Firewall and Load Balancer System using Software Defined Networks

Vijayalaxmi Dhabbe[1] Aparna Mali[2] Prof. Sanjay Pawar[3]
[3]Professor
[1,2,3]Department of Computer Engineering
[1,2,3]Sinhgad Institute of Technology and Science, University of Pune, Pune, Maharashtra, India

Abstract— Network usage and demands are growing at a very fast rate and to meet the current requirements, there is a need for automatic infrastructure scaling. Software defined networking, referred to as a revolutionary new idea in computer networking, is the new approach that promises to dramatically simplify network-control and the management plane. It is achieved through innovative network programmbility. OpenFlow, one of the techniques of SDN technology, is a new approach to networking. Because of many limitations, the development of the traditional load balancing technology has encountered bottlenecks. This has forced companies to find new load balancing method. Software Defined Network (SDN) provides a good method to solve the load balancing problem. We are proposing firewall System for SDN environment which identifies malicious behaviors or attacks and reports to network administrators as intrusion events. In this paper, we implemented load balancing algorithm that based on the latest SDN network architecture. We are using Dijkstra's algorithm to find multiple paths of same length which enables us to reduce the search to a small region in the fat tree topology. The technologies we describe enable network operators to implement a wide range of network policies in a high-level policy language and easily determine sources of performance problems. In addition to the systems themselves, we describe prototype deployments in campus networks that demonstrate how SDN can improve common network management tasks.

Key words: Software-Defined Network, Floodlight Controller, Firewall, load balancer System

I. INTRODUCTION

Network management is challenging. To operate, maintain and secure a communication network. So the network operators must use tool to low-level vendor-specific configuration to implement complex high level network policies. Despite many previous proposals to make networks management easier, many solutions to network management problems amount to remove gap solutions because of the difficulty of changing the underlying infrastructure.

Today, network operators must implement increasingly well-informed policies and complex tasks with a limited and highly constrained set of low-level device configuration commands in a command line interface environment. Not only are network policies low-level, they are also not well equipped to react to continually changing network conditions. State-of-the-art network configuration methods can implement a network policy that deals with a single snapshot of the network state. However, network state changes continually, and operators must manually adjust network configuration in response to changing network conditions. As a result configuration changes are frequent and unwieldy, leading to frequent updating configurations[2] .

Network operators need better ways to con-figure and manage their networks. Unfortunately, today's networks typically involve integration and interconnection of many proprietary, vertically integrated devices. This vertical integration makes it incredibly difficult for operators to specify high-level network-wide policies using current technologies. Innovation in network management has thus been limited to stop-gap techniques and measures, such as tools that analyze low-level configuration to detect errors or otherwise respond to network events. Proprietary software and closed development in net-work devices by a handful of vendors make it extremely difficult to introduce and deploy new protocols. Incremental "updates" to configuration methods and commands are generally dictated unilaterally by vendors. Meanwhile, operators' requirements for more functionality and increasingly complex network policies continue to expand.

In software system outlined Network, directors will form traffic by programming the management at control plane while not interrupting network devices (Switches) at knowledge plane. Management (Control) plane includes a centralized controller, that sets forwarding rules in switch to route traffic from supply to destination. Knowledge (data) plane that consists of network devices handles all packets according the flow entries set by the controller. All flow entries at the switch level are managed by centralized logical controller. With this ability, SDN technique is introduces to handle the cloud service suppliers challenges like dynamical load traffic, additional information measure demand, and security and measurability problems. Enterprise and organization use OpenFlow based mostly SDN to balance the traffic load, direct the traffic, manage on demand, information measure demand and execute polices to scale the network. The software system outlined Networking (SDN) approaches has been earlier adopted by several firms for his or her business significantly cloud and telecommunication services. The normal technologies not able to drive their current business challenges, therefore trendy network techniques supported software system. Application service.

The remainder of this paper is structured as follows. In Section II, we give an overview of the firewall system, load balancing technology and OpenFlow technology; briefly summarizes the research status of OpenFlow-based load balancing. Section III introduced the implement of our two algorithms in more detail.one algorithm for firewall and other one for load balancing. The results about our experiments are described in Section IV. We summarize the main contribution of this paper in Section V.

II. TECHNICAL BACKGROUND

A. SDN Controllers

An SDN controller is an application in software defined networking that manages flow control to enable programmable networking. SDN controllers are based on

protocols such as OpenFlow, that allow servers to tell switches where to send packets.

Five topmost open source controllers in terms of their usage have been analyzed here: POX, Trema, Ryu FloodLight, and OpenDaylight To understand and realize the SDN concept we must choose a suitable controller.

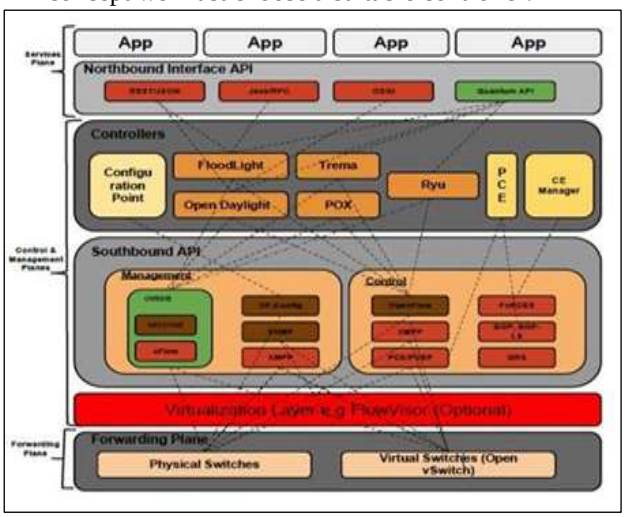

Fig. 1: SDN Controllers

Here we are using FloodLight controller The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers and the number of engineers from Big Switch Networks also included. OpenFlow is an open standard managed by Open Networking Foundation. It specifies a protocol through switch a remote controller can change the behavior of networking devices through a well-defined "forwarding instruction set". The controller are designed to work with the increasing number of switches, virtual switches, routers and access points that support the OpenFlow standard. We are using FloodLight controller v1.2 version.

B. OpenFlow Protocol

OpenFlow provides a mechanism for SDN. While OpenFlow was first proposed as a way to enable researchers to conduct experiments in campus networks, its advantages lead to its use beyond that. OpenFlow's central-control model can avoid the need to construct global policies from switch-by-switch configurations, and can also support near-optimal traffic management [6]. The behavior of a forwarding device can be summarized in two steps: (1) When it receives a packet that does not match a certain entry in its routing table, it contacts the controller that defines how the packet should be forwarded or discard the packet; (2) when the received packet matches a rule in its routing table, the corresponding action in the forwarding table is performed [11].

C. Mininet

Mininet is a network emulator which creates a virtual network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking. It supports custom topology. But also Provides a straightforward and extensible Python API for network creation and experimentation. The figure 2 shows the creation of the custom topology.

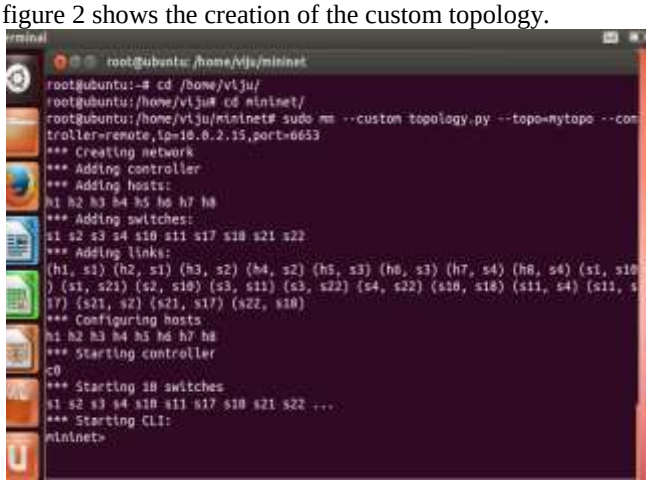

Fig. 2: Mininet Fat Tree Topology Creation

D. Firewall system

In Open Flow switch the ingress packet matched against the flow table and sent to the controller if no match is found. The controller chooses what to do with the packet and sends it back to the switch. The switch then executes the activity which the controller characterized[1].In the event that the packet in the flow table matches the table then the activity is execute. If the packet doesn't match against list it will block that address.

E. Load Balancing Technology

The main goal is to perform load balancing but at the same time we need to ensure that the latency is minimum. Here are using Dijkstra's algorithm to help us to find multiple paths of same length which enables us to reduce the search to a small region in the fat tree topology. In the program simply finds the path with least load .It will forwards traffic on that path[3].

III. IMPLEMENTATION AND EVALUATION

A. Experimental environment for firewall system

It is a firewall application implemented in a software defined net-work, using mininet and python based on the custom topology as shown in the figure 2.In figure 2, a small network is set up on the virtual machine, with mininet installed. This network contains a remote controller, ten switches; each with a host. This topology is utilized to test different principles and strategies characterized in the firewall design. Algorithm: firewall.py: The code actualizes a firewall application in a SDN to block particular host-to-host correspondence based on the time of the day. It recognizes the ip addresses to be blocked.

B. Experimental environment for load balancer system

It is a load balancing application implemented in a software defined net-work, using mininet and python based on the custom fat tree topology as shown in the figure 2.In figure 2, a small network is set up on the virtual machine, with mininet installed. This network contains a remote controller, ten switches; each with a host. This topology is utilized to test different principles and strategies characterized in the firewall design. Algorithm: loadbalancer.py: The code actualizes a load balancing application in a SDN to balancing network[4].

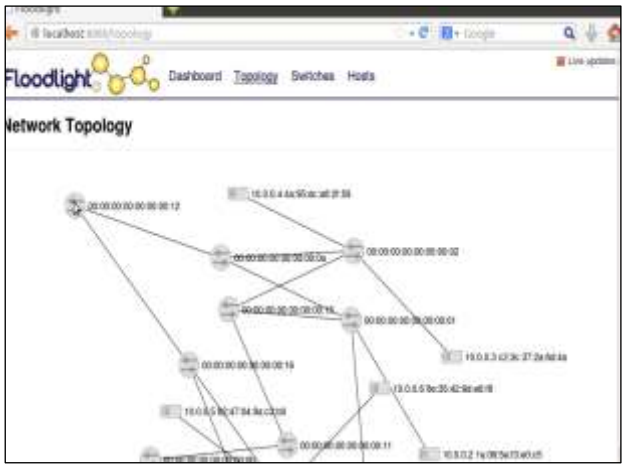

Fig. 3: Topology for load balancer system

C. Experimental program for Firewall system

In the system run FloodLight Controller. Run the fat tree custom topology topology.py. Type the following command in Mininet xterm h4 h7.Now ping h4 to h7 and h7 to h4 .here h7 is authorized host. Terminal open a new tab Ctrl + Shift + T and type sudo wireshark. In wireshark, go to Capture->Interfaces and select s1-eth4 and start the capture. You will see the packets are transferring. Now go to your Terminal and open a new tab and run the firewall.py script and it will display device information and it will block the h7.n wireshark, go to Capture->Interfaces and select s1-eth4 and start the capture .no packets are present

D. Experimental program for Load balancer system

In the system run FloodLight Controller. Run the fat tree custom topology topology.py. Type the following command in Mininet xterm h1 h1.In first console of h1 type, ping 10.0.0.3.In second console of h1 type, ping 10.0.0.4.On Terminal open a new tab and type sudo wireshark. In wireshark, go to Capture->Interfaces and select s1-eth4 and start the capture. In filters section of wireshark type ip.addr==10.0.0.3 and check if you are receiving packets for h1 -> h3. Do same thing for h1->h4. Once you see packets, you can figure that this is the best path. But to confirm it is, repeat the above two steps for s1-eth3 and you will find that no packets are transmitted to this port. Only packets it will receive will be broadcast and multicast. Ignore them. Now in the second console of xterm of h1, stop pinging h4. Our goal is to create congestion on the best path of h1->h3, h1->h4 and vice versa and h1 pinging h3 is enough for that Go to your Terminal and open a new tab and run the loadbalancer.py program by Providing input arguments such as 1,4,3 where 1 is host 1, 4 is host 2 and 3 is host 2's neighbor. Look at the topology above and you will find that these hosts are nothing but h1, h4 and h3 respectively. The loadbalancer.py performs REST requests, so initially the link costs will be 0. Re-run the script few times. This may range from 1-10 times. Here the best route is for h1->h4 and vice versaNow on second console of h1 type ping 10.0.0.4.Go to wireshark and monitor interface s1-eth4 with the filter ip.addr==10.0.0.x where x is 3 and 4. You will find 10.0.0.3 packets but no 10.0.0.4 packetsStop the above capture and now do the capture on s1-eth3, s21-eth1, s21-eth2, s2-eth3 with the filter ip.addr==10.0.0.x where x is 3 and 4. You will find 10.0.0.4 packets but no 10.0.0.3 packets

Fig. 4: Output of Load Balancer System

E. Result

Transfer (Gbytes) - BLB	B/W(Gbits) - BLB	Transfer (Gbytes) - ALB	B/W(Gbits) - ALB
15.7	13.5	38.2	32.8
21.9	18.8	27.6	32.3
24.6	21.1	40.5	34.8
22.3	19.1	40.8	35.1
39.8	34.2	16.5	14.2
Average = 24.86	Average = 21.34	Average = 32.72	Average = 29.84

iPerf H1 to H3 Before Load Balancing (BLB) and After Load Balancing .

Transfer (Gbytes) - BLB	B/W(Gbits) - BLB	Transfer (Gbytes) - ALB	B/W(Gbits) - ALB
18.5	15.9	37.2	31.9
18.1	15.5	39.9	34.3
23.8	20.2	40.2	34.5
17.8	15.3	40.3	34.6
38.4	32.9	18.4	15.8
Average = 23.32	Average = 19.96	Average = 35.2	Average = 30.22

iPerf H1 to H4 Before Load Balancing (BLB) and After Load Balancing

Min	Avg	Max	Mdev
0.049	0.245	4.407	0.807
0.050	0.155	4.523	0.575
0.041	0.068	0.112	0.019
0.041	0.086	0.416	0.066
0.018	0.231	4.093	0.759
Avg:0.0398	Avg:0.157	Avg:2.7102	Avg:0.4452

Ping from H1 to H4 before Load Balancing

Min	Avg	Max	Mdev
0.039	0.075	0.407	0.068
0.048	0.078	0.471	0.091
0.04	0.072	0.064	0.199
0.038	0.074	0.283	0.039
0.048	0.099	0.509	0.108
Avg:0.0426	Avg:0.0796	Avg:0.3468	Avg:0.101

Ping from H1 to H4 After Load Balancing.

IV. CONCLUSION AND FUTURE WORK

The practical virtual systems is create, running genuine portion, switch and application code, on a solitary machine utilizing Mininet. Firewall Policies are defined and rules added to the fire-wall table which changes progressively taking into account .necessity and time of the day. Selective obstructing of packets in view of source and destination IP address will carried out. Load balancer system is also working. Application processing logic implemented using the floodlight OpenFlow Controller. The project can further extended work by investigating more related load-balancing algorithms and multicast algorithms for SDN. Also in future load balancer system can be evaluated by considering more parameters.

REFERENCES

[1] Dependra Dhakal, ”Campus Network using Software Defined Networking” in SMIT,CSE Department, Rangpo, Majitar, Sikkim ,International Journal of Computer Applications (0975 - 8887)Volume 138 - No.4, March 2016
[2] Ben Pfaff, “The design and implementation of open vswitch”, in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI),USENIX, Ed., vol. 1, no. 1. Oakland, CA: USENIX, May 2015, pp.1–14.
[3] Hailong Zhang1, “SDN-BASED LOAD BALANCING STRATEGY FORSERVER CLUSTER”, 2014 IEEE, May 2014.
[4] Senthil Ganesh N, “Dynamic Load Balancing using Software Defined Networks” in International Journal of Computer Applications (0975 – 8887),International Conference on Current Trends in Advanced Computing (ICCTAC-2015)

