A Fine-Grained Detection Mechanism for SDN Rule Collision

Qiu Xiaochen[1], Zheng Shihui[1(✉)], Gu Lize[1], and Cai Yongmei[2]

[1] College of Cyberspace Security,
Beijing University of Posts and Telecommunication, Beijing, China
shihuizh@bupt.edu.cn
[2] College of Computer Science and Engineering,
Xinjiang University of Finance and Economics, Urumqi, China

Abstract. The rules issued by third-party applications may have direct violations or indirect violations with existing security flow rules in the SDN (software-defined network), thereby leading to the failure of security rules. Currently, existing methods cannot detect the rule collision in a comprehensive and fine-grained manner. This paper proposes a deep detection mechanism for rule collision that can detect grammatical errors in the flow rules themselves, and can also detect direct and indirect rule collisions between third-party and security applications based on the set intersection method. In addition, our mechanism can effectively and automatically resolve the rule collision. Finally, we implement the detection mechanism in the RYU controller, and use Mininet to evaluate the function and performance. The results show that the mechanism proposed in this paper can accurately detect the static, dynamic and dependency collisions of flow rules, and ensure that the decline of throughput of the northbound interface of the SDN network is controlled at 20%.

Keywords: Software-defined network · OpenFlow · Flow table · Collision detection and resolution

1 Introduction

The software-defined network (SDN) [1] is a new type of network architecture proposed by Clean State research group of Stanford University, which brings a tremendous change to the traditional network. The essential features of SDN are the separation of control and data plane and an open network programmable ability. In 2008, the research team led by Professor Nick McKeown of Stanford University proposed the concept of OpenFlow. The OpenFlow protocol [2] is currently the mainstream southbound communication protocol of SDN. It defines the communication standard between SDN controllers and OpenFlow switches.

With the development of SDN technology, the security issues of SDNs have become increasingly clear. Our research primarily focuses on the security threat to the SDN's application layer, which is called the flow rule collision [3, 4]. The openness and programmability of SDNs allow a large number of third-party applications, including security applications, run on the network concurrently. Because the logic and

S. Liu and G. Yang (Eds.): ADHIP 2018, LNICST 279, pp. 549–559, 2019.
https://doi.org/10.1007/978-3-030-19086-6_60

functions of these applications are different, they will issue different flow rules according to their needs, and there will inevitably be rule collision.

For the abovementioned threat of rule collision, the Porras [6] research group from the United States designed a security-enhanced kernel for the SDN operating system in the Nox controller [7], which is called FortNox, and proposed a policy collision detection method based on the alias sets algorithm. The source IP address and destination IP address in the flow entry are put into two sets, and the addresses before and after the modification in the Set-Field flow entry are also added to the above two sets. Finally, the source address and destination address sets are compared with firewall rules in the SDN to discover the policy collision. FLOVER [8] is a new detection system that can detect whether a flow policy deployed in an OpenFlow network violates the security policies. This system can detect invalid and invisible routes due to errors, but it does not consider firewall policies. Reference [5] proposed a method based on "First-order" logic to detect rule collision, and deployed an "inference engine" to detect rule collision before issuing the flow rules. Wang Juan et al. proposed a real-time dynamic policy collision detection solution based on FlowPath [9], which detected direct and indirect collision using the real-time state of the SDN. Reference [10] proposed to use ADRS (anomaly detecting and resolving for SDN) to solve the anomalies in the policies and rules, used the interval tree model to quickly scan the flow table, and established a shared model to allocate network priorities. Through these two models, they proposed an automated algorithm to detect and resolve collisions. The Khurshid team [11] proposed a new type of inspection tool named Veriflow. The tool is located between the controller and the network equipment, and creates a Tire tree by coding to simulate the forwarding of the new rules after they are added, thus realizing network detection.

The aforementioned solutions are not comprehensive in terms of rule collision detection. Fortnox's solution based on the alias sets algorithm can only detect simple dynamic collision and intercept them, and only the source and destination IP address of the OpenFlow field are considered in the dependency collision detection. Therefore, this paper will propose a more complete and fine-grained rule collision detection mechanism with automated collision solution. The new mechanism can fully detect the grammatical errors existing in the flow rules themselves, thereby preventing excessive invalid rules that are causing redundancy in the flow table. In addition, it can also detect the fine-grained direct and indirect violations between the rules from third-party application and security application.

2 OpenFlow Rule Collision Representation

2.1 OpenFlow Protocol

The OpenFlow protocol indicates that each OpenFlow switch can have multiple flow tables, each table may contain multiple flow entries, which describe how to match and process data packets arriving at the switch. The flow entry consists of three main parts: "match fields", "counters" and "actions". The "match field" is used to define the information of the packets that needs to be matched. The "counter" field is used to

count the number of packets processed in this flow entry. The "actions" represent the actions to be performed on data packets that match this flow entry, including "forward", "discard", "modify" and other operations.

Next we formalize the flow table in an OpenFlow switch, all the flow rules *F(j)* in the *j-th* OpenFlow switch in SDN network data plane are formalized as:

$$F(j) = F_1, F_2, F_3 \ldots F_n \tag{1}$$

Each flow rule F_i consists of the matching domain C_i, the priority P_i and the action field A_i, n is the number of flow rules, it may be defined as follows:

$$F_i = C_i, P_i, A_i (1 \leq i \leq n) \tag{2}$$

$$C_i = (f_1, f_2, f_3 \ldots f_n) \tag{3}$$

In the matching domain C_i, f_1, f_2, f_3 ... f_n respectively represent the header field of OpenFlow protocol. OpenFlow1.0 protocol contains 12 header fields: in_port, dl_src, dl_dst, dl_type, dl_vlan, dl_vlan_pcp, nw_src, nw_dst, nw_proto, nw_tos, tp_src, tp_dst. In the new version of OpenFlow protocol, the number of match fields is also increasing.

2.2 Classification of Flow Rule Collision

Reference [5] defines two types of flow rule collisions: static and dynamic collisions. A static collision refers to an internal collision in the rules themselves which have the wrong parameters or error syntax. A dynamic collision refers to a collision among flow entries, in a flow table where two or more flow entries match with one data packet at the same time.

In addition to the two types of rule collisions mentioned above, we call the flow rule whose "actions" field contains the content of rewriting the data packets the Set-Field rule. An attacker can issue malicious Set-Field rules that could rewrite the packet header that arrives at the switches, and the dependency relationship between the Set-Field rules may cause the security rules to be invalid. This collision is called a dependency collision.

As shown in the SDN topology in Fig. 1, the network contains three switches, one controller, and four hosts. The security rule in the SDN controller network is that: Host A (10.0.1.12) to C (10.0.13.12) cannot communicate. If an attacker issues the following three Set-Field rules:

1. S1:Match(Src:10.0.1.12/24,Dst:10.0.2.12/24) Action (SET_NW_SRC:10.0.4.12/24 AND Forward)
2. S2:Match(Src:10.0.4.12/24,Dst:10.0.2.12/24) Action (SET_NW_DST:10.0.3.12/24 AND Forward)
3. S3:Match(Src:10.0.4.12/24,Dst:10.0.3.12/24) Action (Forward).

The packet sent by host A (Src: 10.0.1.12 Dst: 10.0.2.12) will be rewritten by the Set-Field rules existing in switch S1 and S2. Finally it will arrive at host C through

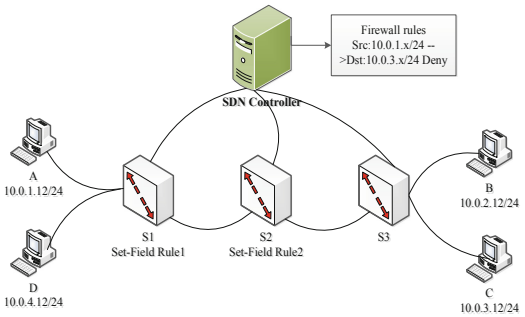

Fig. 1. Dependency collision that causes failure of firewall rules

switches S1, S2, and S3 due to the modification of the packet, which is a violation of the security rule that host A cannot communicate with host C. This type of collision is due to the dependency of the flow rules and it is extremely harmful.

3 Comprehensive Rule Collision Detection Mechanism

3.1 Solution Outline

As shown in Fig. 2, our comprehensive detection and solution mechanism for rule collisions is mainly implemented at the SDN control plane. When the supernatant applications issue the flow rules by the northbound interface, first they will be checked by the identity authentication and authorization mechanism of the controller. Then, the collision detection mechanism can intercept the rules to be issued and real-time detect whether there are static collisions in rules themselves and/or dynamic collisions with the existing security rules. Once there is a collision, it will proceed with the automatic collision solution. In addition, the global Set-Field type rules in the switches are offline compared with all existing security rules in the SDN to detect dependency collisions, which is performed using the improved alias-set algorithm.

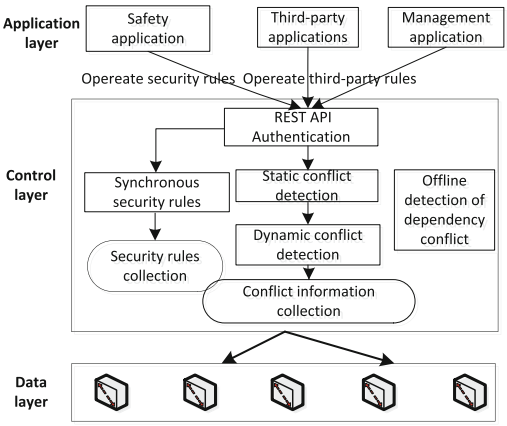

Fig. 2. The overall structure of the detection mechanism

3.2 Collision Detection Algorithm

First, the detection mechanism for static collisions proposed in this paper mainly detects whether there are some errors in the rule to be issued from the following aspects: the dl_src and dl_dst are the same, the nw_src and nw_dst are the same, and whether the value range of the header fields are legal. After the detection of the static collision, incorrect flow rules will be effectively filtered to prevent them from being sent to the OpenFlow switches.

The detection of dynamic collisions primarily focuses between the flow rules from third-party applications and those from applications with higher priority. To establish a feasible model of collision detection, we first need to determine the relationship between the two flow rules, including equal, inclusive, intersectant and irrelevant. According to the definition of the openflow flow rule in the previous chapter, the matching domain of the rule F_i can be represented as C_i, f_k^i represents the *k-th* header field in C_i, the F_j, C_j and f_k^j is alike. The relationship between flow rules F_i and F_j is defined as follows:

1. Equal: For the two match domain C_i and C_j in two flow rules, $\forall k : f_k^i = f_k^j (1 \le k \le n)$, the value of the header field are correspondingly equal with each other.
2. Inclusive: For C_i and C_j, $\forall k : f_k^i \subseteq f_k^j (1 \le k \le n)$, and $\exists k : f_k^i \ne f_k^j$, then the *i-th* flow rule becomes a child of the *j-th* flow rule. We call the rule F_i includes F_j or F_j is included in F_i, if the flow rule F_i has a higher priority than F_j, then the F_j will become a repetitive and unmeaning flow rule which won't work as long as the F_i exists.
3. Intersectant: For C_i and C_j, $\exists k : f_k^i \cap f_k^j \ne \varnothing$, and $\exists m, n : f_m^i \subsetneq f_m^j, f_n^j \subsetneq f_n^i$ the two rules will be matched only when the specific packet arrives.
4. Irrelevant: for C_i and C_j, $\exists k : f_k^i \cap f_k^j = \varnothing$, the two flow rules are irrelevant and do not interact.

The detection of a dynamic collision mainly aims at the situation when the relationship between two flow rules is equal, inclusive or intersectant. The detection algorithm will judge whether there is a dynamic collision according to the relationship between the two matching domains of the rules and the size of the priority. The collision detection and solution mainly include the following situations:

5. If C_1 and C_2 are equal or C_1 contains C_2, it represents the dynamic collision and reject to issue the rule F_2
6. If C_2 contains C_1, then compare priority P_1 and P_2. If $P_1 < P_2$, then adjust $P_2 = P_1 - 1$ so that the priority of the third-party flow rule is less than the security rule, then it will be issued normally
7. If C_2 intersects C_1, compare P_1 and P_2. If $P_1 < P_2$, set $P_2 = P_1 - 1$, then it will be issued normally
8. If C_2 and C_1 are irrelevant, then it will be issued normally.

In terms of the detection of dependency collisions, we only need to compare the existing firewall rules and all Set-Field rules that could rewrite the data packet. The firewall rules in the SDN are generally expressed as follows:

<dl_src><dl_dst><dl_type><nw_src><nw_dst><nw_type><tp_src><tp_dst><actions>.

The difference from the FortNox detection solution is that we will consider all header fields in the firewall rules to perform dependency collision detection in a fine-grained manner, thereby making the detection results more accurate and preventing false alarms. When using the set-intersection algorithm to detect dependency collisions, it is necessary to aggregate the flow rules into the format as the following:

$$SRC \rightarrow DST\ actions \tag{4}$$

$$SRC = \{SRCFi\}(i \in N, l \leq i \leq n) \tag{5}$$

$$DST = \{DSTFi\}(i \in N, 1 \leq i \leq n) \tag{6}$$

$$SRCF_i = (C_{dl\_src}, C_{dl\_type}, C_{nw\_src}, C_{nw\_type}, C_{to\_src}) \tag{7}$$

$$DSTF_i = \left(C_{dl\_dst}, C_{nw\_dst}, C_{tp\_dst}\right) \tag{8}$$

The SRC is called the source set, and the DST is called the destination set. Each element $SRCF_i$ in the source set includes the following head fields of OpenFlow: dl_src, dl_type, nw_src, nw_type and tp_src field. Each element $DSTF_i$ in the destination set DST includes the following head fields: dl_dst, nw_dst and tp_dst field. The actions in the firewall rules include "Allow", "Deny" and "Packetin".

When we use the set-intersection algorithm to detect the dependency collision among the flow rules, it first needs to covert the all firewall rules whose action are "Deny" to aggregate their representation in the format of $S_{src1} \rightarrow S_{dst1}$ *actions1*, and the all Set-Field flow rules are aggregated into $S_{src2} \rightarrow S_{dst2}$ *actions2*. The values before and after the modification operation in the Set-Field rules are respectively added to the source set S_{src2} and the destination set $S_{dst2.}$ Next a pairwise comparison is made between the firewall rules set and the Set-Field rules set to detect the dependency collision. We judge the to the principles as the following:

9. If $S_{src1} \cap S_{src2} = \varnothing$ *or* $S_{dst1} \cap S_{dst2} = \varnothing$, it represents that there is no dependency collision
10. If $S_{src1} \cap S_{src2} \neq \varnothing$ *and* $S_{dst1} \cap S_{dst2} \neq \varnothing$, actions1 = actions2, it represents that there is no dependency collision
11. If $S_{src1} \cap S_{src2} \neq \varnothing$ *and* $S_{dst1} \cap S_{dst2} \neq \varnothing$, actions1 $\neq$ actions2, it represents that there is a dependency collision, and then automatic collision resolution is required to delete the Set-Field rules that have dependency relationships.

The algorithm of detecting dependency collision based on set-intersection method is described as follows:

Algorithm1. The detection of dependency collision

input:
S_Set means an aggregate representation of firewall rules;
F_Set means an aggregate representation of Set-Field rules
output: the result of detecting dependency collision
FOR(i=1,2,3,….,m) DO
S_{src1}=S_Set[i].Src; S_{dst1}= S_Set[i].Dst;
actions1=S_Set[i].Actions
FOR (j=1,2,3,….,n) DO
  S_{src2}=F_Set[j].Src; S_{dst2}= F_Set[j].Dst;
  actions2=F_Set[j].Actions
  IF($S_{src1} \cap S_{src2} = \emptyset$ || $S_{dst1} \cap S_{dst2} = \emptyset$)
    continue;
  ELSE
    IF ($S_{src1} \cap S_{src2} \neq \emptyset$) srcResult=true;
    IF ($S_{dst1} \cap S_{dst2} \neq \emptyset$) dstResult=true;
    IF (srcResult && dstResult) break;
  IF (srcResult && dstResult)
    IF (actions1= actions2) return No
    ELSE return YES
  ELSE return No

4 Implementation and Evaluation

Based on an open source RYU controller, we implement the synchronization module of flow rules, the real-time detection module of rule collision and the offline detection module. We once again compile the source code with new functions of the RYU, and it becomes a security controller with a rule collision detection mechanism. This is done to evaluate the effects of the detection module and performance. We deploy and start the RYU controller in a server with the Ubuntu 15.04 LTS Operating System and 4 GB RAM. Moreover, in another server with the same configuration, we use Mininet to build the simulation environment of the SDN topology, which as shown in Fig. 3. It contains one RYU controller, three SDN switches (s1, s2, and s3) and six hosts (h1, h2, h3, h4, h5, and h6).

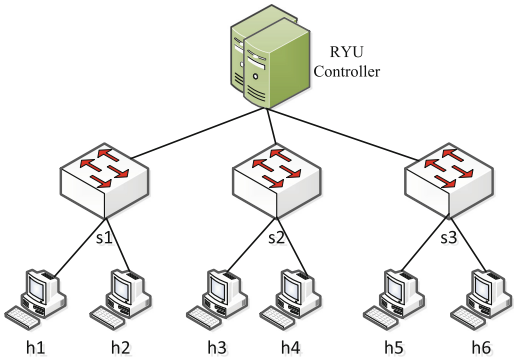

Fig. 3. The simulation environment of SDN network topology

4.1 Function Testing

To test the correctness of the detection mechanism, first, we issue the security flow rule {(*,*,*,'0x0800','10.0.0.1,'10.0.0.101','tcp',*,*) priority:1000 actions: Deny]} to s1 and {(*,*,*,'0x0800','192.168.1.22,'192.168.3.22','tcp',*,'8080') priority:2000 actions: Deny} to s2.

Table 1. The rules to be issued

Number	Rule
1	S1:(*,*,*,'0x0800','10.0.0.1,'10.0.0.100',*,*,*) priority:100 actions:Output:12
2	S1: (*,*,*,'0x0800','10.0.0.1,'10.0.0.101',tcp,*,*) priority:2000 actions: Output:12
3	S1: (*,*,*,'0x0800','10.0.0.1,'10.0.0.101',tcp,8,*) priority:2000 actions:Output:1
4	S1: (*,*,*,'0x0800','10.0.0.1,'10.0.0.101',*,*,*) priority:2000 actions:Output:4
5	S1: (*,*,*,'0x0800','10.0.0.1,'10.0.0.101',*,8,*) priority:1000 actions:Output:4
6	S2:(*,*,*,'0x0800','192.168.1.22,'192.168.2.22',*,*,*) priority:100 actions: OUTPUT:2; SET_NW_SRC:192.168.4.22
7	S2:(*,*,*,'0x0800','192.168.4.22,'192.168.2.22',*,*,*) priority:100 actions: OUTPUT:2; SET_NW_DST:192.168.3.22
8	S3:(*,*,*,'0x0800','192.168.8.22,'192.168.8.22',*,*,*) priority:100 actions: OUTPUT:2

After the security rules are issued successfully, we begin to issue some rules from the third-party application, which as shown in Table 1. The rules to be issued have static collisions themselves or dynamic and dependency collisions with the security rules in SDN. In our experiments, we need to record the results of the rule collision detection and the collision resolution, which verifies whether the functioning of our detection algorithm is correct. Table 2 shows the detection results and collision

solution, the collision detection algorithm can correctly and effectively detect the static collisions, dynamic collisions and dependency collisions in flow rules. Furthermore, rule collision situations can automatically be resolved by the detection system.

Table 2. The result of collision detection

Number	Check result	Collision resolution
1	No conflict	Normally be issued
2	Equal with s1-1,the actions is diff.	Refused to issue
3	Included by s1-1,the actions is diff.	Refused to issue
4	Included by s1-1,the actions is diff.	Adjust the priority = 999 and normally be issued
5	Intersects with s1-1, the actions is diff.	Refused to issue
6	No conflict	Normally be issued
7	Depends with the sixth rule, has a conflict with s2-1	Delete the dependency conflict flow rules
8	Static conflict	Refused to issue

4.2 Performance Testing

We start 10 threads in Jmeter and continuously request the northbound interface of RYU using HTTP protocol. Within 60 s, when the numbers of security flow rules in the SDN network are 50, 100, 200, 500, 1000, 2000 and 5000, we record the changes of the throughput capacity and the average response times of the northbound interface. The test results of the throughput are shown in Fig. 4. It can be observed that after adding the function of real-time detection to the RYU controller, the throughput is relatively stable, although it is reduced by approximately 20% compared to the original RYU. The average response time is shown in Fig. 5. After adding the collision detection mechanism, the average response time is extended. However, the collision detection mechanism is taken into account to ensure the overall security of the SDN. Under the premise of security, the decrease of the throughput and request response time is within the acceptable range.

In addition, the main factor leading to the decrease of the throughput of the northbound interface is the real-time collision detection

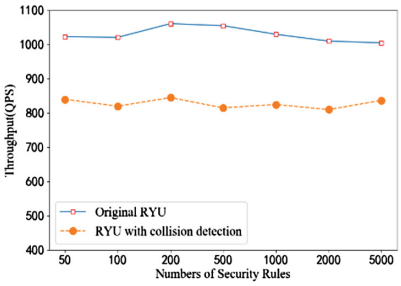

Fig. 4. The result of the throughput

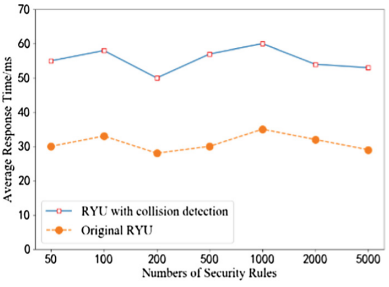

Fig. 5. The result of the average response time

before the flow rules are deployed to the OpenFlow switches. Under the conditions of different numbers of existing security rules in the SDN, we record the real-time detection times. The results are shown in Fig. 6. When the number of existing security rules is from 100 to 1200, the actual real-time detection time almost linearly increases. However, in an actual SDN network, the number of security rules in a single OpenFlow switch does not exceed 1000. Therefore, within the controllable range of flow table entries, the delay is within an acceptable range.

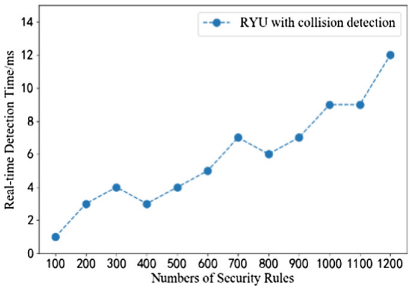

Fig. 6. The time of the real-time detection

5 Conclusion

For the security problems of rule collision in the application layer of the SDN network, we propose a comprehensive and fine-grained detection mechanism for rule collisions and implement it in the RYU controller. We evaluate the function and performance of the mechanism in a simulated SDN environment. The results show that it can correctly and efficiently detect various rule collisions and automatically resolve the collisions, which enhances the security of the SDN, and its performance is within the acceptable range. However, this paper does not consider the real-time network status when detecting dependency collisions. Therefore, there will be a delay in dependency collision resolution. In the future, our research will focus on this aspect and improve the detection mechanism of the flow rules.

Acknowledgments. This work was supported by the National Science Foundation of China (Grant No. 61502048) and the National Science and Technology Major Project (Grant No. 2017YFB0803001).

References

1. Michel, O., Keller, E.: SDN in wide-area networks: a survey. In: Fourth International Conference on Software Defined Systems, pp. 37–42. IEEE (2017)
2. Mckeown, N., Anderson, T., Balakrishnan, H., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
3. Rawat, D.B., Reddy, S.R.: Software defined networking architecture, security and energy efficiency: a survey. IEEE Commun. Surv. Tutorials **19**(1), 325–346 (2017)
4. Li, W., Meng, W., Kwok, L.F.: A survey on OpenFlow-based software defined networks: security challenges and countermeasures. J. Netw. Comput. Appl. **68**, 126–139 (2016)
5. Batista, B.L.A., Campos, G.A.L.D., Fernandez, M.P.: Flow-based conflict detection in OpenFlow networks using first-order logic. In: Computers & Communication. IEEE Computer Society, pp. 1–6 (2014)

6. Porras, P., Shin, S., Yegneswaran, V., et al.: A security enforcement kernel for OpenFlow networks, pp. 121–126 (2012)
7. Gude, N., Koponen, T., Pettit, J., et al.: NOX: towards an operating system for networks. ACM SIGCOMM Comput. Commun. Rev. **38**(3), 105–110 (2008)
8. Son, S., Shin, S., Yegneswaran, V., et al.: Model checking invariant security properties in OpenFlow. In: IEEE International Conference on Communications, pp. 1974–1979. IEEE (2013)
9. Wang, J., Wang, J., Jiao, H.Y., et al.: A method of Open-Flow-based real-time conflict detection and resolution for SDN access control policies. Chin. J. Comput. **38**(4), 872–883 (2015)
10. Wang, P., Huang, L., Xu, H., et al.: Rule anomalies detecting and resolving for software defined networks. In: Global Communications Conference, pp. 1–6. IEEE (2016)
11. Khurshid, A., Zhou, W., Caesar, M., et al.: VeriFlow: verifying network-wide invariants in real time. In: The Workshop on Hot Topics in Software Defined Networks, pp. 49–54. ACM (2012)