A Scalable Flow Rule Translation Implementation For Software Defined Security

Hao Tu[1], Weiming Li[1*], Dong Li[1],Junqing Yu[1,2]
Network and Computation Center[1]
School of Computer Science and Technology[2]
Huazhong University of Science and Technology
Wuhan, China
{tuhao, lwm, lidong, yjqing}@hust.edu.cn

***Abstract*—Software defined networking brings many possibilities to network security, one of the most important security challenge it can help with is the possibility to make network traffic pass through specific security devices, in other words, determine where to deploy these devices logically. However, most researches focus on high level policy and interaction framework but ignored how to translate them to low-level OpenFlow rules with scalability. We analyze different actions used in common security scenarios and resource constraints of physical switch. Based on them, we propose a rule translation implementation which can optimize the resource consumption according to different actions by selecting forward path dynamically.**

Keywords—network security; softwre defined networking

I. INTRODUCTION

Software Defined Networking (SDN) provides an abstraction of network devices and operations, enables flexible network policies by allowing controller applications to install packet-handling rules on a distributed collection of switches. It brings network security many possibilities, since current security devices are deployed on fixed position, but more advanced attacks, such as APT, use devices inside network to launch exploitation which cannot detect by security devices deployed on network edge. Software Defined Security (SDS) was proposed which present exploiting the SDN architecture to enhance network security. With SDN, the suspicious network packets can be processed more flexibly, providing of a highly reactive security monitoring, analysis and response system. The central controller is key to this system. Traffic analysis or anomaly-detection methods deployed in the network generate security-related data, which can be regularly transferred to the central controller. Applications can be run at the controller to analyze and correlate this feedback from the complete network. Based on the analysis, new or updated security policy can be propagated across the network in the form of flow rules.

Security as an advantage of the SDN framework has been recognized, a lot of work have been proposed to cooperation between security devices and SDN controllers [1, 2], but most of them focus on the framework and policy language while give limit consideration and information about how to translate them to OpenFlow[3] rules, like it is a simple problem and easy to handle.

But actually, maybe it is not so difficult in a small experimental network, but if we want process this from not theoretical view but feasible view, especially deploying SDS solution in large scale network, the following issues have to be considered:

- There are several different types of actions, i.e., mirror, redirection, drop, with different considerations when decide the forwarding tables. For example, it is common that sent a suspicious packet to IDS while do not want to interrupt the existed communication. In this action a packet need go through specific switch twice, but how switch know it should forward the packet to IDS or the original destination. A detail analysis of action demands of all possible scenarios is needed.
- Physical switches do have constraints like Ternary Content Addressable Memory (TCAM), CPU, interface bandwidth, etc. TCMA is the only way to identify the matching entries for each packet. However, TCAM is large and power hungry, which make it too expensive. The merchant chipsets in commodity switches typically support just a few thousand of entries which means we cannot add new rules infinitely.

These issues force us to consider many low-level details including the choice of path, the resource limits on each switch. We analysis the actions demand carefully and proposed a translation algorithm. And we implement a proof-of-concept system and experiment with POX and Mininet, the results show that it takes less than 100 ms to translate high-level policy to low-level flow rules in a network contains up to 1024 switches automatically, avoiding to manage the low-level resources.

In summary, this paper makes the following contributions:

- Presentation of the different forwarding templates for different actions by analyzing common scenarios in network security.
- Development of a novel path decision algorithm which minimize resource consumption for different actions based on the template and shortest path algorithm.
- A prototype implementation and preliminary performance evaluation of its overhead.

* Corresponding author.

II. BACKGROUND AND MOTIVATION

In this section we present a common example in network security scenario and explain the difficulty it faced.

Figure 1 shows a hacker attacks host A and gains control, then wants attack host B and C through host A. In order to detect and stop this kind of attack, firstly we need mirror the network traffic to IDS. Since one of the biggest problem is the false alert, sometimes we need double check, e.g. if host A attacks other host inside network, then it must be a real alert, which needs examine packets from A to B. It is difficult to deploy in current network, but using SDN, we can redirect the packets from host A to host B to the IDS, if there is no attack, then return it back to host B, otherwise drop them.

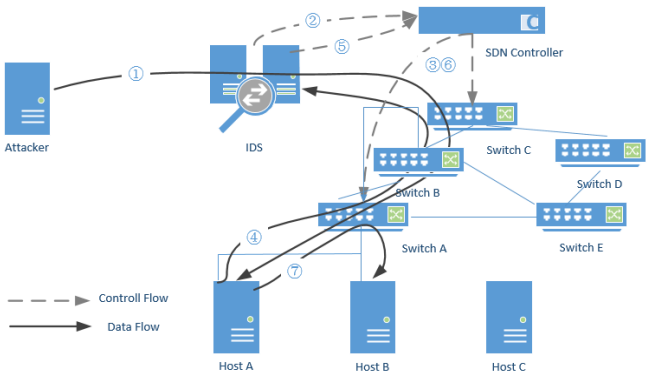

Fig. 1. Softwre defined security example

As we know, there are some different actions, mirror, redirection, drop, etc. However, the switches just forward the packets according flow tables, check the match fields, e.g., Ingress Port, Ether src and dst, Ether type, IP src and dst, TCP src and dst port, Vlan id and priority. Each action need a few flow rule operations and some of them need operate very carefully, e.g. IDS need mirror the packets, but which switch should be add rules to, from the switch near the source or the opposite? When switch A received the packets from host A to host B, should it forward them to switch B or host B?

Furthermore, as the figure 1 shows, most of the rule changes happened in few switches while others can do nothing. If it is a large network with thousands of hosts or more, this kind of operation will be frequent and plenty. The resource in some switches will be exhausted while the others will stay quite free. If we can migrate some of them to the free ones and keep the action results unchanged, it will let the SDS solution more scalable.

Though these issues can be resolved by careful design, they force programmers to reason about many low-level details. It will be much better if we can automatically translate high-level policy to low-level flow rules.

III. IMPLEMENTATION

As we saw in the previous section, how to generate flow rules from high level policy efficiently is not an easy task. To address this problem, we propose an implementation which take topology and resource into account in order to hide the detail and use the resource as balanced as possible.

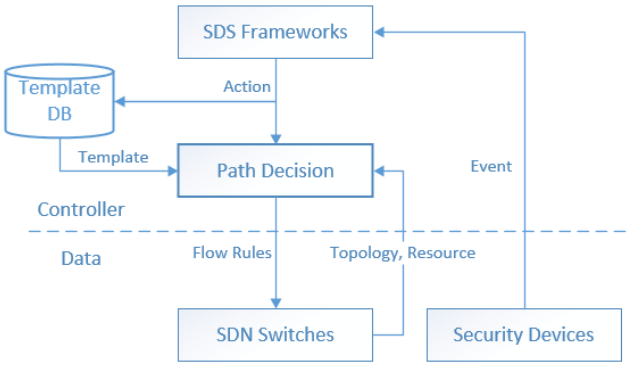

Fig. 2. System overview

As Figure 2 shows, firstly, we analysis the different action policy needed and exploit OpenFlow protocol to support them. Those analyses are generalized to a template db which can be used when new action arrive. At the same time, the device information was collected to obtain topology and resource, this can be done by SNMP reading. Now when action defined by high-level policy from security devices, the path decision module will decide the optimized path and generate flow rules.

A. Actions analysis

According different the network security applications, 5 kinds of actions are given:

- **Mirror**: Most DPI based application such as IDS/IPS need analysis the packets, but not all the packets they need can be collected on fix position, so it's an important action which duplicate the packets and send to security devices.
- **Redirection**: sent packet to security devices, which is used to send packets to fake system like honeypot.
- **MITM**: differ from Redirection, MITM not only sent packet to security devices, but also send them back to original destination, which cannot be found by attacker or user.
- **Drop**: simply drop the packet so the attack's packet cannot arrive the target. Usually used in prevent an attacker outside our network from access some node inside our network.
- **Quarantine**: not simply drop but only let them reach specific range.

As we see, the actions are much more flexible than current simple access or drop, but at the same time, they need us design the flow table carefully. Table I give a description of the different actions and examples used in common network security applications.

If Mirror action, we find out all path from the node in path N1, N3 to N5, and select the shortest path in them, which means the TCMA is least, bandwidth usability is high and the

hop is short. Note the TCMA is always be considered at the first priority, but the bandwidth and hop differ from different actions. For Mirror action, the hop is less important than bandwidth.

TABLE I. ACTION DESCREPTION

Action Type	Example	OpenFlow Feature
Mirror	N1, N3 -> N1, {N3 & N5}	Multi_action,
Redirection	N1, N3 -> N1, {!N3}, N6	In_port, priority
MITM	N1, N3 -> N1, N6, N3	In_port, priority
Drop	N1, {ALL} -> N1, {!N3}	priority
Quarantine	N3, {ALL} -> N1, {!(N4 & N5 & N6)}	priority

If Redirection action, we should let the packets go to N6 not N3, which need find out all path from the node in path N1, N3 to N6, and select the shortest one without N3.

The MITM is particular because it need return the packet to N3. That means for switch N1, at first it needs forward the packet to N6, and then it needs forward the packet to N3. Obviously, the switch will be confused since it cannot match two flow rules. Some research give tag-based solutions, but it need use some fields in flow header which may be used. Here we use two Ingress Port include flow rule instead the one only match destination IP address. If the packet from N1 to N3 comes from ingress port connect to N1, which means it is original output traffic and should forward to IDS, then forward it to N3; if the packet comes from ingress port connect to N3, which means it has been checked by the IDS and should be returned back, then forward it to N1.

Drop and Quarantine action is a little different from the above three. Drop needs add a drop rule before the packet arrive N3, so we can add this rule in any node in the path N1, N3, but if we need drop packet to a set of node, we should consider to combine them if possible, since it can reduce the rule numbers.

As we see, different actions need different rule generation methods and have different demands to resource. There are some OpenFlow features we exploited to support the actions. Firstly, the action supported one or more forward PORT set in one rule. It is useful in Mirror action. Secondly, the flow rule priority. Since matching starts at the first flow table and may continue to additional flow tables. Flow entries match packets in priority order, with the first matching entry in each table being used. If a matching entry is found, the instructions associated with the specific flow entry are executed. This feature is used to set a subset forwarding rule in a whole set, like redirection, drop action. Finally, IN_PORT field in matching field is important for those action which make a packet go through a switch one more time, like MITM.

All kind of action and some features it need store in a basic template DB, the features include resource constraints too. When a new action arrive, the DB will select proper path template for it. Of cause, we can add new template to the template DB if new kind of security application added, it is extensible.

B. Path decision

There are several resources we need consider, one of them is TCAM, which can read all rules in parallel to identify the matching entries for each packet. However, it is expensive and power hungry so cannot be used too much. The other is Bandwidth, since some actions may duplicate the packets or make the packet out and in, which double the bandwidth they needed. Some other resources (e.g. CPU/Memory) may be need to consider but just use bandwidth to present them, if needed, we can add them to our implementation too.

We want find a path which can meet the different actions' need and minimize the resource consumption at the same time. All the actions can be treated as a path decision problem which the path contains specific nodes.

So it is much different from traditional routing algorithm since traditional routing algorithm needs find a shortest path from the source node to the destination node, while our algorithm needs find a series node to deploy OpenFlow rules.

We can still use the shortest path solving idea, but not single-source shortest paths. We have to find specific shortest paths according to different actions.

```
1   for action in all_actions:
2     switch(action.type):
3       case Mirror:
4         for node in path[src_node, dst_node]:
5           do min_path(node, mir_node, mir_template)
6       case Redirection:
7         do min_path(src_node, dst_node,  red_template)
8       case MITM:
9         do p1 = min_path(src_node, mitm_node,
                          red_template)
10              P2 = min_path(mith_node, dst_node,
                          red_template)
11        for node in (p1 ∩ p2):
12          do set_port_tag
13      case Drop:
14        for node in path[action.src, action.dst]:
15          do min_path(node, dst_node, drop_template)
16    rule_generate(path, tag)
```

Fig. 3. Path decision algorithm

As the figure 3 presented, according to different action, the algorithm process differently. If mirror action, we try to find a node in the path from source node to destination node which has the shortest path to mirror node. This kind of idea is also used in drop or quarantine action. If redirection action, we just find a shortest path from the source node to new destination node by set a high priority rule to overcome the original one. While in MITM action, a packet may go through some switches more than once, so we need to set IN_PORT tag in rules to make the switches know how to forward the packet according the ingress port.

IV. EVALUATION

To analyze the performance overhead of our implementation, we deployed our prototype into a laboratory network Mininet. We use POX as the controller. The prototype was hosted on two 6-core Intel Xeon E7430 CPUs with 48 GB RAM and running a CentOS.

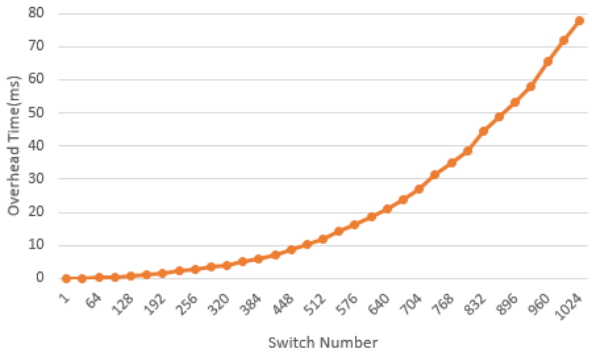

Fig. 4. Overhead

In Figure 4, we illustrate the results representing the computational delay required to conduct path decision analysis. As it shows, our implementation bring overhead to the controller, but when the network it handled is not too large, e.g. less than 512 switches, the overhead is acceptable. However, if the number large than 512, as the switch number increase, the overhead increase quickly. There are two possible reasons: one is because the complexity of shortest algorithm is proportional to the square of the node number, when the switch number become big enough, the complexity become much bigger; the other is the system overhead of Mininet, its simulation cannot afford too much switches. Though the experiment result is preliminary, it still shows it is possible to make the translation process more automatic and efficient by acceptable overhead.

V. RELATED WORK

Casado et al.[3,4] specifically considered the security aspects of a separate control and forwarding framework. SANE[4] focus on a logically centralized controller responsible for authentication of hosts and policy enforcement. Ethane[5] extended the work of SANE but required less alteration to the original network. Based on these work, OpenFlow was proposed.

OpenSAFE[1] presents a high-level policy language to manage the routing of traffic through network monitoring devices. Based on similar idea, CloudWatcher[6] focusing on SDN in the cloud and controls network flows to make specific network packets go through specific security devices. Fresco[2] presents an OpenFlow Security Application Development Framework. The idea behind FRESCO is to allow the rapid design and development of security application by some reusable modules, which can be incorporated as an OpenFlow controller application. But both above researches mentioned how to translate their high-level policy to low-level flow rules.

As the security application or device is a kind of network middle-box, some middle-box policy enforcement research is also considerable. The FlowTags[7] proposes the use of minimally modified middle-boxes, which interact with a SDN controller through a FlowTags Application Programming Interface (API). FlowTags, consisting of traffic flow information, are embedded in packet headers to provide flow tracking and enable controlled routing of tagged packets. SIMPLE[8] is an approach for using SDN to manage middle-box deployments. Neither SDN capabilities nor middle-box functionality modifications is needed, which makes it can be employed more easily than [7] for legacy systems. A disadvantage of them is that they focus on static policy, which means they cannot process dynamic actions.

VI. CONCLUSION

SDS is a hot topic in network security, most research pay more attention on high level policy and framework, manipulate the OpenFlow rules from theoretical view, while less on how to look realize it efficiently from feasible view. We propose a novel path decision algorithm based on action template and shortest path algorithm. Though there are many advanced problems need be researched, the preliminary result shows it can translate the high level policy to low level rule efficiently.

ACKNOWLEDGMENT

We are grateful for helpful comments from the anonymous reviewers to an earlier version of this paper. This work was supported by the National Science Foundation of China No.61370230 and the Fundamental Research Funds for the Central universities.

REFERENCES

[1] Ballard, Jeffrey R., Ian Rae, and Aditya Akella. "Extensible and scalable network monitoring using opensafe." Proc. INM/WREN, 2010.

[2] Shin, Seungwon, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Tyson. "Fresco: Modular composable security services for software-defined networks." Internet Society NDSS, 2013.

[3] 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 Computer Communications Review, April 2008.

[4] Casado, Martin, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. "SANE: A protection architecture for enterprise networks." In USENIX Security Symposium. 2006.

[5] Casado, Martin, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. "Ethane: Taking control of the enterprise." ACM SIGCOMM Computer Communication Review 37, no. 4 (2007): 1-12.

[6] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in 20th IEEE International Conference on Network Protocols (ICNP). IEEE, 2012, pp. 1–6.

[7] Fayazbakhsh, Seyed Kaveh, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. "FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions." In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp. 19-24. ACM, 2013.

[8] Qazi, Zafar Ayyub, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. "SIMPLE-fying middlebox policy enforcement using SDN." In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, pp. 27-38. ACM, 2013.