Indian Journal of Science and Technology, Vol 9(14), DOI: 10.17485/ijst/2016/v9i14/88442, April 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Leveraging the Power of Software Defined Paradigm to Control Communication in a Network

X. Agnise Kala Rani and Deepa Balagopal*

Department of Computer Applications, Karpagam Academy of Higher Education, Eachanari, Coimbatore - 641021, Tamil Nadu, India;
agneskala72@gmail.com, deepabalagopal@gmail.com

Abstract

Objective : The numerous protocols and products in the existing network architecture not just complicates the network, but also require that the administrators master the intricacies of configuring the hardware and software, setting the rules, and monitoring and creating reports. In this paper, we present a component that can control the communication between devices from a single point. **Methods:** In this paper, we present a component developed using Python and OpenFlow protocol over POX controller. This is a layer 3-layer 4 component that converts all the switches in a software defined network to filtering devices thus functioning as a communication blocker or packet filter between machines in the network. **Findings:** Our test results show that this component is distributive and topology independent and functions without degrading the performance of the controller. **Improvement:** To the best of our knowledge, our component is one of the first for POX controller since it is topology independent, restrictive, and functions on the basis of IP addresses and not by blocking of MAC addresses.

Keywords: Network Security, OpenFlow, POX SDN Controller, Programmable Networks, SDN, Software Defined Network

1. Introduction

The traditional network architecture is inundated with numerous components and protocols. To introduce a change within a network, all the embedded software in the components needs to be configured individually. With cloud and virtualization being the trend in the industry, this scenario lacks flexibility and adaptability. The lack of global view results in an enormous amount of time to plan changes or to troubleshoot a problem within the network.

The Software Defined Networking (SDN) paradigm digresses from the traditional network architecture. It is intended to solve the problems faced by the classical network, namely-poor scalability, vendor dependence and complexity[1]. SDN provides the ability to orchestrate and automate the networks thus providing greater control and offers various deployment models that may reduce operational expenses[2].

In this paper, we explore how SDN can provide a global mechanism for network management and configuration. We have tried to demonstrate the simplicity and power of SDN by providing the implementation details and test results of a component that can be used to control bidirectional communication between hosts within a local area network. With the component running along with the SDN controller, we demonstrate how all the switches within the entire network can be made to behave like packet filtering devices and how they can be controlled from one point rather than configuring them individually. Our component in this study is developed for the POX[3] controller. POX has several components that enhance its functioning but a packet filter cum communication blocker is absent. Some previous studies[4,5,6] have attempted to create similar components but they are either just functioning by blocking mac addresses in layer 2[4] or have not shown test results with multiple topologies[5,6]. In all these cases, the firewalls work allowing all communication by default and blocking only

*Author for correspondence

the hosts mentioned in the respective data structures. Our component works in reverse –it blocks all hosts by default and allows only those specified by the admin i.e. it is restrictive in nature.

The rest of this paper is organized as follows. We describe the SDN concept in section 2. In section 3 we briefly focus on the OpenFlow protocol. In section 4, we describe the security loophole in the traditional network architecture. We describe the NetWatch component in section 5. In Section 6 we present the implementation details and the test results. In Section 7 we evaluate the benefits of our component and describe some scenarios where it could be used. Finally, we conclude our paper in Section 8.

2. About SDN

The SDN paradigm[7] attempts to separate the control plane from the network devices and place it in a vantage point thus providing it with a global view of the network. Network services can be managed through abstractions of devices' low level functionalities.

The bottom layer in the architecture as depicted in figure 1, known as the data layer, constitutes of the hardware components—switches, routers etc. These devices are responsible for forwarding network packets. The middle layer, where the Controller is positioned, termed as the control plane, is the entity that encapsulates the networking logic. The top layer, known as the application layer, has the applications that interact with the Controller and perform network related services. The aforementioned layers communicate with each other through well-defined APIs.

3. The OpenFlow Protocol

The control and forwarding planes of a software defined network requires a communication interface—this role is played by OpenFlow[8], one of the most popular standards for SDN. The network devices, such as switches and routers in the forwarding plane, can be accessed and programmed via the OpenFlow API[9].

Traditionally, switches have been handling both forwarding and routing decisions. With OpenFlow enabled switches[10], the routing decisions are moved to the Controller. An unknown packet, when received by the Switch, is forwarded to the Controller. The Controller decides on what to do with the packet. The communication between the switch and the Controller is via the OpenFlow protocol.

4. Problem: Security Vulnerability in the Traditional LAN

The private networks within organizations or campuses are protected from the public networks by means of firewalls. But by default, all insiders are trusted-which means that the internal traffic is not seen and cannot be filtered by the traditional firewall. Moreover, the traditional firewall is proprietary, which means that each one comes at a cost and has its own rules for configuring, thus requiring individual effort.

In contrast, with the SDN paradigm, it becomes possible to control insider communication. Implementing components such as firewalls and network monitoring

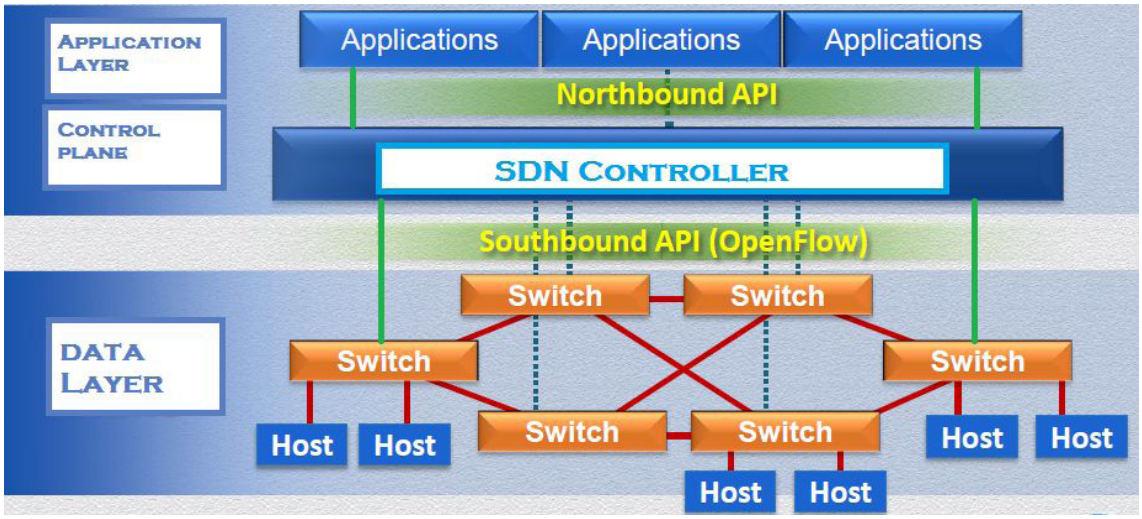

Figure 1: The Simplified SDN Architecture.

services as controller based applications becomes easier with this paradigm. For instance, in small scale networks, if we can implement such services on the controller, the cost of adding additional security layer can be significantly reduced.

Instead of relying on a hardware or software firewall, if a component can empower the switches themselves with packet filtering capabilities, not just external traffic but internal traffic can also be controlled. Again, since it is based on the software defined concept, the switches can be managed from a single point rather than configuring separately.

5. About NetWatch

Our component, NetWatch, is a restrictive firewall component for OpenFlow that empowers the switches for IP or port blocking and packet filtering. Simply stated, when the policy rules are specified, the controller reads them and installs the corresponding flow entries in the switch flow tables. The rules can optionally include the source IP address, the source port, the destination IP address, the destination port and the packet type.

NetWatch includes two types of functionalities:

- It allows only those IP addresses or ports specified in the policy file to transmit packets
- It allows the transmission of only those packet types mentioned in the rule.

NetWatch allows network administrators to restrict communication between host machines within a network. The application can be run directly on the controller for small networks. The first case can be achieved by specifying the source/destination IP or port in the policy file. The second case can be achieved by specifying the packet type along with the IP or port. The keyword "any" can be used in case the user wants to skip specifying any of the above. NetWatch ensures internal security within a network. It has the advantage of being able to filter packets without requiring any significant hardware changes or software modifications. The component can be run alongside the controller framework (Figure 2) and tests show that this does not add to the performance overhead.

Figure 2 NetWatch.

6. Implementation and Test Results

We have implemented NetWatch as a patch to the POX controller. The application that reads the policy file, installs rules and processes the incoming packets is written as a Python program running as a POX module. The pseudo code for NetWatch is given below. The algorithm is designed to work for multiple topologies, ranging from the simplest topology of one switch and multiple hosts, a tree topology with 15 or more switches with varying fan-outs or a linear topology with multiple switches and hosts.

In conducted stress tests using the iperf utility, we did not observe any noticeable difference in the performance of the POX controller before or after adding the NetWatch component. We used the Mininet[11,12] environment to create various network topologies and evaluate our component. For each topology, we defined some rules in the format - source IP address, the source port, the destination IP address, the destination port, packet type - in the config file and using iperf blasted packets from one host to another. Some sample rules we used in the config file were:

- 10.0.0.3 any 10.0.0.16 any any—allows traffic to and from 10.0.0.3 to 10.0.0.16
- 10.0.0.1 any any any any—allows traffic from 10.0.0.1 to all hosts

Algorithm:

```
Input: A set of policies P , set of Switches S

Output:  A firewall that is equivalent to P

For every S Є S
      Insert ICMP & ARP flow entries into the connected switches
For every policy P Є P
      Insert flow table entries into the connected switches
While(true)
     If a new packet arrives
          Check the existing flow entries in the flow table
               If allowed = True
                    Controller inserts bi directional flow entry allowing transmission between
                    source and destination
               else
                    Drop
End
```

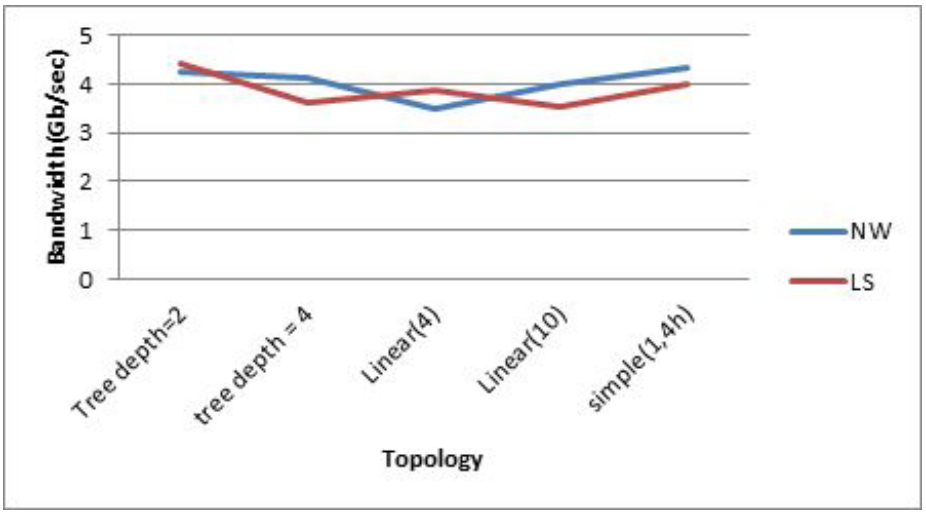

Figure 3: Performance comparison of NetWatch with POX components.

- 10.0.0.3 any 10.0.0.4 5015 UDP—allows UDP traffic to and from 10.0.0.3 to 10.0.0.4 via destination port 5015
- any any any any any—allows all traffic

The graph in Figure 3 shows the performance comparison between our component and another stock component called Learning Switch (LS) in POX with various topologies.

7. Benefits and Applications

NetWatch is distributive in nature since it propagates the policy rules from a central point to all the switches just when the switches connect to the network. Unlike the standard packet filtering software, there is no possibility of a traffic bottleneck since the component itself does not perform the check, instead it empowers the switches for

filtering. As the test results show (Figure 3), it also does not restrict itself to functioning only for a specify topology.

With the above benefits, following are some possible applications of NetWatch envisioned by us for a local area network:

7.1 Content Control for Home or School Network

The component could be used to allow contents only from preferred sites for host machines in a home/school network. This would be particularly suitable if children are using the systems for academic or entertainment purposes.

7.2 Segregation of Networks

NetWatch could be used to segregate one part of the network from another—for instance, a production network and non-production network or the campus lab from the administration network. This would prevent the much needed protection from any probable malicious activity happening within the network.

7.3 Communication Blocker

The component could be used to restrict any form of communication between specific systems within a network. The simple policy specification makes it possible to do so with only the IP address of the source or target systems. If at all a complete stoppage of any kind of communication between all the systems within the network is required, all one has to do is to maintain an empty config file. On the other hand a single rule with the keyword "any" in the config file will allow access between all the systems.

8. Conclusion

In this paper, we have explored the structure of and benefits that could be gained from SDN/OpenFlow network. We have also highlighted a security issue prevalent in the existing networks. To demonstrate simplicity and ease of use of an SDN, we proposed and implemented a novel component that could ensure the internal security of devices within the network. The simulation results show that the security component can effectively convert the switches to filtering devices without affecting the performance of the network. To conclude, our experimental module shows that the SDN paradigm, if implemented in local area networks can prove to be asset in configuring and controlling the network as a whole from a single vantage point.

9. References

1. Feamster N, Rexford J, Zegura E. The road to SDN: an intellectual history of programmable networks. ACM SIGCOMM Computer Communication Review. 2014 Apr 8; 44(2):87–98.
2. Eadala Sravan Yadav, Nagarajan V. A Review on Deployment Architectures of Path Computation Element using Software Defined Networking Paradigm. Indian Journal of Science and Technology. 2016; 9(10):1–10.
3. POX Controller. Date accessed:19/12/2015: Available from: http://www.noxrepo.org/pox/about-pox/.
4. Javid T, Riaz T, Rasheed A. A layer2 firewall for software defined network. In Information Assurance and Cyber Security (CIACS). IEEE, 2014 Conference. 2014 Jun 12; p. 39–42.
5. Suh M, Park SH, Lee B, Yang S. Building firewall over the software-defined network controller. In Advanced Communication Technology (ICACT). IEEE, 2014 16th International Conference. 2014 Feb 16; p. 744–48.
6. Kaur K, Kumar K, Singh J, Ghumman NS. Programmable firewall using Software Defined Networking. In Computing for Sustainable Global Development (INDIACom), IEEE, 2015 2nd International Conference. 2015 Mar 11; p. 2125–29).
7. Sezer S, Scott-Hayward S, Chouhan PK, Fraser B, Lake D, Finnegan J, Viljoen N, Miller M, Rao N. Are we ready for SDN? Implementation challenges for software-defined networks. Communications Magazine, IEEE. 2013 Jul; 51(7):36–43.
8. Kreutz D, Ramos FM, Esteves Verissimo P, Esteve Rothenberg C, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. Proceedings of the IEEE. 2015 Jan; 103(1):14–76.
9. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review. 2008 Mar 31; 38(2):69–74.2.
10. Open Network Foundation: OpenFlow Switch Specification version 1.5.1. 2015 April.
11. Mininet. Date accessed: 19/12/2015: Available from: http://mininet.org/.
12. Handigol N, Heller B, Jeyakumar V, Lantz B, McKeown N. Reproducible network experiments using container-based emulation. ACM, Proceedings of the 8th international conference on Emerging networking experiments and technologies. 2012 Dec 10; p. 253–64.