LPM: Layered Policy Management for Software-Defined Networks *

Wonkyu Han[1], Hongxin Hu[2] and Gail-Joon Ahn[1]

[1] Arizona State University, Tempe, AZ 85287, USA
{whan7,gahn}@asu.edu
[2] Clemson University, Clemson, SC 29634, USA
hhu@desu.edu

Abstract. Software-Defined Networking (SDN) as an emerging paradigm in networking divides the network architecture into three distinct layers such as application, control, and data layers. The multi-layered network architecture in SDN tremendously helps manage and control network traffic flows but each layer heavily relies on complex network policies. Managing and enforcing these network policies require dedicated cautions since combining multiple network modules in an SDN application not only becomes a non-trivial job, but also requires considerable efforts to identify dependencies within a module and between modules. In addition, multi-tenant SDN applications make network management tasks more difficult since there may exist unexpected interferences between traffic flows. In order to accommodate such complex network dynamics in SDN, we propose a novel policy management framework for SDN, called layered policy management (LPM). We also articulate challenges for each layer in terms of policy management and describe appropriate resolution strategies. In addition, we present a proof-of-concept implementation and demonstrate the feasibility of our approach with an SDN-based simulated network.

Keywords: Policy Management, Software-Defined Networking, Security.

1 Introduction

Traditional network environments are ill-suited to meet the requirements of today's enterprises, carriers, and end users. Software-Defined Networking (SDN) was recently introduced as a new network paradigm which is able to provide unprecedented programmability, automation, and network control by decoupling the control and data layers, and logically centralizing network intelligence and state [6]. A typical architecture of SDN consists of three distinct layers such as application, control, and data layers. Network applications in the application layer can communicate with an SDN controller via an open interface and define network-wide policies based on a global view of the network provided by the controller. The SDN controller, which resides in the control layer, manages network services, and provides an abstract view of the network to the application layer. At the same time, the controller translates policies defined by applications into actual rules for processing packets, which are identifiable by the data layer.

* This work was partially supported by the grant from Department of Energy (DE-SC0004308).

V. Atluri and G. Pernul (Eds.): DBSec 2014, LNCS 8566, pp. 356–363, 2014.

The multi-layered SDN architecture significantly helps manage and process network traffic flows. However, each layer of SDN architecture heavily relies on complicated network policies and managing those policies in SDN requires not only dedicated cautions but also considerable efforts. Our study reveals that such a multi-layered architecture brings great challenges in policy management for SDN as follows:

- **Policy management in SDN application layer:** An SDN application could employ multiple modules, such as Firewall (FW), Load-Balance (LB), Route, and Monitor, to process the same flow by composing rules produced by those modules [9]. However, such a task is not trivial since rules may overlap each other within a module (intra-module dependency) or between modules (inter-module dependency).
- **Policy management in SDN control layer:** In SDN control layer, there may exist multiple SDN applications running on top of a controller and they might jointly process the same traffic flow. In such a situation, flow rules from different applications that process the same flow may also overlap each other (inter-application dependency) and even lead to policy conflicts [10].
- **Policy management in SDN data layer:** In SDN data layer, different flows may go through the same switches and rules defining different flows in the same flow table may also overlap each other (intra-table dependency). In such a case, an unintended modification of a flow path could happen.

To address the above-mentioned challenges, we propose a novel framework for managing policies with respect to three layers in SDN architecture. In SDN application layer, we adopt a policy segmentation mechanism to compute and eliminate intra-module and inter-module dependencies, and enable a secure and efficient policy composition. In SDN control layer, our framework identifies inter-application dependencies and provides two kinds of resolution strategies. In addition, we propose a flow isolation mechanism to resolve intra-table dependencies in SDN data layer. We also provide a prototype implementation of our framework in an open SDN controller and evaluate our approach using a real-world network configuration and an emulated OpenFlow network.

This paper is organized as follows. Section 2 overviews our framework and presents policy management challenges and corresponding resolution strategies based on three layers of SDN architecture. In Section 3, we describe our implementation details and evaluation results followed by the related work discussed in Section 4. Section 5 concludes this paper.

2 Layered Policy Management (LPM) Framework

2.1 Overview

Our LPM framework enables a layered policy management with respect to three layers of SDN architecture as illustrated in Figure 1.

In SDN application layer, a main challenge comes from policy composition in an SDN application, where intra-module and inter-module dependencies should be addressed. Partially or entirely overlapped rules in a module make nontrivial intra-module

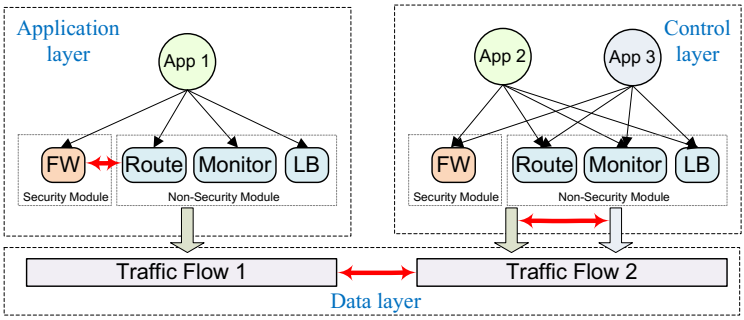


Fig. 1. Multi-layered SDN policy management: (i) application layer; (ii) control layer; and (iii) data layer

dependencies and make the process of policy composition more difficult. In addition, inter-module dependencies between security and non-security modules may cause security challenges due to inappropriate composition sequence and dynamic packet modification. Our framework addresses insecure and inefficient policy composition issues in an SDN application and adopts a policy segmentation mechanism to address those issues.

In SDN control layer, multiple applications in an SDN controller processing the same flow may cause inter-application dependencies. As shown in Figure 1, `App 2` and `App 3` intend to process the same flow `Flow 2`, thereby the policies produced by two applications may conflict with each other. Our framework also leverages the policy segmentation mechanism to eliminate the conflicts and applies two resolution strategies by allowing them to jointly process the same flow or assigning dependent applications with different priorities to break inter-application dependencies.

In SDN data layer, each physical switch stores a number of flow rules with corresponding priorities into its flow table. A rule defining one flow, such as `Flow 1` in Figure 1, with a lower priority might be affected by another rule for another flow, such as `Flow 2` in Figure 1, with a higher priority, causing intra-table dependency. Since intra-table dependency might change the behaviors of associated flows, our framework provides a flow isolation mechanism to address such an issue.

2.2 Policy Management in SDN Application Layer

Considerations and Challenges. While an SDN application with multiple modules processes a network traffic flow, a fundamental consideration is to address intra-module and inter-module dependencies in policy composition. To illustrate these issues, we adopt two kinds of policy composition operators introduced in [9]. "*Parallel*" operator ($|$) means the *union* of two modules and generates a set of packet processing rules which should be applied to the same flow simultaneously. "*Sequential*" operator ($\gg$) stands for the serialization of modules so that the matching rules would be performed one by one on the same flow. We next investigate several policy management challenges in SDN application layer.

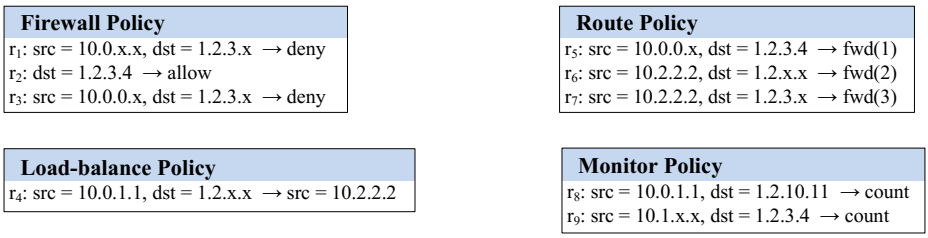


Fig. 2. Sample policies defined by four different network modules

(1) *Intra-module and inter-module dependency*: Assume that there exist four different modules in an SDN application, such as Firewall (FW), Load-Balance (LB), Route, and Monitor, and all rules in each module have been sorted by their priorities as depicted in Figure 2. In FW policy, r_1, r_2, and r_3 are mutually dependent, and r_2 and r_3 are partially overlapped by r_1, representing intra-module dependencies. In addition, the rule r_1 in FW policy is dependent with both r_4 in LB policy and r_5 in Route policy, representing inter-module dependencies. Furthermore, since r_2 in FW policy is dependent with r_9 in Monitor policy, FW module is dependent with all other modules which implies that determining inter-module dependencies requires considerable efforts.

(2) *Insecure and inefficient policy composition*: Suppose that two modules in an SDN application are sequentially composed, represented as $LB \gg FW$. In this case, r_4 in LB policy modifies packets' source IP address to 10.2.2.2 and could enable malicious packets to bypass the firewall since r_1 in FW policy cannot block these packets. Hence, we could notice that an inaccurate sequence during the composition may cause security breaches in SDN applications. In addition, a programmer may want to compose two modules in parallel, such as $FW \mid Route$. We could observe that all rules in FW policy are dependent with r_5 in Route policy. Since r_1 has the highest priority, r_1 and r_5 can be jointly combined and the following rule can be obtained: src = 10.0.0.x, dst = 1.2.3.4 $\longrightarrow$ $deny, fwd(1)$. Indeed, r_5 is not necessary to be composed with FW rules, since the FW rule r_1 ultimately blocks packets matching the rule pattern. Therefore, we could also observe that it is obviously inefficient to *always* compose the multiple policies as Pyretic [9] does.

As discussed above, there exist a few challenges in the application layer. First, since the security policies are generally considered more important than the policies produced by non-security modules, distinguishing security modules from non-security modules is vital in composing secure policies. In addition, commodity SDN switches typically support only a few thousands of rules [12], hence we should also strive to provide mechanisms with respect to an efficient policy composition.

Resolution Strategy. Our resolution approach globally examines all modules along with their rules to identify overlapping rules and generate disjointed matching space for removing intra-module and inter-module dependencies. To eliminate these depen-

dencies, we first sort the rules in each module by their priorities and insert all modules into a global segmentation table. Derived from the approach discussed in [8], our policy segmentation mechanism generates a set of disjointed matching space, called *segment*. For example, r_1 and r_2 in FW policy are partially dependent with each other. Thus, we obtain three disjointed segments: $s_a = r_1 - r_2$, $s_b = r_2 - r_1$, and $s_c = r_1 \cap r_2$. Each segment maintains overlapping rules, which indicate the existence of intra-module or inter-module dependencies.

Regarding intra-module dependencies, not all overlapping rules from the same module in a segment are effective, since only one of those rules with the highest priority will be applicable to process matching packets. Therefore, to remove intra-module dependencies, we only need to consider the *effective* rule for policy composition. However, inter-module dependencies between security and non-security modules may cause insecure and inefficient policy composition as discussed above. To address such an issue, we distinguish security modules from non-security modules using a separator (:), which indicates that its left-hand side refers security modules with higher priorities while non-security modules are located on the right-hand side with lower priorities. At the same time, to achieve an efficient policy composition, our resolution only enables to composing *allow* rules from security modules with other rules from non-security modules since it is unnecessary to perform policy composition once a rule from security modules denies the matching space. For instance, the composition sequence, $LB \gg FW$, is not valid in our scheme since the separator (:) would distinguish security modules and non-security modules, i.e., $FW : LB$. In addition, our mechanism does not compose r_1 in FW policy with r_4 in LB policy. Because r_1 is a *deny* rule, our mechanism simply generates a *deny* flow entry without considering overlapping rules from non-security modules.

2.3 Policy Management in SDN Control Layer

Inter-application Dependency. The root cause for inter-application dependencies is that multiple SDN applications may attempt to enforce their policies over the same network flow. Suppose that `APP 2` in Figure 1 composes LB, Route and Monitor modules sequentially, $LB \gg Route \gg Monitor$. However, `APP 3` composes the same modules in the opposite order, $Monitor \gg Route \gg LB$. Incoming packets matching the source IP address 10.0.1.1 and the destination IP address 1.2.10.11 will be managed by two different applications, since both r_4 in LB policy and r_8 in Monitor policy can handle these packets. The `APP 2` first applies r_4 in LB policy to modify the source IP address of matched packets to 10.2.2.2 and then applies r_6 in Route policy to forward them to port 2. Note that there is no matching rule in Monitor policy. On the other hand, the `APP 3` first enforces r_8 in Monitor policy to count the packets of the same flow and then drops the matched packets because there is no matching rule in Route policy.

Resolution Strategy. In our resolution approach, we consider two situations: (i) different applications are allowed to jointly manage the same flow and (ii) applications are mutually exclusive. For the former case, we may allow inter-application dependencies and apply composition operators to combine multiple policies from different

applications. With respect to the latter case, we eliminate inter-application dependencies by assigning different priorities to conflicting applications. Then, the application with the highest priority overrides the applications with the lower priorities when the flows are processed. For example, an application that employs security modules may have a higher priority to take the precedence over other normal applications. Different conflict resolution strategies proposed by our previous work [8] are also applied to resolve inter-application dependencies caused by conflicting applications.

2.4 Policy Management in SDN Data Layer

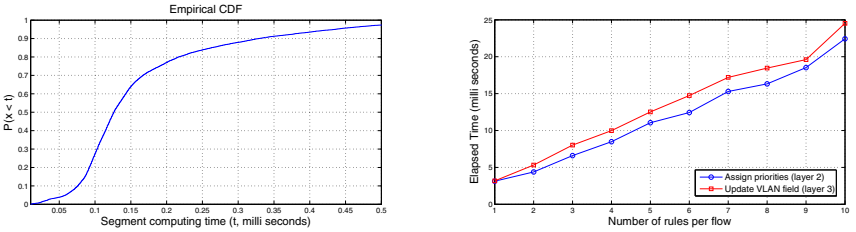
Intra-table Dependency. The flow paths of distinct flows managed by different SDN applications may overlap each other in the flow tables, introducing intra-table dependencies. For example, suppose that there exist two traffic flows processed by different applications as shown in Figure 1. One application, `App 1`, generates a policy for a flow `Flow 1`, which matches packets whose source and destination IP addresses are 10.2.2.2 and 1.2.3.4 respectively and forwards the packets to the port 2. On the other hand, another application `App 2` manages a different flow `Flow 2`, but the generated policy modifies the source IP address of matched packets to 10.2.2.2 and forwards the packets to the port 2. Even though incoming packets of two flows might be different, outgoing packets for those flows may overlap with each other. Thus, this situation may cause a potential loss of flow control for an application if there exists an intra-table dependency between the flow paths.

Resolution Strategy. Our resolution approach for this layer is to remove intra-table dependencies through *flow isolation*. Inspired by the approach discussed in [7], which leverages tags to differentiate packets belonging to different versions of policies for enabling consistent network updates, we also utilize tags to eliminate the dependencies in a flow table. Using this strategy, a new flow policy is preprocessed by adding a tag to distinguish the matching pattern with other policies. The rule of the flow policy in the ingress switch will take additional action on the packets to label them with the same tag. When the packets leave the network, the corresponding rule of the flow policy in the egress switch will remove the tag from the packets.

3 Implementation and Evaluation

We have implemented our framework on top of an open SDN controller, Floodlight [1]. Our proof-of-concept implementation captures every flow rule created by applications and produces a set of segments, which are able to remove intra-module and inter-module dependencies. Also, our resolution strategy component obtains a global view of network from the Floodlight controller and utilizes various resolution strategies for SDN control layer and data layer.

Our experiments were performed with Floodlight v0.90 and Mininet v2.1.0 [3]. We obtained a *real-world* network configuration from Stanford backbone network [2], which has 26 switches with corresponding ACL rules. We removed redundant ACL rules, converted them to a FW policy, and in turn obtained 1,206 FW rules in total.



(a) Segment generation time (CDF) (b) Elapsed time for resolution strategies

Fig. 3. Experimental results of our approach

At the same time, we generated $8,908$ Floodlight-recognizable flow rules by parsing original network rules in Stanford network configuration. Because these real network rules perform both routing and load-balancing tasks, we assume that these rules are generated by two modules, *Route* and *LB* modules.

To measure overheads caused by our policy segmentation mechanism, we installed all network rules into the simulated network and measured the update operation of policy segmentation. Our experiments show that 456 segments out of 688 FW rules were produced by the policy segmentation mechanism and $8,273$ segments out of $8,908$ network rules were generated. As shown in Figure 3(a), 75% of updates were completed within 0.2 milliseconds and most of cases (98%) were computed in less than 0.5 milliseconds.

We also evaluated the performance of two resolution strategies: (i) assigning priorities to eliminate inter-application dependencies in the SDN control layer and (ii) updating VLAN fields (flow-tagging) to eliminate intra-table dependencies in the SDN data layer. Both resolution strategies update a set of flow rules which define corresponding flows. First, we measured elapsed time for assigning priorities of rules. As shown in Figure 3(b), the elapsed time grows in accordance with the increased number of rules per each flow. Similarly, we checked the elapsed time for updating VLAN fields for isolating conflicting flows. The elapsed time increases with the increased number of rules per each flow, as expected. However, it generally took more time, since our mechanism needs to figure out ingress and egress switches in examining flows, adding and striping VLAN tags from the packets.

4 Related Work

Modular network programming has recently received considerable attention in SDN community. For instance, Pyretic [9] enables a program to combine different policies generated by different modules together using policy composition operators. However, due to the lack of policy dependency detection mechanism in Pyretic, it is obviously inefficient to *always* compose the multiple policies and install them into the network switches. FRESCO [11] deals with security application development framework using modular programming for SDN, but it cannot directly handle dependencies between modules in SDN applications either. Several policy composition mechanisms

such as [4,5] support *pair-wise* composition for access control policies and could be potentially utilized to deal with intra-module dependencies in SDN. In contrast, our framework addresses *various* dependencies including intra-module, inter-module, inter-application, and intra-table dependencies in SDN.

5 Conclusion

We have articulated numerous problematic issues and security challenges in SDN policy management and proposed a novel framework to facilitate a layered policy management approach with respect to three layers in the SDN architecture. Our experimental results with the proof-of-concept prototype showed that our resolution is efficient and only introduces manageable performance overheads to the networks. For the future work, we will extend our framework to support dynamic policy updates. In addition, we would expand our solution to support comprehensive SDN policy management considering heterogeneous and distributed controllers.

References

1. Floodlight: Open SDN Controller, `http://www.projectfloodlight.org`
2. Header Space Library, `https://bitbucket.org/peymank/hassel-public`
3. Mininet: An Instant Virtual Network on Your Laptop, `http://mininet.org`
4. Bandara, A.K., Lupu, E.C., Russo, A.: Using event calculus to formalise policy specification and analysis. In: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 26–39. IEEE (2003)
5. Bonatti, P., De Capitani di Vimercati, S., Samarati, P.: An algebra for composing access control policies. ACM Transactions on Information and System Security (TISSEC) 5(1), 1–35 (2002)
6. ONF Market Education Committee, et al.: Software-defined networking: The new norm for networks. ONF White Paper. Open Networking Foundation, Palo Alto (2012)
7. Fayazbakhsh, S.K., Chiang, L., Sekar, V., Yu, M., Mogul, J.C.: Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014), pp. 543–546. USENIX Association (2014)
8. Hu, H., Ahn, G.-J., Kulkarni, K.: Detecting and resolving firewall policy anomalies. IEEE Transactions on Dependable and Secure Computing 9(3), 318–331 (2012)
9. Monsanto, C., Reich, J., Foster, N., Rexford, J., Walker, D.: Composing software-defined networks. In: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, pp. 1–14. USENIX Association (2013)
10. Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., Gu, G.: A security enforcement kernel for openflow networks. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, pp. 121–126. ACM (2012)
11. Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., Tyson, M.: Fresco: Modular composable security services for software-defined networks. In: Proceedings of Network and Distributed Security Symposium (2013)
12. Stephens, B., Cox, A., Felter, W., Dixon, C., Carter, J.: Past: Scalable ethernet for data centers. In: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012), pp. 49–60. ACM (2012)