Implementation and analysis of network functions in a virtualized topology using RYU controller

Dr. Nader F. Mir, Pinak N. Karadkar

Department of Electrical Engineering, San Jose State university, San Jose, California 95192.

Introduction

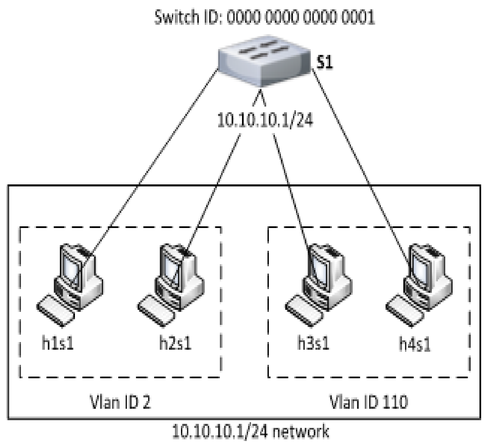
Virtualization has been studied and developed to change the way networks operate by enhancing standard IT technologies to consolidate network component types in large servers, storage devices and switches. In traditional network establishments such as routers, firewalls and content delivery appliances, each function was implemented by a certain set of software and hardware. However, the software and hardware for such networks was interdependent, viz. they cannot be separated easily.

The primary goal of this project is to implement a routing functionality on a virtualized network topology using simulation software Mininet and OpenFlow controller RYU. The parameters that would be analyzed are source and destination IP address, link capacity, bandwidth usage, host reachability, etc. In addition to this, security features such as firewall implementation and packet filtering is also discussed extensively. As an application, this project may also be integrated with a web GUI to ease the reading of all the aforementioned parameters.

Design and topology

Network topology

The network topology that was created for the implementation of this scenario to test firewall functioning consists of a simple single switch attached to four hosts with two of those hosts in one VLAN and the other two hosts in another VLAN.



Methodology

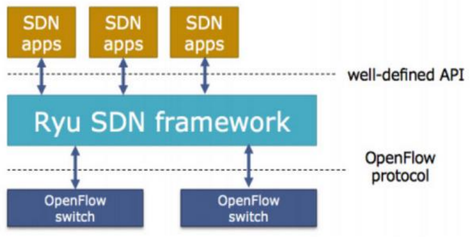
The traffic that the hosts generate passed through the virtual switches in mininet. RYU controller extracts this information using the code that was developed for the project in Python. RYU is an open source controller and is freely available on github using any of the following commands.

% pip install ryu

For installing from source code;

% git clone git://github.com/osrg/ryu.git

% cd ryu; python ./setup.py install



This topology can be implemented or created by using the Mininet command. As mininet works on a single Linux operating system, the command is preceded by a $ for shell prompt and # for root shell prompt. Mininet precedes mininet command that will be entered in the shell prompt. The *sudo mn* command creates a mininet topology. The commands that are concatenated to it are external features to be added. The command for the topology that we use in this network implementation looks like this:

sudo mn --topo single,<. number of hosts> --mac --switch ovsk –controller=remote

The firewall execution is displayed at the successful output of the firewall enable command. While the result is displayed, the parameters can be observed that include the details of firewall running after which the rules will be added to the firewall. Rules can range in many ways

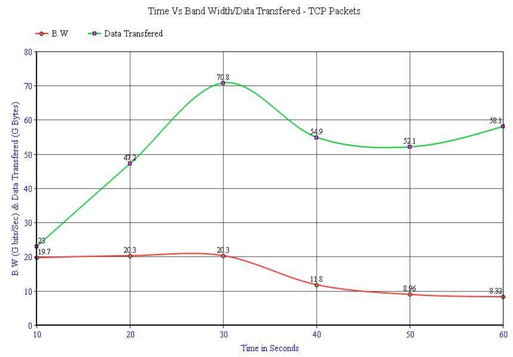
```
root@bharath-VirtualBox:~# curl -X POST -d '{"nw_src": "10.0.0.0/8", "nw_proto"
: "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001/2
[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "vla
n_id": 2, "details": "Rule added. : rule_id=1"}]}]root@bharath-VirtualBox:~#
root@bharath-VirtualBox:~#
root@bharath-VirtualBox:~#
root@bharath-VirtualBox:~# curl -X POST -d '{"nw_dst": "10.0.0.0/8", "nw_proto"
: "ICMP"}' http://localhost:8080/firewall/rules/0000000000000001/2
```

The rules can be altered in such a way that they block a few packets from or to a particular network. For example, in this project 2. implementation, ICMP packets are allowed from 10.0.0.0 to the network 10.0.0.0. This implementation is allowed only for one of the VLANs with ID

Since ICMP packets are allowed from 10.0.0.0 network to 10.0.0.0 network under VLAN 2, ping is successful with 0 % packet loss and average RTT as 0.14 millisecond.

Results

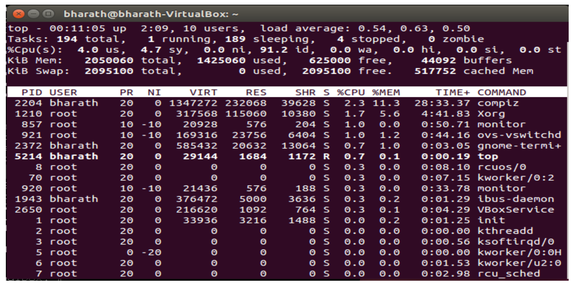
The packets that were monitored using the Wireshark tool and the packet generation using *iperf* utility tool are also focused and analyzed in this section. Graphical analyses is more accurate and lets the user understand the exact parameters and their performance evaluation and their role in the project. This section plays a very important role in the conclusion of the project.



The figure shows how the implemented firewall only allows ICMP packets to pass throughout the network.

```
[FW][INFO] dpid=0000000000000001: Blocked packet = ethernet(dst='00:00:00:00:00:
04',ethertype=33024,src='00:00:00:00:00:03'), vlan(cfi=0,ethertype=2048,pcp=0,vi
d=110), ipv4(csum=27813,dst='10.0.0.4',flags=2,header_length=5,identification=47
613,offset=0,option=None,proto=1,src='10.0.0.3',tos=0,total_length=84,ttl=64,ver
sion=4), icmp(code=0,csum=46184,data=echo(data='\x03\xdb"U\x00\x00\x00\x00\x00P\xc8\
x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\
x1e\x1f !"#$%&\'()*+,-./01234567',id=3531,seq=1),type=8)
```

Figure below shows the CPU utilization of the designed system while implementing the firewall functionality.



4.0 µs - Means that the processor is utilizing or spending 4% of its time running user space processes (does not belong to the kernel). Most of the times, when the CPU is not idle, it spends its time running user space programs. [18]

Conclusions

The main focus of the project was to observe and analyze how security really works in present infrastructures and how it can be implemented on a virtual environment. The following conclusions can be drawn after the implementation:

1. CPU utilization: The amount of CPU time used in the designed system is much lesser than the hardware equivalent of firewall routers or packet filters.
2. Invoking RYU and extracting data: RYU, which also follows the OpenFlow protocol scheme is successfully invoked and establishes connection with the Open vSwitch in order to extract the data from the flow tables of the switches. The controller successfully manages and controls the switches and helps in the network functions to achieve their tasks. A python module is implemented to perform certain functionalities. This module allows the controller to fetch flow table data from the switches.

Key References

1. Guru Parulkar, Saurav Das, Nick Mckeown, “SDN based Unified Control Architecture” http://archive.openflow.org/downloads/technicalreports/MS_Thesis_Polito_2009_Manuel_Palacin_OpenFlow.pdf

2. Gelberger, A., Yemini, N., & Giladi, R. (2013). Performance Analysis of Software-Defined Networking (SDN). In *Proceedings of the 2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems* (pp. 389-393). IEEE Computer Society

3. Manuel Palacin Mateo (2009) “OpenFlow Switching Performance” http://archive.openflow.org/downloads/technicalreports/MS_Thesis_Polito_2009_Manuel_Palacin_OpenFlow.pdf

Network Functions Virtualization white paper, “Network operator perspectives on Industry progress” (2013) at the SDN and OpenFlow world Congress. https://portal.etsi.org/nfv/nfv_white_paper2.pdf

5. Introduction to Mininet by Bob Lantz, Nikhil Handigol, Brandon Heller, and Vimal Jeyakumar. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

6. Carmi Arad, Gil Levy “Exact match hash lookup databases in network switch devices” US Patent number 2014030194 A1. http://www.google.com/patents/US20140301394

Acknowledgements

The author would like to take this opportunity to thank Prof. Nader F. Mir for his valuable guidance and inputs that helped in the successful completion of the project.