UNISAFE: A Union of Security Actions for Software Switches

Taejune Park
KAIST, Republic of Korea
taejune.park@kaist.ac.kr

Yeonkeun Kim
KAIST, Republic of Korea
yeonk@kaist.ac.kr

Seungwon Shin
KAIST, Republic of Korea
claude@kaist.ac.kr

ABSTRACT

As Software-defined architectures, such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV), are getting popular, the necessity of software-based switch (a.k.a., software switch) is also increasing because it can adopt new functions/features without much difficulty compared with hardware-based switches. Nowadays we can easily observe that researchers devise new network functions and embed them into a software switch. However, most those proposals are highly biased at network communities, and thus it is hard to find some trials of leveraging the abilities of a software switch for security. In this paper, we consider that how we can enrich security functions/features in software-defined environments, and in this context we propose a new software switch architecture - with the name of UNISAFE - that can enable diverse security actions. Furthermore, UNISAFE provides *action clustering* which joins UNISAFE actions of multiple-flows together. It makes that UNISAFE can check flows synthetically, and thus a user can establish effective security policies and save system resources. In addition, we describe the design and implementation of UNISAFE and suggest some use-cases for how UNISAFE works.

1. INTRODUCTION

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are now considered as new networking paradigms for future networking environments. They are currently changing networking environments, and they provide many opportunities in devising new networking techniques. In addition, this paradigm change also requires the rapid adoption of new functions/features, and it leads us to employ another technology - software-based switch (a.k.a., software switch).

Unlike a hardware-based switch whose most functions are already embedded as a form of hardware logic, a software switch can be easily modified, and thus we can add any new functions/features without difficulty. Therefore, many systems such as virtual machines [15, 2] or cloud frameworks [10] have started employing software switches in their system.

However, although the popularity of a software switch is increasing rapidly, its usage pattern is quite restricted, specifically networking monitoring and management. In addition, most research projects about a software switch have highly focused on improving its performance [3]. As a result, software switches can show an impressive performance and usefulness as a switch.

Likewise, a software switch can embed diverse new features, and its performance is dramatically improved. Then, can we leverage this software switch in some other areas, such as security? Basically, a software switch does not include some security concepts, but we believe that its basic feature (and its surprising extensibility) can be used for security purposes. Of course, some researchers have already provided security functions to software switches. Avant-Guard [13] and Mekky *et al.* [7] are typical trials to include security functions in software switches. Avant-Guard implemented a connection migration and trigger modules into a programmable switch, and Mekky *et al.* suggested a method enabling customized packet handling. However, they are inflexible and inconvenient to deploy since they only detect specific attacks or need additional configurations.

To address this issue (i.e., enabling diverse security functions at a software switch), we devise new security actions[1] called UNISAFE, which is a union of security actions for software switches. A user can use those security actions similar with other OpenFlow actions, and those actions enable a software switch to easily provide fine-grained security services without deploying additional devices or applications. Furthermore, UNISAFE also supports an attractive function, *action clustering*, which clusters several UNISAFE actions as a single action. It enables multiple-flows to share one UNISAFE action, so that a user can establish effective security policies and save system resources.

In this paper, we describe the design of UNISAFE, and a prototype implementation of UNISAFE on Open vSwitch [9], which is one of the most popular software switches in these days. Furthermore, we evaluate our prototype UNISAFE to verify its feasibility and efficiency.

To this end, this paper provides the following contributions:

- We propose a union of security actions for software switches called UNISAFE, and it provides software-based

[1]Here, we mainly consider OpenFlow [6] based actions, such as `forward` and `set`, in enabling security functions, because OpenFlow is a de-facto standard protocol for SDN.

SDN-NFVSec'16, March 11 2016, New Orleans, LA, USA
© 2016 ACM. ISBN 978-1-4503-4078-6/16/03...$15.00
DOI: http://dx.doi.org/10.1145/2876019.2876025

network security functions as extended switch operations without any hardware device or application.

- We introduce *action clustering* which is a technique to logically integrate multiple UNISAFE actions into a single action. This technique realizes complicated security policies and even reduces the resource usage of the software switch.
- We implement a prototype of UNISAFE on Open vSwitch with three example UNISAFE actions: DoS Detector, Scan Detector, and Deep Packet Inspection. Our evaluation indicates that UNISAFE can provide flexible security services only with a small overhead.
- We provide several use cases to show how UNISAFE can be employed in some feasible examples using a well-known software switch. They demonstrate how our system is easy to use and simplifies complicated security policies.

2. BACKGROUND

Before explaining UNISAFE, we look into a software switch and its operational scenario by examining a well-known open-source software switch - Open vSwitch [9].

2.1 Software switch

A software switch is a switch platform which runs as software purely to provide various service environments easily, and it is introduced to overcome the limitations of legacy network devices (e.g., inflexible management and expensive maintenance). This software switch usually handles network flows with software modules, and it is commonly made up with a user-space daemon and a kernel-space module. The user-space daemon is in charge of taking new rules and sending them to the kernel-space module. The kernel-space module is in charge of processing traffic according to flow rules.

2.2 Open vSwitch

Open vSwitch(OVS) is one of the most popular software switches. It provides multi-layer flow tables and standard management interfaces to handle network packets efficiently. In addition, it supports diverse management protocols, such as NetFlow [8], sFlow [12] and IPFIX [5]. It has been included in some open-source Linux operating systems, adopted as a switch of the hypervisor for some virtual machines [15]. It is also used in various open-source projects such as OpenStack [10] and Docker [2].

OVS also consists of two main components; (i) ovs-vswitchd and (ii) kernel datapath module. We explain how these two components manage flow rules and handle network packets with those rules. When a new flow rule is inserted in OVS, the rule parser in ovs-vswitchd parses user's input. After parsing, the flow information is saved flow-tables of user-space at first. When a new packet (or a flow) arrives in OVS, OVS searches kernel flow-tables at first. If a matched flow rule is not found, OVS sends a message to the user-space module. If matched flows exist in the user-space table, OVS copies the flow rule information such as match field and actions field from the user-space to the kernel-space flow-tables. After copying, the packet (or the flow) is processed in the kernel-space. If OVS can find a matched flow in the kernel flow-tables, OVS updates statistic information and executes actions [11].

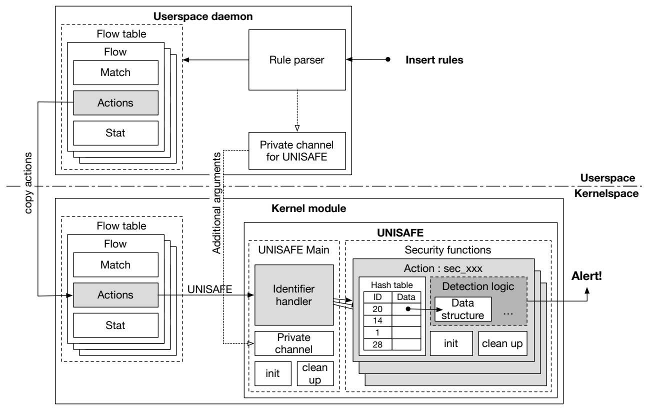


Figure 1: Overall Architecture of UNISAFE. After a flow rule is matched, the identifier handler calls a proper security action according to the action field of the matched rule. Each security action in UNISAFE is named as **sec_xxx**.

3. DESIGN

The main purpose of UNISAFE is to build a flexible and security-enhanced data plane by adding diverse security functions to a software switch. To achieve this purpose, we devise UNISAFE as a union of new security actions for software switches. These security actions can be used similar with other OpenFlow actions (e.g., `output, flood, drop`) and even make an action set clustering multiple actions. In this section, we present the overall design philosophy of UNISAFE at first, and then we describe each component of UNISAFE in detail.

3.1 Overall Design

As illustrated in Figure 1, UNISAFE employs two core modules: i) UNISAFE Main and ii) Security actions, and they are located in the kernel-space of software switches for providing diverse security services. Another module of UNISAFE is called a *private channel*, which transmits additional arguments from the user-space to the core modules in the kernel-space.

When a user installs a new flow rule, it is stored in the user-space at first, and then copied to the kernel-space when a matched flow arrives at the switch. If a flow rule has long arguments, UNISAFE copies them to the kernel-space directly rather than the user-space through the private channel to reduce processing overhead. When a UNISAFE flow rule, which is a flow rule having UNISAFE actions, is matched to incoming flows, UNISAFE sends this flow information and arguments to the UNISAFE *Main* module in order to provide desired security services located in the *security functions set* module.

3.2 UNISAFE Main

The UNISAFE main module initializes or cleans up UNISAFE instances when a software switch is launched or terminated. It also contains an *identifier handler* so that the software switch forwards received packets to a proper security action according to UNISAFE actions described in matched flow rules. In addition, this module has a *private channel*, which supports direct communication between the user-space and the kernel-space, to transmit arguments to security actions.

Identifier Handler: UNISAFE assigns a unique identifier to each UNISAFE action, and thus, a user (or an application) can inform a desired security service to UNISAFE by specifying a corresponding identifier in the action field of flow rules. When a received flow is matched to a UNISAFE flow rule, the flow information is forwarded to an *identifier handler* submodule rather than security actions directly. Then, the handler submodule forwards the information and arguments to desired security actions.

The reason why UNISAFE adopts this design is for enhancing programmability that is not supported by other hardware switches. Therefore, a developer can simply add a new security action by registering its identifier and source code files at the UNISAFE main.

Private Channel: Most of actions in a software switch require additional arguments that describe how to handle packets. Those arguments are copied from the user-space to the kernel-space when a target packet arrives at a switch. However, if the length of arguments is too long, followed packets would be delayed since the copy process requires a notable processing time. This can be quite critical for actions which have more complicated arguments such as an IDS or an IPS.

To resolve this issue, UNISAFE provides an alternative way to copy arguments. A *private channel* directly connects the user-space daemon with the UNISAFE main module in the kernel-space, hence some UNISAFE actions can receive their arguments through the private channel instantly before a new packet arrives. We describe an example case of using the private channel in Section 4.1.

3.3 Security Action Set

UNISAFE manages security actions according to their behaviors. Each action, which is written as **sec_xxx** in Figure 1, includes a detection logic and data structures that are managed by a hash table. The detection logic, as its name implies, detects a specific attack using status information of packets, and this information is stored in data structures separately. When a packet is passed to a security action, it looks up the hash table and updates related status information so that the detection logic finds a violation (i.e., attack) and alerts to a user if detected.

Data Structure: Each security action manages additional data structures to store data such as statistics information, policies, or rules to provide continuous and consistent security services. These data are handled by the hash table which the key of is a *cluster ID*. This is different with the above action identifier. It is used for *action clustering* which is described at the next section. Using these data, security actions can check security violations and sends an alert message to a user if UNISAFE detects any violation.

3.4 Action Clustering

The other attractive functionality of UNISAFE is *action clustering*. It logically integrates multiple UNISAFE actions into a single action so that UNISAFE can check different flows synthetically without installing additional flow rules. Therefore, the action clustering allows a user to employ comprehensive security policy and even conserve system resources.

UNISAFE uses a *cluster ID* of UNISAFE actions in each flow rule to notice which UNISAFE actions are logically integrated into a comprehensive action. When a user assigns an equal cluster ID to several UNISAFE actions, they share the same policies, status, or statistics information to verify a security violation inclusively. The cluster IDs are managed by a hash table in each UNISAFE action, thus, UNISAFE refers a cluster

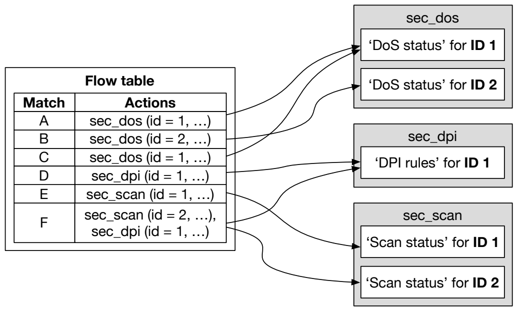


Figure 2: Example of Action Clustering. Multiple flows having the same cluster ID can share one UNISAFE action.

ID after calling a proper security action according to a type of UNISAFE action. Therefore, two flow rules which have the same cluster ID are grouped only if they have the same UNISAFE action.

Figure 2 describes an example of the action clustering. The first three rules have the same UNISAFE action but different cluster IDs. Since UNISAFE clusters separated actions according to their cluster IDs, Match A refers different status data with Match B but the same one with Match C. The next two rules have the same cluster ID but different UNISAFE actions. This implies that there is no correlation between Match D and E, so they are managed separately. The last rule is a complicated example. One UNISAFE action of Match F (i.e., *sec_dpi*) is clustered with Match D since they have the same cluster ID, but the other (i.e., *sec_scan*) does not compose any cluster with anyone since there is no equal cluster ID for that security action.

4. IMPLEMENTATION

We have implemented UNISAFE in the Open vSwitch version 2.4.90. At first, we have modified *ovs/lib/ofp-actions* so that a user can install UNISAFE flow rules into OVS through ovs-ofctl or ovs-dpctl. We designate the form of UNISAFE actions as **sec_*name*(args)**, which means that a UNISAFE action is identified as *name* and has *arg* as dedicated arguments to each UNISAFE action. For instance, *sec_dpi(rules)* denotes that a DPI action receives a rule file which is description for what it detects. Each UNISAFE flow rule has a *cluster ID* that is used for action clustering. If a user does not set a cluster ID in a flow rule, it is filled with a random unique number.

Next, we have modified some parts in *ovs/datpath* to implement the UNISAFE main and substantive security functions. Furthermore, we also have modified *ovs/ofproto/ofproto-dpif-xlate* to copy UNISAFE actions from the user-space to the kernel-space. Regardless of that, the private channel is implemented additionally using Netlink [4] at both the user-space and the kernel-space.

4.1 Example UNISAFE Security Actions

We have implemented three security actions for UNISAFE; DoS detector, Scan detector, and Deep Packet Inspection(DPI). **DoS detector** observes the bandwidth usage of flows by counting flow bytes. Whenever a packet arrives, DoS detector stores packet bytes accumulatively and a current time in each data structure based on the cluster ID of UNISAFE flow rules, supporting the action clustering. If the bandwidth of matched flows exceeds a given Mbps(Megabits per second), which is an argument defined by a user, then the action notifies the user of detecting a DoS attack.

Scan detector counts the number of activated TCP/UDP ports and decides a port scanning attack if the number of activated ports is larger than a given threshold during a time interval. For every incoming packets, Scan detector stores described port numbers and timestamps in data structures according to cluster IDs and deletes outdated information. If the action regards a network is under a port scanning attack, then it alerts a user.

Deep Packet Inspection inspects a packet payload to find specific patterns. Those patterns are described in a rule file by a user, so DPI accepts the path of the rule file as an argument and stores rule information in data structures. DPI uses Boyer-Moor algorithm [1] for the pattern matching process and alerts a user if a packet has any specified pattern.

Moreover, DPI is a good example of why the private channel is needed. Most of cases, the size of patter information is large enough to affect performance when it is copied to the kernel-space after arriving packets. However, UNISAFE can process packets immediately without any delay on other works since the private channel copies arguments instantly after installing a flow rule.

5. USAGE SCENARIO

In this section, we present several example usage scenarios to demonstrate how to apply UNISAFE with OVS. Since the basic usage of UNISAFE is not different with other OVS actions, a user can easily combine UNISAFE actions with other OVS actions.

5.1 Example Usage Scenarios

Case 1: security service chain

A service chain allows a switch to perform a set of actions sequentially. For instance, a user who wants to modify a source IP address and forward it can write the action field of a flow rule as below.

- *actions=mod_nw_src(...),output:x*

Similarly, UNISAFE supports the chaining of security actions, so if a user wants to make a service chain 'DoS detector $\rightarrow$ DPI', then an action field can be

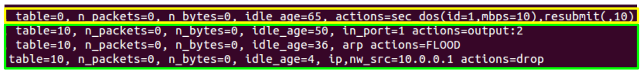
- *actions=sec_dos(...),sec_dpi(...)*

Thus, matched packets are processed in DoS detector first and then DPI. When DPI should be processed before DoS detector, the user simply changes the order of the chain.

Case 2: run actions *additionally* under the specific condition

We assume that a simple case that all flows are investigated by DoS detector while TCP flows are additionally inspected by DPI. Intuitively, those functionalities are represented by two flow rules: *actions=sec_dos(...)* and *actions=sec_dos(...),sec_dpi(...)*. However, they induces inaccurate functionality because UNISAFE creates independent DoS detector instances which monitor flows separately according to whether they are TCP flows or not. However, UNISAFE can solve this problem with the *action clustering*, defining flow rules as follows.

- *priority=10000, ,actions=sec_dos(**id=10**, ...)*
- *priority=10001, dl_type=ip, nw_proto=tcp, actions=sec_dos(**id=10**, ...),sec_dpi(...)*



```
 table=0, n_packets=0, n_bytes=0, idle_age=65, actions=sec_dos(id=1,mbps=10),resubmit(,10)
 table=10, n_packets=0, n_bytes=0, idle_age=50, in_port=1 actions=output:2
 table=10, n_packets=0, n_bytes=0, idle_age=36, arp actions=FLOOD
table=10, n_packets=0, n_bytes=0, idle_age=4, ip,nw_src=10.0.0.1 actions=drop
```

Figure 3: A OVS multiple flow table example for monitoring every packet.

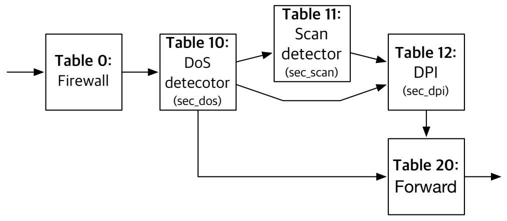


Figure 4: The example of the complex service chain.

Flows are matched to the first rule if they are not TCP flows, otherwise matched to the second rule. Since DoS detectors in both rule have the same cluster ID **id=10**, they are clustered and detect a DoS attack compositely while DPI still inspects only TCP flows.

Case 3: monitor every packet, not per flow

Due to the security service chaining and the action clustering, UNISAFE provides simple but strong security services to specific flows. However, if a user wants to verify all flows using UNISAFE with different actions, how can the user write flow rules? Should the user write flow rules for all possible cases?

To address this problem, a user can apply the other OVS action, `goto_table`. OVS provides multi-layer flow tables, which allow a user to consist layered switching steps. Therefore, the user can assign one flow table only for DoS detector instead of writing all possible flow rules, compressing the number of flow rules in a switch.

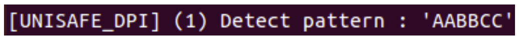
Figure 3 describes an example of multi-layer flow tables. The first flow rule matches all packets and executes DoS detector. After that, it calls a `goto_table` action which is displayed as `resubmit` in OVS and forwards packets to **table 10**. At table 10, OVS processes packets separately according to flow rules in the table.

Case 4: complex service chain with multi-layer tables

Combining and applying above use-cases, we show a feasible use-case using UNISAFE. Figure 4 illustrates the structure of this.

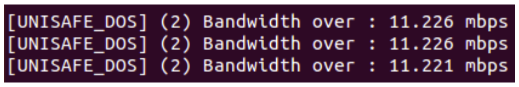
In this example, we compose a security service chain using several multi-layer flow tables, and each table has the responsibility for one function: Firewall (a set of `forward` and `drop` actions), DoS detector(*sec_dos*), Scan detector(*sec_scan*) and DPI(*sec_dpi*). When packets arrive at this switch, the Firewall table examines all packets at first and forwards it to the DoS detector table using the `goto_table`. Next, the DoS detector table checks a bandwidth about all packets using the **action clustering** and sends packets to the Scan detector table, the DPI table or the Forward table according to defined rules. The Scan detector table detects port scanning attacks (it there are) and relays packets to the DPI table. The DPI table inspects packets and forwards them to the Forward table. Finally, the Forward table is in charge of forwarding packets which is considered as valid packets by these service chain.

As seen from the above, UNISAFE is incredibly flexible, easy to use, and can be applied even to complex security



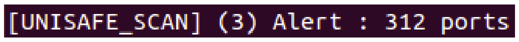
```
[UNISAFE_DPI] (1) Detect pattern : 'AABBCC'
```

(a) Alert message example of DPI



```
[UNISAFE_DOS] (2) Bandwidth over : 11.226 mbps
[UNISAFE_DOS] (2) Bandwidth over : 11.226 mbps
[UNISAFE_DOS] (2) Bandwidth over : 11.221 mbps
```

(b) Alert message example of DoS Detector



```
[UNISAFE_SCAN] (3) Alert : 312 ports
```

(c) Alert message example of Scan Detector

Figure 5: An examples of alert messages. When security violations are detected, UNISAFE alerts with the cluster ID and detail information.

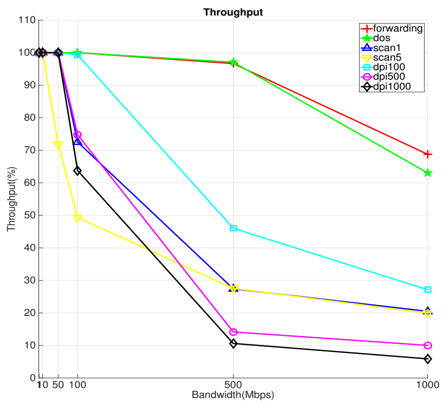


Figure 6: Throughput of UNISAFE

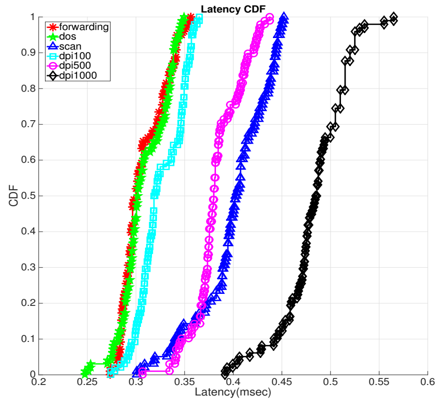


Figure 7: Latency of UNISAFE

systems. To build those systems without UNISAFE, a network requires other security applications and complicated configurations, raising additional operation costs. Whereas, UNISAFE can be instantly deployed to construct even complex systems without any additional cost.

5.2 Example Detection Cases

Figure 5 shows example cases when UNISAFE detects some suspicious network flows. UNISAFE generates alert messages with an action name, ID, and detail message and values. Currently, UNISAFE only displays alert messages in this prototype, but we will add much more interesting functions in the near future.

6. EVALUATION

In this section, we present some performance measurement our devised system to verify feasibility and efficiency of UNISAFE. We have tested UNISAFE at a test server which has Intel I7-4790 3.6GHz, 16Gb, and Realtek RTL 8169 PCI Gigabit Ethernet controllers. Using this system, we evaluate the throughput and the latency of three prototype UNISAFE actions: DoS detector, Scan detector, and DPI.

6.1 Throughput

To verify how UNISAFE affects the packet processing of software switches, we firstly evaluate the throughput of three prototype UNISAFE actions by connecting two hosts to UNISAFE and sending TCP packets from one host to the other. We record 100,000 random TCP packets and vary a bandwidth up to 1000Mbps using `tcpreplay`. We also perform the same measurement for a simple forwarding action (i.e., OVS without UNISAFE) to compare with each UNISAFE action.

Figure 6 describes the measured throughput of each prototype UNISAFE action. The red line with the *cross* dot means a simple forwarding for comparing other actions. It shows almost 70% throughput when a bandwidth is 1000Mbps. The result of DoS detector which has the green line with the $\star$ dot follows the simple forwarding even if throughput is just dropped at the 1000Mbps point. However, other actions fall short of the simple forwarding and DoS detector. The throughput of Scan detector whose time interval is 1sec(the blue line with the $\triangle$ dot) is already under 50% at the 500Mbps point. The other scan detector that a time interval is 5sec(the yellow line with the $\triangledown$ dot) is under 50% at the 100Mbps point. DPI with 100, 500, 1000 rules(each, the cyan line with the $\square$ dot, the magenta line with the $\bigcirc$ dot, and the black line with the $\diamond$ dot) are similar with Scan detectors. Even if they might keep acceptable throughput before the 100Mbps point, they are also decreased rapidly after the 500Mbps point.

6.2 Latency

Next, we evaluate the latency of UNISAFE actions using a ping message. We measure round-trip times (RTTs) by sending a ping message from one host to the other in 100 times, and calculate the average of them as the latency of each UNISAFE action, omitting the smallest and the largest RTT value.

Figure 7 illustrates the measured latency, and the meaning of each line is equal to the throughput result of Figure 6. We measure the simple forwarding again for comparing with others. 90% of packets are processed within 0.35ms. The latency of DoS detector is similar with the simple forwarding alike the throughput, but the latency of DPIs are delayed according to the number of rules. However, the delay time may be acceptable compared with throughput decline. Scan detector also shows reasonable result. We assume Scan detector observes 5000 ports when we measure latency, but it processes every packet within almost 0.45ms.

6.3 Discussion

Unfortunately, the throughput cannot show competent results except DoS detector. In comparison with throughput of the simple forwarding at the 1000mbps point, the throughput of DoS detector shows 94% of the simple forwarding, the throughput of Scan detector with 1sec and 5sec time interval shows 30% and 29%, and the throughput of DPI with 100, 500 and 1000 rules shows 40%, 15% and 8% levels.

However, the latency might be considered relatively reasonable. The latency gap between the best case(0.25ms of DoS detector) and the worst case(0.55ms of DPI 1000rules) is only 0.30ms.

Even if it may not be enough to deploy Unisafe in busy and huge network environments for now, we think that it may be enough for small networks because it is fine before 100Mbps point.

Note that Unisafe actions have **NOT** been optimized yet, and we only show the concept and possibility of Unisafe in this paper. However, we are under the improvement of performance, so we believe that Unisafe will show the reasonable performance as much as to be able to deploy in a real network topology soon.

7. RELATED WORK

Avant-guard[13] has proposed a secure OpenFlow switch architecture. It has solved the scalability challenge and the responsiveness using the connection migration and the actuating triggers. It can protect DDoS attack, Scanning attack, and other triggerable intrusion. However, it is not flexible as much as Unisafe because it can only protect specific attacks and it lacks programmability compared with Unisafe.

Mekky *et al.*[7] have proposed an extended SDN architecture that enables a fast customized packet handling. For this work, they have introduced *App table* which contains data-plane application information. However, due to App table, packets have to be matched more than two times, thus it decreases overall performance. However, Unisafe actions are executed after matching immediately, it does not need additional costs. In addition, they have implemented a prototype using OVS, but it runs at the user-space while Unisafe runs at the kernel-space which enables much faster packet processing.

OFX [14] is an OpenFlow extension for switch-level security applications. Previous approaches which try to add security function into SDN environments have implemented security applications at SDN controllers. They have shown poor performance or poor deployability. To address this problem, OFX has been suggested as a framework which enables installing security modules into switches. However, it relies on OpenFlow and an OpenFlow controller, hence it cannot run alone.

In addition, all researches mentioned above might not be easy and convenient to make service chains as much as Unisafe and might difficult to establish complex policies but Unisafe can do it easily thanks to the clustering.

8. CONCLUSION

In this paper, we describe the design and implementation of Unisafe which is a union of security actions for software switches. Unisafe enables software switches to easily provide security services without additional security devices or applications. As an example of Unisafe, we implement three Unisafe actions: DoS detector, Scan detector and Deep Packet Inspection, and we evaluate them. Since they are not optimized yet, the throughput degradation is a little big compared with a simple forwarding except DoS detector. However, we believe that Unisafe has the possibility to make networks more secure. We will improve and develop Unisafe continuously and add several new actions such as Anomaly detector or Session management as future works.

9. ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. B0126-15-1026, Development of Core Technologies for SDN-based Moving Target Defense)

10. REFERENCES

[1] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, Oct. 1977.

[2] Docker. Open-source project that automates the deployment of applications inside software containers. https://www.docker.com/.

[3] M. Honda, F. Huici, G. Lettieri, and L. Rizzo. mswitch: A highly-scalable, modular software switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 1:1–1:13, New York, NY, USA, 2015. ACM.

[4] IETF. RFC3549, Linux Netlink as an IP Services Protocol. http://www.ietf.org/rfc/rfc3549.txt.

[5] IPFIX. IP Flow Information Export. https://tools.ietf.org/html/rfc7011.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. In *Proceedings of ACM SIGCOMM Computer Communication Review*, April 2008.

[7] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman. Application-aware data plane processing in sdn. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 13–18, New York, NY, USA, 2014. ACM.

[8] NetFlow. Cisco IOS NetFlow. http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[9] Open vSwitch. An Open Virtual Switch. http://openvswitch.org/.

[10] OpenStack. Open source software for creating private and public clouds. https://www.openstack.org/.

[11] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association.

[12] sFlow. sFlow that an industry standard technology for monitoring high speed switched networks. http://www.sflow.org/.

[13] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCSâĂŹ13)*, November 2013.

[14] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith. Poster: Ofx: Enabling openflow extensions for switch-level security applications. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1678–1680, New York, NY, USA, 2015. ACM.

[15] Xen. Xen Wiki - Networking. http://wiki.xenproject.org/wiki/Xen_Networking#Open_vSwitch.