A Virtual Intrusion Detection System Based on Ryu

Biyang Fu
School of Computing
Clemson University
Clemosn,SC,US
biyangf@clemson.edu

Linyu Zhang
School of Computing
Clemson University
Clemosn,SC,US
linyuz@clemson.edu

Jie Hao
School of Computing
Clemson University
Clemosn,SC,US
hao6@clemson.edu

Zhenchao Sui
School of Computing
Clemson University
Clemosn,SC,US
zsui@clemson.edu

***Abstract*—Network security defense is one of the important tasks in the popularization and use of the Internet. It has attracted many network security experts and enterprises to study. Similarly, software-defined network (SDN) has received more and more attention in recent years because of its simplicity, flexible expansion, and openness. With the freely-movable SDN, engineers will be able to quickly and high-level view all areas of the network and modify the network to change the rules. This freedom and control can also bring better security to your system. By quickly restricting and viewing the inside of the network from a central perspective, managers can effectively make changes. In this paper, by coupling Snort and Open vSwitch to Ryu, a set of component-based SDN framework, it realizes the functions of network traffic monitoring, dynamic scheduling and load distribution.**

***Index Terms*—Network security; SDN; traffic monitor; Snort; Open vSwitch; load distribution**

I. Introduction

With the rapid development of basic network technology and network applications and the improvement of users' network performance requirements, network governance has become an urgent problem to be solved. Effective network traffic monitoring [8] can ensure the stable operation and sustainable development of the network. More importantly, With the expansion of network scale and the development of hacker technology, the number of intrusion and attack cases is increasing, which poses serious challenges to stable network services, information security, and Internet order. Network traffic monitoring plays an important role in the entire network governance system.

At present, with the continuous deepening of network applications and frequent technology upgrades, security threats such as illegal access and malicious attacks are also constantly being updated and intensified. Security protection and management systems [9] such as firewalls, VPNs, IDS, antivirus, identity authentication, data encryption, and security audits have been widely used in the network. From the perspective of network security professionals, the most direct requirement is to monitor various operational performance status in the network according to classification in a unified interface, obtain relevant data information, log information, and alarm information, etc., and perform classification, summary, analysis, and audit, at the same time complete the attack event alarm, response and other functions. Therefore, the user's network governance [10] needs to continuously improve the overall network security governance solution, from the overall control configuration of the unified security governance platform to the multi-level, distributed security system, to achieve centralized monitoring, policy management, auditing of various network security resources, and interaction between multiple security function modules, thus effectively simplifying network security governance, and improving network security level and availability, controllability, and governance.

In this project, our goal is to develop a virtual intrusion detection system (vIDS) based on containers and virtual machines. We integrated Snort and Open vSwitch into the SDN framework and controlled the flow monitoring, scheduling, and forwarding of the entire system by rewriting the rules of the Ryu controller. Based on Snort's function, we created a simple Deep Packet Inspection (DPI) system which can install/delete signatures dynamically. That is to say, the user of the IDS can install/delete signatures without rebooting the IDS. In the following work,, these features can be used to implement dynamic scaling of IDS. Conceptually, vIDS exists in the form of "instance" and a controller is required to manage the vIDS instances. So, we pass the rewritten rules of the Ryu controller to the Open vSwitch APIs to monitor the network traffic. Once the system detects an overload of a vIDS instance, it will boot another vIDS instance according to the rules and some of the traffic will be forwarded to the new vIDS instance.

II. Related Work

Network traffic [11] is the amount of data transmitted on the network. All information is contained in network communication data packets. Traffic monitoring [8] is the management and control of data packets communicating on the network, and the optimization and limitation of it. The purpose is to allow and guarantee the efficient transmission of useful data packets, prohibiting or restricting the transmission of illegal data packets, one guarantee one limit is the essence of traffic monitoring. There are usually four methods for traffic monitoring: the first is to use the Simple Network Management Protocol (SNMP) [12] technology, which can support the network management system to monitor the operation of the device; the second is to use hardware probe technology to connect the probes in series to capture in the traffic link, the

traffic information is obtained by shunting the data packets on the link; the third is to use port mirroring technology [13] to forward the data traffic of one or more source ports on the switch or router to a specified port to achieve the network monitoring. Without seriously affecting the normal throughput of the source port, you can monitor and analyze the network traffic through the mirror port. Port mirroring analysis technology [13] is a more basic method of network testing, especially suitable for network failure analysis; the fourth is the use of software technology, based on applications and flow control strategies to analyze network traffic and quickly find problems in the network. This is the most common and complete network traffic monitoring strategy today.

In the beginning, we study the paper "Deep Packet Inspection as a Service" [14], which is provided by our professor, to learn how to create DPI instances as services. In this paper, the processing and analysis tasks performed in the middleboxes are provided as services. In today's traffic transmission architecture, traffic needs to reach its destination through a series of middleboxes with DPI components. Each pass through the middleboxes will be scanned over and over again to ensure its security. This means it will take a lot of time to process DPI tasks. This paper we studied combines the DPI components from various middleboxes, and deploys one or more DPI service instances as services provided to different middleboxes in the network. So we don't have to scan the DPI tasks over and over again, against provide the results of the services to various middleboxes.

Snort [15] is a free IDS software. Snort first creates a directory based on the remote IP address and then records the detected packets in the tcpdump binary format or stores them in these directories in its own decoded form. In this way, you can use Snort to monitor or filter your location. Required packages. Snort is a lightweight intrusion detection system, which has the ability to intercept network data packets, perform real-time network data analysis, alarm, and log. Snort's message interception code is based on the libpcap library, inheriting the platform compatibility of the libpcap library. It can perform protocol analysis, content search/matching, and can be used to detect various attacks and detections, such as buffer overflow, covert port scanning, CGI attacks, SMB detection, OS fingerprint feature detection, and so on. Snort uses a flexible rule language to describe network data packets, so it can quickly translate new attacks. Snort has real-time alarm capability, and can write alarm information to Syslog, specified files, UNIX sockets, or use WinPopup messages. Snort has good expansion capabilities, it supports a plug-in system, and can easily add new functions through its defined interface. Snort can also record network data, and its log file can be in tcpdump format or decoded ASCII format.

Open vSwitch (OVS) [16] is a virtual switch running on a virtualization platform, which supports the OpenFlow protocol and also supports tunneling technologies such as gre / vxlan / IPsec. Before OVS, Linux-based virtualization platforms such as KVM or Xen [17] lacked a feature-rich virtual switch, so OVS quickly rose and began to become popular in Xen / KVM [17], and is used in more and more open-source projects, such as Network solutions in OpenStack neutron. In terms of Flow controllers or management tools for virtual switches, some commercial products have integrated controllers or management tools, such as Cisco 1000V's Virtual Supervisor Manager (VSM) [18] and VMware's vCenter [19] in distributed switches. OVS requires the use of third-party controllers or management tools to implement complex forwarding strategies. For example, OVS supports OpenFlow protocol, we can use any controller that supports OpenFlow protocol to remotely manage OVS. The ML2 plug-in in OpenStack Neutron [20] can also manage OVS. But this does not mean that OVS must have a controller to work. Without connecting an external controller, OVS itself can rely on MAC address learning to implement Layer 2 packet forwarding, just like Linux Bridge [21]. OVS supports NetFlow, IPFIX, sFlow, SPAN / RSPAN, and other traffic monitoring protocols; supports fine ACL and QoS policies; can use OpenFlow and OVSDB protocols for centralized control; supports VM interface-based traffic management strategies and other functions.

III. Problems to be solved

The workload of the system's DPI is huge that we have to use different devices to do DPI work. Thus, the management of the DPI devices is a problem.How to balance the workload to each device and how to communicate with them.Inspired by current suggestions for Network Function Virtualization (NFV)and the flexible routing capabilities of Software Defined Networks (SDN), this project focus on Deep Packet Inspection (DPI), wherethe payload of packets is inspected against a set of patterns. DPI is a common task in many middleboxes. To implement this there are some problems to be solved.

- The workload of the system's DPI is huge that we have to use distributes devices to do DPI work. Thus, the management of the DPI devices is a problem.How to balance the workload to each device and how to communicate with other middleboxes.
- The compatibility issue.As different middlebox type may have different pattern sets. How to integrated the different pattern sets,and let the DPI service perform with different sets is a problem to be solved.
- Support simple Deep Packet Inspection (DPI) by string matching. Snort is a well known signature-based IDS. Can install/delete signatures in the IDS dynamically. That is to say, the user of the IDS can install/delete signatures without rebooting the IDS. In practice, these features can be used to implement dynamic scaling of IDS.
- Can create a new vIDS instance when the existing vIDS is overloaded. Conceptually, vIDS exists in the form of "instance" and a controller is required to manage the vIDS instances. But in this project, we are not going to implement a controller for vIDS. Instead, you should implement a program to monitor the traffic via Open vSwitch APIs. Once this program detects an overload of a vIDS instance, the program will boot another vIDS

instance and some of the traffic will be forwarded to the new vIDS instance.

- How to create a new instance is a question to consider. First, we want to create a new VM to run Snort, but we do n't know if this is possible. After research, we found that using another VM to start the VM has many defects, so we must find other reliable and reasonable ways, and finally we use Docker as a container.
- We speculate to connect docker0 to the bridge of topology s1 with a network bridge. But in this way, the connection problem has not been solved. The Snort instance in the Docker container cannot obtain network packets from outside host1 and host2. In order to solve this problem, we searched for related materials and did some experiments. Finally, we can solve this problem by creating a custom network and providing a static ip address for the new Docker image.
- In addition, we need to send the network data to the host and snort, regarding forwarding issues. We need to rewrite the program code of the controller to complete the addition of rules for forwarding using groups and buckets. However, after several initial experiments, it has been confirmed that this method of control cannot perfectly achieve the reasonable distribution of data packets. Not sure if it's OpenFlow's problem or our operation problem. In order to solve this problem, we tried other ways to achieve it, such as dynamically updating the flow table.

IV. Approach

In this project, we need to develop virtual intrusion detection system based on container and the virtual machine (vIDS), we will Snort and Open vSwitch integrated into SDN framework, and rewrite the rules of Ryu controller to control the traffic monitoring, scheduling and forward on the basis of the function of the Snort, we created a simple deep packet inspection (DPI) system, the system can dynamically install or remove the tag. In other words, IDS enables users to install or remove signatures without restarting the IDS. These capabilities can be used to implement dynamic extensions to IDS. Conceptually, vIDS exists as an instance and requires a controller to manage the vIDS instance. Therefore, we pass the rules that the Ryu controller overrides to the Open vSwitch API to monitor network traffic. When the system detects that one vIDS instance is overloaded, it will run another vIDS instance according to the rules, and some traffic will be forwarded to the new vIDS instance.

We use mininet for basic networks. There are three hosts and a switch, and there are three links between the switch and the host, which is also a controller. We installed the snort application in docker. So first, we need to start docker to run snort. After docker starts, we can see the rules in snort. Once we're done editing the rules, we can start snort. We need to link the snort docker addition to the switch, so we need to add a port to s1. In ryu-manager, we need to run 2 files, "mymonitor.py" and "simple switch 3.py". Now we need to run the controller. We use the ping command to simulate network traffic for packages of different sizes, and when ping small packages, the switch and snort0 don't experience any pressure as the progress progresses, so the controller doesn't do anything but display network status information. When the command is switched to using the big package ping, after a few seconds, when the network traffic exceeds the limits we set in our program, the controller starts routing traffic to snort1 in another docker. So 2 dockers receive information from the network. When we stopped the bulk ping to simulate reduced network traffic pressure, the controller stopped routing traffic to snort1. After the controller stops routing traffic to snort2, only snort 1 still shows the network package in the console window, and snort2 has stopped completely.

V. Implementation

A. About the paper we study

At the beginning, we tried to run the demo of the the paper we study.The source code of the paper is published on GitHub https://github.com/DeepnessLab/moly. This demo is run on a topology shown in Figure 1

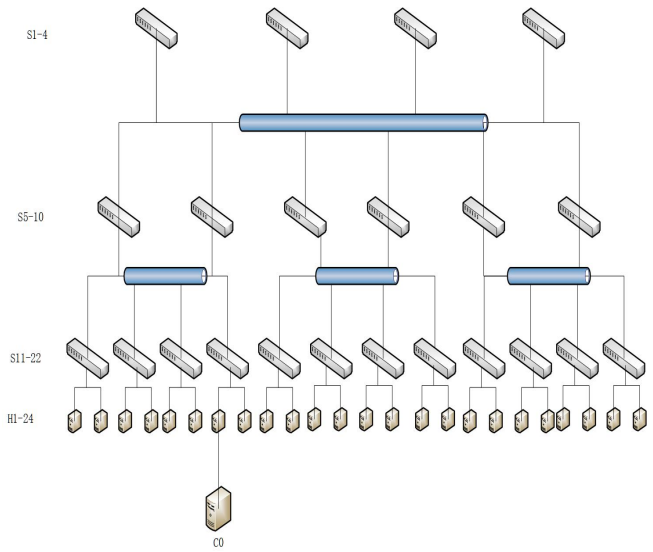

Fig. 1. Topology of the paper's demo.

We run a Ryu as a controller on a virtual machine and run DPI Controller on the host7. We use host 1,2,3 as middleboxes and we can successfully register the middleboxes on the DPI Controller.And then when it comes to creating DPI instances on host 4,5,6, we were stuck.Because we can't create the instance by following the guide. We communicated with the author to ask for help, but the source code of create DPI instance is not fully implemented.At first, we want to implement this part by ourselves, but after going though the code, we found that the DPI Instance has a strong coupling with the DPI Controller, middleboxes and TSA. The workload may be too much. So we decided to follow the professor's advice to implement a vIDS by ourselves.

B. Virtual IDS

We use a simple topology to build a small vIDS. The topology is shown in the Figure 2

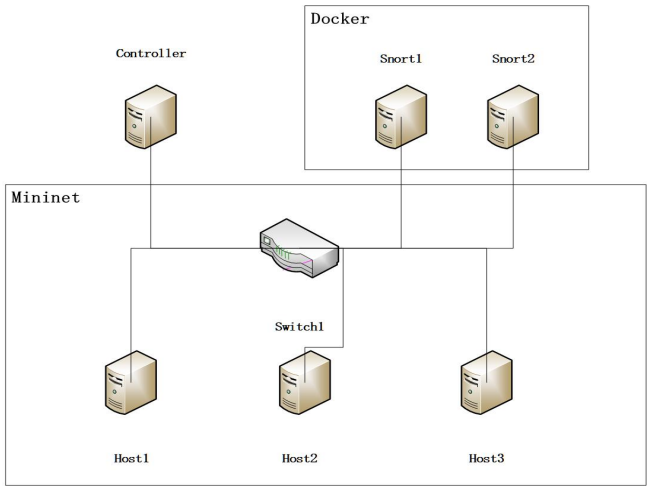

Fig. 2. Topology of vIDS.

1) Environment: We use VMWare to run a virtual machine with Ubuntu 16.04. Use RYU 4.15, Docker 17.03 ,Mininet and Snort 3.0 to set up the topology.

2) Process: For the controller, we use RYU. To meet the need of monitoring and forwarding, we rewrite the RYU controller.The controller runs on the virtual machine. We use Mininet to build the topology and to simulate the network. The three hosts in mininet are connected to a switch. For the DPI part, we use Docker as a container to run the snort. By starting a new Docker, we can easily run two snort instances. The port of each instance is added to the net bridge. We can see the state of the net bridge s1 in the Figure 3.Port 5 and 6 are the port of the two instances.

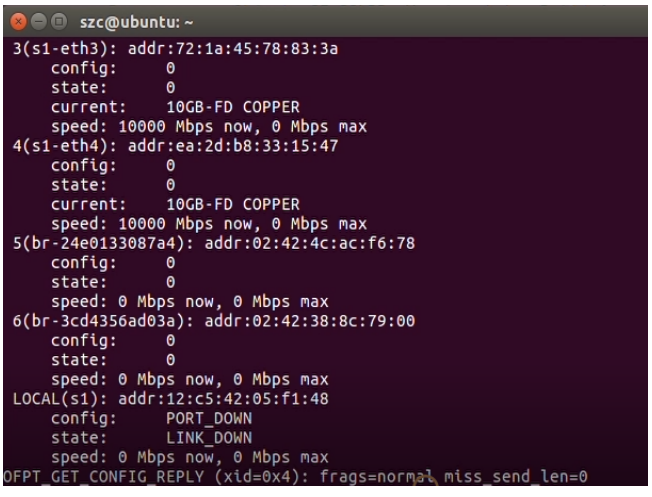

Fig. 3. Topology of vIDS.

And for the Docker, the most recent Docker Snort image is Snort 2.9.8. To meet the need of changing rules dynamically without reboot the snort, we need Snort version more than 2.9.9. So we make our own Docker image for the experiment. We set the Snort rules to alert the ICMP packet,so that we can use ping command to test the system easily.

3) Issues in Process and Solution:

- The first issue is about how to create a new instance.At first, we want to create a new VM to run Snort, and we don't know whether it is possible.After researching, we found that start a VM by another VM is not a good way to do this. So we ask Hongda about this issue. Following his advice, we choose Docker as a container to do this.
- The Snort instance in the Docker does not have a net port, it can not get the packet when host1 ping host2. We link the net bridge docker0 to the bridge of the topology s1. But it did not work.To solve this issue, we did research on the Internet. Finally, we can solve this by creating self-defined network and giving an ip address to the new Docker image.
- The third is about the forwarding problem. We successfully rewrote the controller code to add the rules using group and bucket to do the forward. But is did not work. It will send all the packet to one random destination. After going through the OVS community, we found that in the version of OpenFlow we use, this function is not fully implemented.It will decide the bucket by the destination's hash value, the same destination will result the same action.To solve this , we rewrite the flow-table though the controller dynamically.

VI. Experiment and Result

As shown on figure 4, we use mininet for the basic network. There is 3 hosts and one switch and 3 links between the switch and the hosts, also a controller.

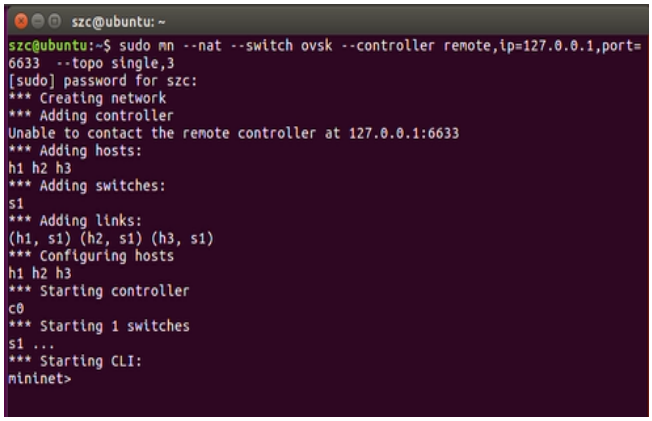

Fig. 4. Start mininet

We installed the snort application in the docker. So first, we need to start the docker to run the snort,as shown in the figure 5.

After the docker startup,we can see the rules in snort.

When we finished editing the rules,we can start snort.

It is necessary to add link the snort docker to the switch, so we need to add a port to s1 according to figure 8.

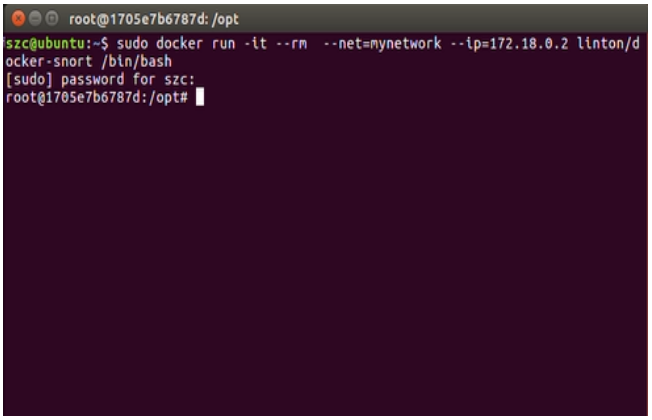

Fig. 5. Open docker

Fig. 6. View snort rules

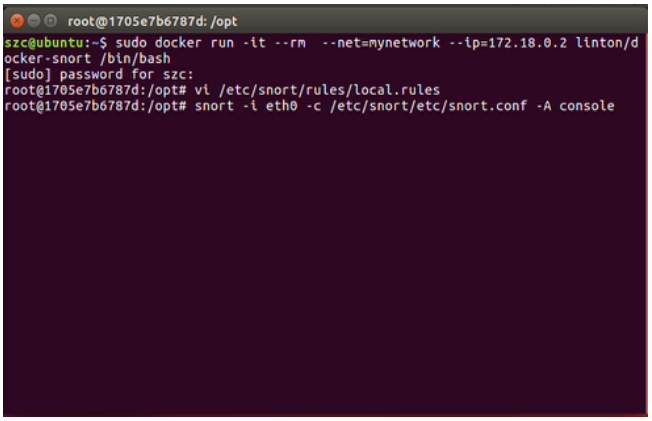

Fig. 7. Run snort

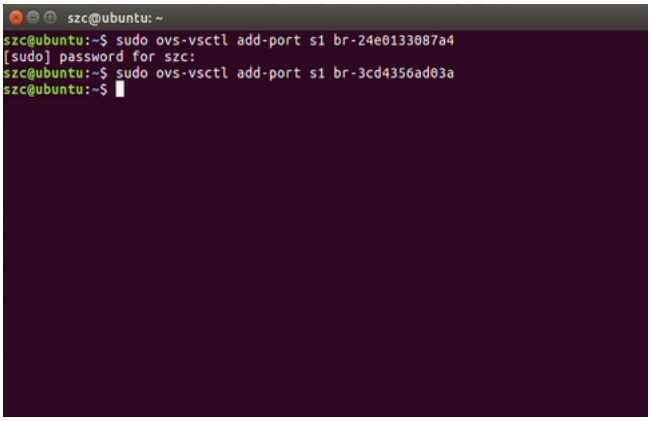

Fig. 8. Add port

There is 2 files need to be run in the ryu-manager,the ”mymonitor.py” and the ”simple switch 3.py” ,now we need to run the controller.

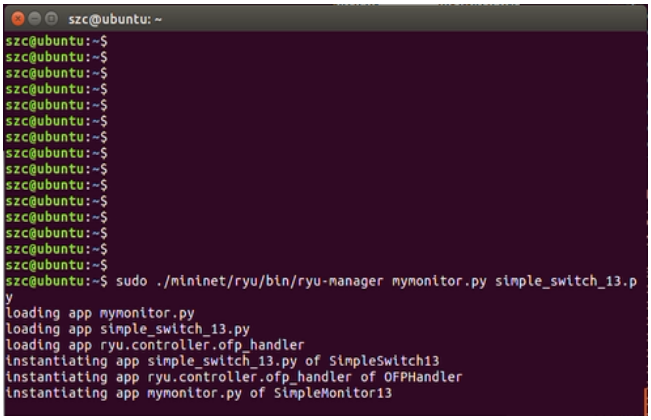

Fig. 9. Run controller

We use the ping command with different size packages to simulate the network traffic, In figure 10, when ping with small packages, with the progress keeps going,the switch and the snort0 doesn’t face any pressure, so the controller does nothing except shows the network condition information.

In figure 12, when switch the command to ping with big packages,a few seconds later,when the network traffic is over the limit we set in the program,the controller start to route the traffic to the snort1 in the other docker.which we can see from figure 13.

So the 2 snorts in 2 dockers are receiving information from the network,as is shown in figure 14.

When we stop the big package ping to simulate the decrease of the network traffic pressure,the controller stop to route the traffic to snort1. We can see this happen in figure 15.

After the controller stop to route the traffic to snort2, only snort 1 still shows the network package from in the console window, snort2 fully stopped. We can see this happen in figure 16 in detail.

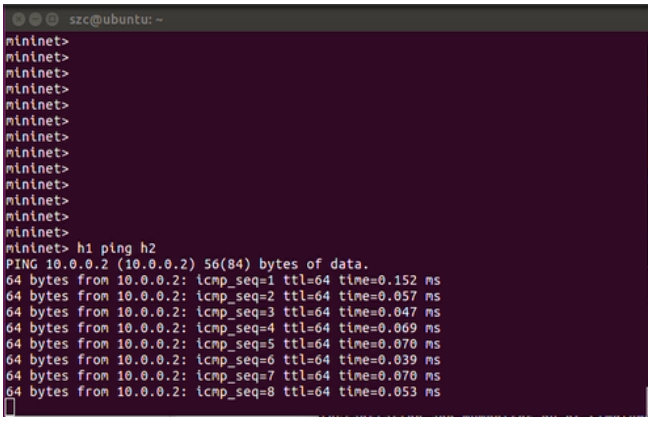

Fig. 10. Ping with small packages

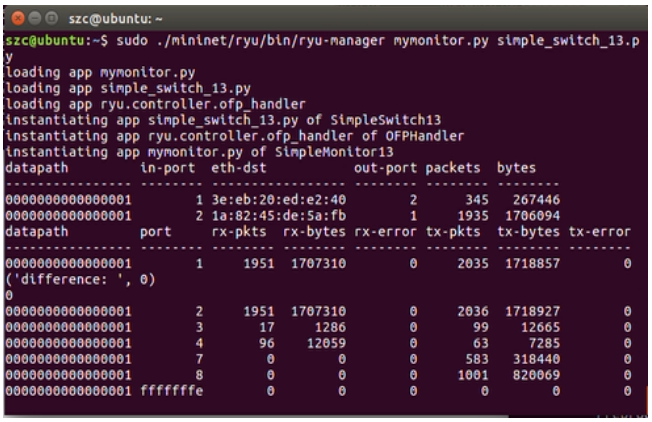

Fig. 11. Controller condition with no action

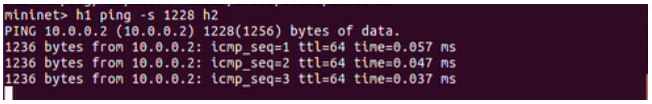

Fig. 12. Ping with big packages

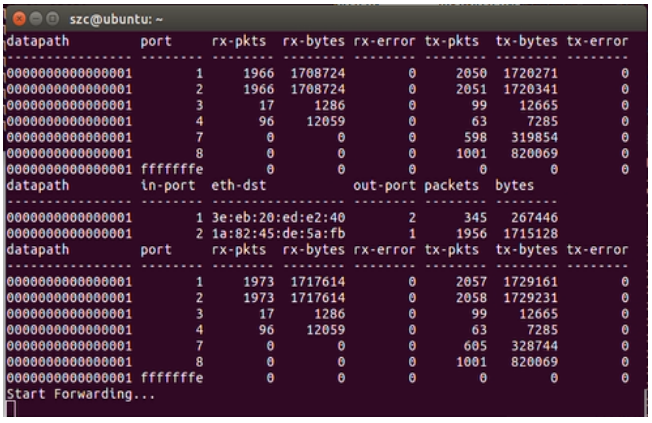

Fig. 13. Controller condition with action

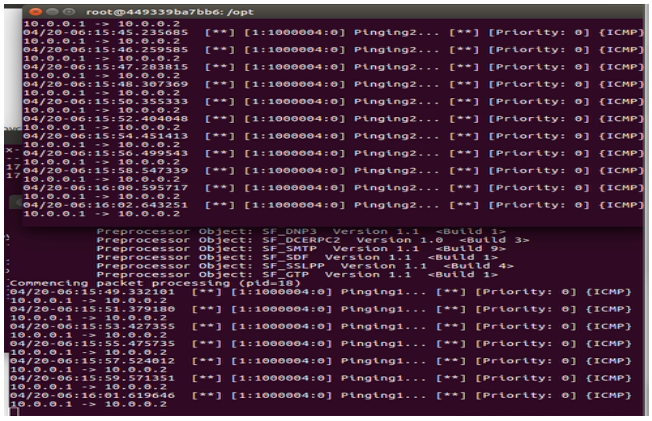

Fig. 14. Two snort running

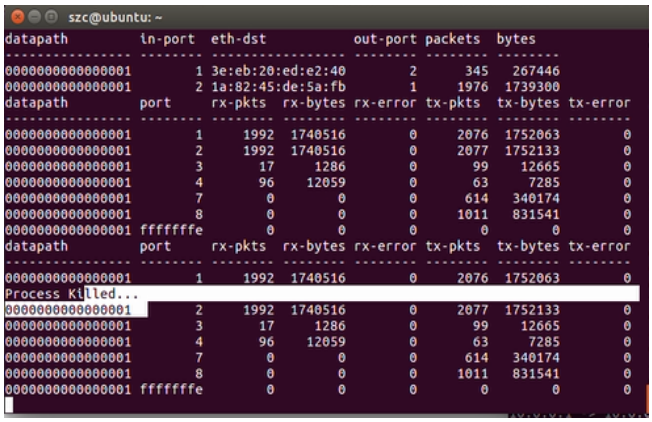

Fig. 15. Stop forewarding

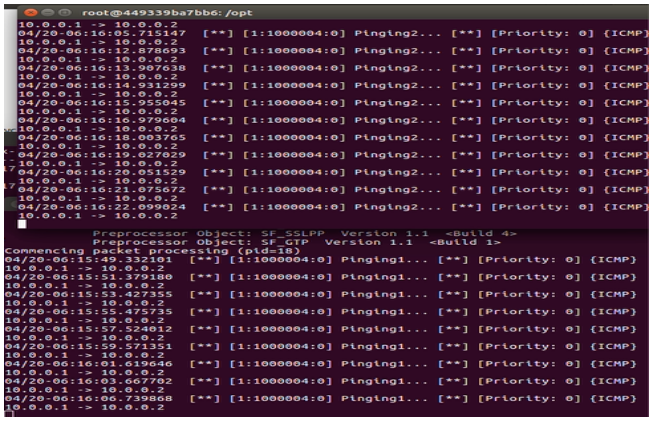

Fig. 16. Only snort1 receive packages

VII. Summary and Future Work

Although they are widely used in all kinds of networks because the important roles, middleboxes and monitoring tools is still expensive and hard to manage. Virtualization,SDN,NFV give a revolution in the way middleboxes and monitoring tools are managed.Monitoring applications shared lots of common tasks.To enhance the performance, lower costs,flexible and scalable design and the easy way to monitor applications.

Snort supports simple deep package inspection by string matching, and it is well known signature-based IDS.When the existing vIDS is overloaded, the controller can create a new vIDS instance.Once the program detects an overload of a vIDS instance, the program will forward packages to another vIDS instance and some of the traffic will be forwarded to the new vIDS instance.

In the future work, we can try to judge the network condition dynamically and scientifically. Also, It is essential to implement a high level api control package forwarding in the openvswitch. So that the controller pressure would reduce and have more resource to deal with other works.

VIII. Contribution

- Linyu Zhang: Decide the traffic rules to do the forwarding;To run the demo of the paper.
- Zhechao Sui: Make the Docker Snort image;Deploy the docker in the topology.
- Jie Hao: Setup the topology of the paper;Implement flow monitoring in vIDS.
- Biyang Fu: Implement Snort dynamic loading rules;Do the test of vIDS.

References

[1] Afek Y, Bremler-Barr A, Harchol Y, et al. Mca 2: multi-core architecture for mitigating complexity attacks[C]//2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 2012: 235-246.

[2] Aho A V, Corasick M J. Efficient string matching: an aid to bibliographic search[J]. Communications of the ACM, 1975, 18(6): 333-340.

[3] Intel Corp. Service-aware network architecture based on SDN, NFV, and network intelligence, 2014. http://www.qosmos.com/wp-content/uploads/2014/01/IntelQosmosDNNFV329290-002US-secured.pdf

[4] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying middlebox policy enforcement using SDN. In SIGCOMM, pages 27–38, 2013.

[5] Sun Wu and Udi Manber. A fast algorithm for multi-pattern 281 searching. Technical report, Chung-Cheng University, University of Arizona, 1994.

[6] Michela Becchi and Patrick Crowley. A hybrid finite automaton for practical deep packet inspection. In CoNEXT, page 1, 2007.

[7] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In SIGCOMM, pages 339–350, 2006.

[8] Bauer B, inventor; Intel Corp, assignee. Network traffic monitoring. United States patent application US 10/236,402. 2004 Mar 11.

[9] Shamosh R, Berlent HN, inventors; Berlent Harvey N, assignee. Security protection system and method. United States patent US 5,144,661. 1992 Sep 1.

[10] Provan KG, Kenis P. Modes of network governance: Structure, management, and effectiveness. Journal of public administration research and theory. 2008 Apr 1;18(2):229-52.

[11] Bugenhagen MK, inventor; Embarq Holdings Co LLC, assignee. System and method for dynamically shaping network traffic. United States patent US 7,808,918. 2010 Oct 5.

[12] Hare, Chris. "Simple Network Management Protocol (SNMP)." (2011).

[13] Bussiere R, inventor; Cabletron Systems Inc, assignee. Remote port mirroring system and method thereof. United States patent US 6,041,042. 2000 Mar 21.

[14] Bremler-Barr A, Harchol Y, Hay D, Koral Y. Deep packet inspection as a service. InProceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies 2014 Dec 2 (pp. 271-282).

[15] Roesch M. Snort: Lightweight intrusion detection for networks. InLisa 1999 Nov 7 (Vol. 99, No. 1, pp. 229-238).

[16] Pfaff B, Pettit J, Koponen T, Jackson E, Zhou A, Rajahalme J, Gross J, Wang A, Stringer J, Shelar P, Amidon K. The design and implementation of open vswitch. In12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) 2015 (pp. 117-130).

[17] Deshane T, Shepherd Z, Matthews J, Ben-Yehuda M, Shah A, Rao B. Quantitative comparison of Xen and KVM. Xen Summit, Boston, MA, USA. 2008 Jun 23:1-2.

[18] Peterson, B.D., 2012. Security Implications of the Cisco Nexus 1000V.

[19] Wang X, Hembroff GC, Yedica R. Using VMware VCenter lab manager in undergraduate education for system administration and network security. InProceedings of the 2010 ACM conference on Information technology education 2010 Oct 7 (pp. 43-52).

[20] Malik A, Ahmed J, Qadir J, Ilyas MU. A measurement study of open source SDN layers in OpenStack under network perturbation. Computer Communications. 2017 Apr 1;102:139-49.

[21] James TY. Performance evaluation of Linux bridge. InTelecommunications System Management Conference 2004.