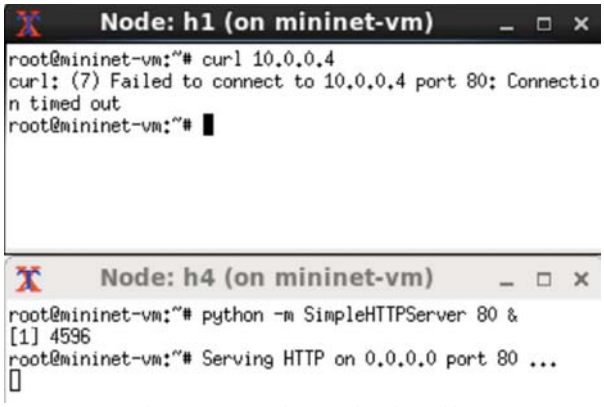

Fig. 5. HTTP test between host h1 and h4

We tested UDP traffic by first implementing UDP server on 53 port on host h8 by using netcat utility "nc" in server mode. We tried to test by running "nc" in client mode on host h7. We tried sending one string "hello" to our UDP server.

We were not able to send the string as shown in Fig. 6. It means that our distributed firewall is working properly and we can use this firewall in any scenario.

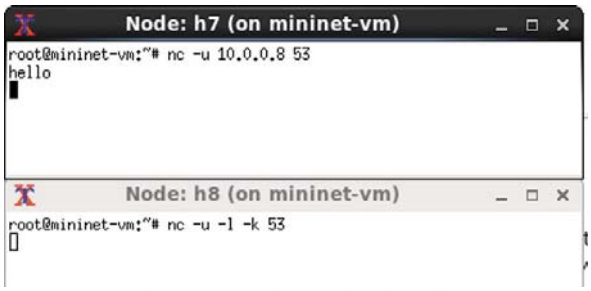

Fig. 6. UDP test between host h7 and h8

CONCLUSION

Our distributed firewall is working properly and is able to handle ICMP, TCP and UDP traffic. We can insert rules in all switches or particular switch based on our requirements. Future work can involve testing this firewall on real hardware. Another area of work could be implementation of Application level distributed firewall.

REFERENCES

[1] Hu, Fei, Qi Hao, and Ke Bao, "A survey on software-defined network and openflow: from concept to implementation," *IEEE Communications Surveys & Tutorials* 16, no. 4 (2014): 2181-2206.

[2] Jarraya, Yosr, Taous Madi, and Mourad Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE Communications Surveys & Tutorials* 16, no. 4 (2014): 1955-1980.

[3] Kreutz, Diego, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE* 103, no. 1 (2015): 14-76.

[4] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review* 38, no. 2 (2008): 69-74.

[5] Scott-Hayward, Sandra, Gemma O'Callaghan, and Sakir Sezer, "Sdn security: A survey," In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pp. 1-7. IEEE, 2013.

[6] Javid, Tariq, Tehseen Riaz, and Asad Rasheed, "A layer2 firewall for software defined network," In *Information Assurance and Cyber Security (CIACS), 2014 Conference on*, pp. 39-42. IEEE, 2014.

[7] Kaur, Karamjeet, Krishan Kumar, Japinder Singh, and Navtej Singh Ghumman, "Programmable firewall using Software Defined Networking," In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, pp. 2125-2129. IEEE, 2015.

[8] Pena, Justin Gregory V., and William Emmanuel Yu, "Development of a distributed firewall using software defined networking technology," In *2014 4th IEEE International Conference on Information Science and Technology*, pp. 449-452. IEEE, 2014.

[9] de Oliveira, Rogério Leão Santos, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete, "Using mininet for emulation and prototyping software-defined networks," In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pp. 1-6. IEEE, 2014.

[10] Mccauley, J. "Pox: A python-based openflow controller." (2014).

[11] Fernandez, Marcial P, "Comparing openflow controller paradigms scalability: Reactive and proactive," In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pp. 1009-1016. IEEE, 2013.