SDNSOC: Object Oriented SDN Framework

Ankur Chowdhary, Dijiang Huang and Gail-Joon Ahn
{achaud16,dijiang,gahn1}@asu.edu
Arizona State University

Myong Kang, Anya Kim and Alexander Velazquez
{myong.kang,anya.kim,alexander.velazquez}@nrl.navy.mil
U.S. Naval Research Lab

ABSTRACT

The cloud networks managed by SDN can have multi-tier policy and rule conflicts. The application plane can have conflicting user-defined policies, and the infrastructure layer can have OpenFlow rules conflicting with each other. There is no scalable, and, automated programming framework to detect and resolve multi-tier conflicts in SDN-based cloud networks. We present an object-oriented programming framework - SDN Security Operation Center (SDNSOC), which handles policy composition at application plane, flow rule conflict detection and resolution at the control plane. We follow the design principles of object-oriented paradigm such as code-re-utilization, methods abstraction, aggregation for the implementation of SDNSOC on a multi-tenant cloud network. The key benefits obtained using this approach are (i) The network administrator is abstracted from complex-implementation details of SFC. The end-to-end policy composition of different network functions is handled by an object-oriented framework in an automated fashion. We achieve 37% lower latency in SFC composition compared to nearest competitors - SICS and PGA. (ii) Policy conflict detection between the existing traffic rules and incoming traffic is handled by SDNSOC in a scalable manner. The solution scales well on a large cloud network., and 18% faster security policy conflict detection on a cloud network with 100k OpenFlow rules compared to similar works - Brew, and Flowguard.

CCS CONCEPTS

• **Security and privacy → Virtualization and security**; • **Networks → Security protocols**; *Network performance modeling*; *Cloud computing*; • **Software and its engineering → Object oriented architectures**.

KEYWORDS

Software Defined Networking (SDN); Service Function Chaining (SFC); Policy Conflict Detection

ACM Reference Format:
Ankur Chowdhary, Dijiang Huang and Gail-Joon Ahn, Myong Kang, Anya Kim and, and Alexander Velazquez. 2019. SDNSOC: Object Oriented SDN Framework. In *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFVSec '19), March 27, 2019, Richardson, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3309194.3309196

1 INTRODUCTION

The centralized command and control mechanism of SDN allows separation of control and data-plane functionality in a cloud network. The network traffic while servicing a request from a client to server often passes through various virtual network function (VNFs), such as Firewall, Intrusion Detection System (IDS), load balancer. This end-to-end delivery of traffic between two hosts, while passing through a set of ordered or partially ordered VNFs and ordering constraints that must be applied to the packets, is also referred to as service function chaining (SFC) [8].

Challenges in Policy Composition: There are several issues that limit the deployment of SFC. The security policies of individual VNFs are intertwined with the OpenFlow rules when the underlying cloud network is managed by SDN. Moreover, the network administrator needs to take care of topological dependencies between different VNFs, configuration complexities, consistent ordering of VNFs in the SFC as highlighted by Quinn *et al* [16]. The results from a survey of enterprise middlebox deployments in cloud indicate that 60-70% of all failures in middleboxes are because of misconfiguration issues [17]. There is no production grade framework currently, which presents the SFC composition as an abstracted interface to the user/administrator so that he/she is shielded from underlying details that limit the scalability and security in SFC. The network-wide policy enforcement using policy graphs has been considered by PGA [15], but there is a duplication of VNFs while the SFC is deployed using PGA to achieve the desired packet processing capability. We use efficient object-oriented data-structures to prevent data and code-duplication.

OpenFlow Rule Conflict Issues: The application plane of the SDN allows distributed authorship of a single policy domain. Additionally, the header space of the packet entering the network may match more than one flow rule in the OpenFlow table. A new flow rule can enable (or disable) the network traffic that is otherwise disabled (or enabled) by existing rules. FortNOX [14] utilizes role-based authorization to enforce conflict detection and mitigation. The framework, however, considers only pairwise rule conflicts, without identifying rule dependencies across flow tables. Veriflow [11], NetPlumber [9], Flowguard [7] use real-time network-wide invariant checking, rule-dependency analysis for identifying and resolving rule conflicts, but they lack a declarative and modular design which can interpret SFC requirements and implement them agnostic of semantics in which they are used or underlying hardware. Moreover, these works are limited to flow-rule conflicts and do not consider the security policies at the application level, generating the flow rules.

Need for a programmatic framework: Programming framework based on object-oriented hierarchy for policy composition, rule detection and mitigation can facilitate code-re-utilization, dependencies between security policies at different layers of the cloud network while satisfying user requirements. Existing frameworks such as FRESCO [18] and Frentic [5] consider only traffic monitoring, security application deployment, and query optimization, and lack a built-in mechanism to check flow rule conflicts.

The key contributions of this research work are as follows:

- SDNSOC checks dependencies between SFC requirements and creates a policy graph based on object-oriented design constructs. Policy graph is transformed into OpenFlow rules.
- Utilizes object-oriented design principles such as inheritance, aggregation, composition in order to identify dependencies between VNFs, security policies, and conflicts between the security policies. The framework is able to identify dependencies between security policies, and OpenFlow rules thus handle policy conflict at multiple layers of the cloud network.
- Identifies the flow rule conflicts at the infrastructure layer that may arise because of SFC deployment in order to eliminate flow rule conflicts in SDN-based cloud network.

2 BACKGROUND AND MOTIVATION

Definition 2.1. **OpenFlow Rule:** A flow table F of an OpenFlow switch, can have rules, $\{r_1, r_2, .., r_n\}$ Each rule consists of layer 2-4 packet header fields, protocol (TCP/UDP/FTP), action-set associated with the rule, rule priority, and statistics. We define the flow rule using tuple $r_i = (p_i, \rho_i, h_i, a_i, s_i)$, where a) p_i denotes rule priority, b) ρ_i denotes the protocol of the incoming traffic (TCP/UDP) c) h_i depicts the packet header, d) a_i is the action associated with the rule, e) s_i represents the statistics associated with the rule.

The flow rule header space h_i, consists of physical port of incoming traffic δ_i, source and destination hardware address, i.e., $\alpha_{s\,i}, \alpha_{d\,i}$, source and destination IP address, $\beta_{s\,i}, \beta_{d\,i}$, source and destination port address, $\gamma_{s\,i}, \gamma_{d\,i}$. Packet header can be defined by the tuple $h_i = (\delta_i, \alpha_{s\,i}, \alpha_{d\,i}, \beta_{s\,i}, \beta_{d\,i}, \gamma_{s\,i}, \gamma_{d\,i})$. Rule statistics s_i, comprises of both flow duration and number of packets/bytes for each flow rule $s_i = (d_i, b_i)$.

2.1 Motivating Scenario: Security Policy Composition Issue

Definition 2.2. Security Policy: A security policy is a packet processing requirement specified by the network administrator at the application plane in form of service function chaining requirement, which, is composed into a low-level flow or firewall rules on the network switches.

```
 service-chain-01 {
   classifier{ group: employee
               port:443,80, dst: server}
               50 firewall
               40 load-balancer
               30 server}
  service-chain-02 {
   classifier{ group: remote-user port:443,80
               dst: web-server}
               50 vpn
               40 web-server}
```

Consider the security policies above, there are two service function chains, i.e., *service-chain-01* and *service-chain-02.* The first service chain requirements want all users in the group *employee* to pass through the firewall, load balancer while trying to access the server VM on ports 80,443. On the other hand, the second service chain allows employees who are in External-User group to access webserver through port 80 using VPN. The numeric values [50.40,30] in front of individual VNFs indicate their order of precedence in the VNFs in the SFC.

Definition 2.3. **Security Policy Composition:** The end-to-end service chain creation which satisfies, all the access requirements for different security groups while maintaining the satisfaction of security constraints. An example of a security constraint is that the firewall always operates on the raw un-encrypted traffic.

The two policies have been composed by different security administrators, and leads to **security constraint violation**, since the *External-User* inherits *Employees* and *Web-Server* inherits *Server*, and the traffic passing through the VPN box is tunneled directly to the web-server without being inspected by the Firewall.

We can use the VNF object-oriented (OO) chain creation to compose the security policies from individual SFC requirements. The algorithm will identify the relationship between different SFC groups and create an end-to-end chain satisfying security constraints.

2.2 Motivating Scenario: Flow Rule Conflict Analysis

Definition 2.4. **Packet Classification:** The incoming traffic packet for a network, Π_i, can be classified into subset of rules R_m from the ruleset of the entire network R, i.e., $R_m \subseteq R$, where $R_m = \{\forall_{i=1}^{m} r_i\}$.

Definition 2.5. **Conflict Detection** problem [3] seeks to find the rules $r_i, r_j \in R_m$, which have are conflicting with each other, i.e., $(\rho_i = \rho_j) \wedge (h_i \cap h_j \neq \emptyset) \wedge (a_i \neq a_j) \wedge (p_i \neq p_j)$. The variables used here, have been defined in Definition 2.1 earlier.

Table 1: Motivating Scenario - Conflict Detection

Flow-ID	Src-IP	Dst-IP	Src-Port	Dst-Port	**Action**
1	1	[0-100]	2	[0-100]	drop
2	1	[0-100]	[2,4]	[0-100]	srcip=5
3	2	[0-100]	[2,4]	[0-100]	fwd
4	[5,8]	[0-100]	[0,8]	[0-100]	srcip=2

Consider, Table 1, we use simple numeric values for source and destination addresses for concise representation and consider other OpenFlow fields, e.g., layer 2 source and destination addresses to be wildcarded. There is two type of violations here.

Coverage Violation: The rules 1 and 2 have overlapping header space, the Src-IP of the rules is same, the destination IP of rule 2 is a superset of rule 1, whereas the actions for both rules are different.

Transitive Violations: The According to rule with Flow-ID '1' present in the table, every packet from Src-IP 1 towards Dst-IP 2 must be dropped, however, rule 2 in the table allows modification of source IP to value 5, and the rule 4 sets the source IP of any field between [5-8] to the value 2. Thus, using rules 1,3, and 4 the traffic between Src-IP 1 and Dst-IP 2 is allowed.

Research works, Flowguard [7], and Brew [13], focus on only on Firewall as a use-case for security policy conflict detection, and fail to identify *Transitive Violations*. In this paper, we use object-oriented fundamentals, and, classify the flow rule dependencies into *Inheritance, Polymorphism, Aggregation, and Composition* to identify both direct and indirect dependencies between OpenFlow rules.

3 SDNSOC ARCHITECTURE

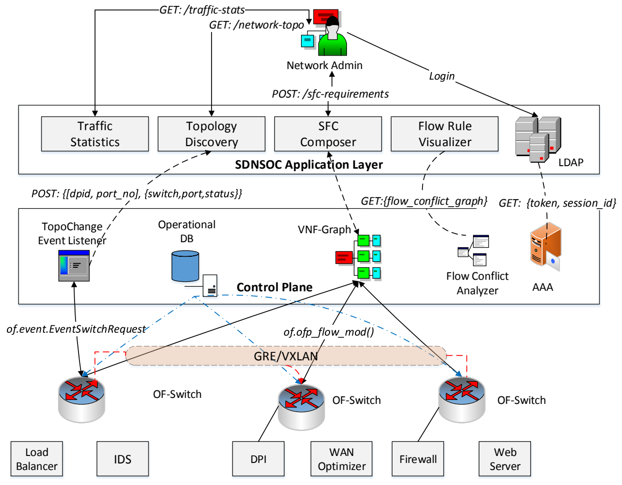


Figure 1: SDNSOC Architecture

The SDNSOC, as shown in the Figure 1, is divided into three layers, i.e., application plane layer - which consists of UI where user can login, and perform network analytics such as traffic statistic detection, topology visualization, OpenFlow table rule visualization and SFC requirement specification, control plane layer - we use OpenDaylight (ODL) controller in the control plane. The UI has been implemented in a PHP lavarel based framework.

Topology Discovery: The controller consists of topology change event listener, which listens on the events such as port status (UP-/DOWN), switch status, port information of hosts connected to switches.

SFC Composition and VNF-Graph: The application plane consists of SFC composition template, along with the order of precedence for processing of different VNFs. Since multiple-users can specify service chains at a given point-in-time. The VNF-Graph creator module in the control plane compiles the requirements of SFC specified by the user.

Conflict Analyzer: The conflict analysis module utilizes REST API to fetch the Flow rules from OpenFlow switches using REST API. The rules are analyzed for potential overlap in the header match and action fields, which can lead to the violation of end-to-end security policies or service delivery. The conflicting flow rules can be visualized at the dashboard UI for in-depth analysis by the network administrator. The VNFs are connected to the OpenFlow switches. The switches, which belong to different network segments are connected using GRE/VXLAN.

4 OBJECT ORIENTED SFC FRAMEWORK

We design an object-oriented multi-tiered network security architecture to provision distributed security in a cloud data-center. A VNF, or networking domain can be considered a class. A class can be instantiated to create an object which represents a specific implementation of the class.

Inheritance enables new classes to inherit the properties and methods of the existing classes. A class which inherits from the superclass is called a subclass or derived class. For instance, consider a class *Subnetwork* which provides the basic layout of the network bits and host bits, e.g., for a class B Subnet Prefix is 16 and Host Prefix is 16, thus there can be 16 host bits or 2^{16} hosts that can be present in this subnetwork. The class *Ethernet Address* can inherit the network mask functionality (Network and Host Prefix) from the class Subnetwork. In addition, subclass also adds features such as IPv4 address (e.g., 192.168.1.12), Gateway (e.g., 192.168.1.1), Nameserver (e.g., 8.8.8.8) and broadcast address (e.g., 192.168.1.255). An object for this class is ethernet address for a specific host.

Polymorphism is one of the features in OOP that allows a single action to be performed in different ways. Using the polymorphism in virtual network functions allows the creation of one interface for instance Firewall , that can be realized in a different way depending upon the application requirement. The smart firewall architecture like Cisco-ASA [6] implements several security features such as Intrusion Detection, Anti-malware capabilities and VPN service in one single device. A polymorphic design of VNF can help in an extension of current VNFs to new firewall architecture. A stateless firewall, which provides basic functions of traffic filtering and NAT, can be extended to a stateful firewall, providing connection tracking for stateful applications, intrusion prevention system (IPS), etc.

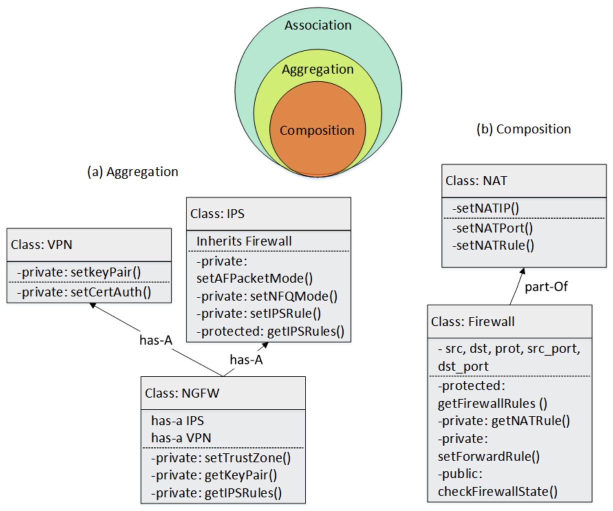


Figure 2: Association in object-oriented architecture

Aggregation is a weak form of association that enables one VNF to utilize another, without having to re-implement the functional logic present in original VNF. For instance, Next Generation Firewall (NGFW) as shown in the Figure 2(a), typically comes with features such as VPN and DPI. The NGFW and VPN can, however, function as a standalone VNFs even if either is missing in security architecture. These weak associations allow re-utilization of desired features amongst VNFs using a *has-A* relationship.

Composition on the other hand, is a stronger form of association, usually represented by *part-Of* relationship. The functionality of Network Address Translation (NAT) cannot exist by itself, and it requires the presence of Firewall VNF as shown in the Figure 2(b). The firewall module can call setNATIP() and setNATPort() functions in the class NAT in order to allow NAT feature mapping an external IP address/port to an internal IP address/port in addition to other features such as port forwarding and blocking a certain type of network traffic.

5 SDNSOC SFC COMPOSITION, CONFLICT DETECTION AND RESOLUTION

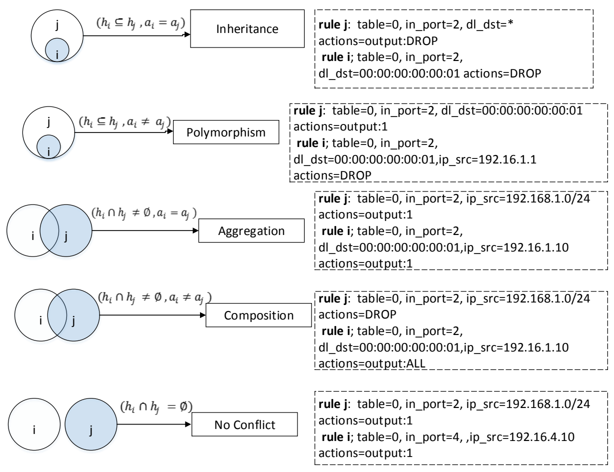
5.1 Flow Composition

Algorithm 1 SFC-Composition

```
 1: procedure VNF-GRAPH(SFC, C)
 2:     $SFC \longleftarrow$ List of Service Chains
 3:     $C \longleftarrow$ List of classification constraints
 4:     $G \longleftarrow \emptyset$ VNF Graph
 5:     for $c_i \in$ C do
 6:         for $sfc_j \in$ SFC do
 7:             if $c_{ij}.match \cap sfc_j.range \neq \emptyset$ then
 8:                 G.addnode($c_{ij}.group$)
 9:             end if
10:         end for
11:     end for
12:     for $i \in G.nodes$ do
13:         for $j \in G.nodes$ do
14:             if $i.range \in j.range || j.range \in i.range$ then
15:                 G.addedge(i,j,inheritance)
16:             else if $i.range \cap j.range \neq \emptyset$ then
17:                 G.addedge(i,j,aggregation)
18:             end if
19:         end for
20:     end for
21:     Return G
22: end procedure
```

The algorithm 1 finds the dependencies between various service function chains and the classification criteria defined as the part of classification constraints. For any incoming traffic, lines 5-7 checks the classification constraints against each service function chain (SFC). If there is a match on the incoming traffic header, a node is added to the VNF-Graph, as shown in the line 8. In the second part of the algorithm, lines 12-20, all the nodes of the graph are matched against each other to check the range overlap, if one node's IP-range is a subset of other, we define inheritance relation between graph nodes. On the other hand, if there is a partial overlap, we define an aggregation relationship. Thus, the edges between the graph nodes are added in this part. The VNF-Graph is converted parsed and converted to flow-rules corresponding to each node of the graph. The flow rules are checked for conflicts based on header space and actions overlap, which we discuss in the next subsection.

5.2 Flow Rule Conflict Detection


j
i
$(h_i \subseteq h_j, a_i = a_j)$
Inheritance
$(h_i \subseteq h_j, a_i \neq a_j)$
Polymorphism
$(h_i \cap h_j \neq \emptyset, a_i = a_j)$
Aggregation
$(h_i \cap h_j \neq \emptyset, a_i \neq a_j)$
Composition
$(h_i \cap h_j = \emptyset)$
No Conflict

```
rule j:  table=0, in_port=2, dl_dst=*
actions=output:DROP
 rule i; table=0, in_port=2,
dl_dst=00:00:00:00:00:01 actions=DROP
```

```
rule j:  table=0, in_port=2, dl_dst=00:00:00:00:00:01
actions=output:1
 rule i; table=0, in_port=2,
dl_dst=00:00:00:00:00:01,ip_src=192.16.1.1
actions=DROP
```

```
rule j:  table=0, in_port=2, ip_src=192.168.1.0/24
actions=output:1
 rule i; table=0, in_port=2,
dl_dst=00:00:00:00:00:01,ip_src=192.16.1.10
actions=output:1
```

```
rule j:  table=0, in_port=2, ip_src=192.168.1.0/24
actions=DROP
 rule i; table=0, in_port=2,
dl_dst=00:00:00:00:00:01,ip_src=192.16.1.10
actions=output:ALL
```

```
rule j:  table=0, in_port=2, ip_src=192.168.1.0/24
actions=output:1
 rule i; table=0, in_port=4, ,ip_src=192.16.4.10
actions=output:1
```

Figure 3: OpenFlow Rule Conflict Analysis

We utilize the class hierarchy that we described for different VNFs in the previous section to illustrate the process of flow rule conflict identification. We consider the overlap in the action fields. As shown in the Figure 3, we can have four different cases of an object-oriented framework, which can cover different scenarios of flow rule conflicts.

- **Inheritance:** As shown in the example above, the header fields $h_i \subseteq h_j$, and actions of both rules are same, thus rule *i* is a specialization of rule *j*.
- **Polymorphism:** The example showcases two rules, where rule *i* inherits the header values of rule *j*, however, the action fields are different. This is similar to polymorphism property in the object-oriented design, thus we classify such cases of rules are polymorphic conflicts.
- **Aggregation:** Rules *(i, j)* in the example of aggregation have overlapping header fields, i.e., $h_i \cap h_j \neq \emptyset$, however, both rules, have same action, thus a third rule, *k* can replace both rules, but this doesn't happen automatically in flow tables. We classify such conflict scenarios as aggregation.
- **Composition:** Rules *(i, j)* in this conflict scenario have overlapping header fields, and conflicting actions, thus the intersecting part of both rules, $h_i \cap h_j$, is composed of aggregated actions from both rules. This type of rule conflicts can be classified into composition category.

The cases where there is no overlap in the header fields, there is no case of rule conflict, irrespective of the actions, as shown in the Figure 3.

6 IMPLEMENTATION AND EVALUATION

6.1 System Setup

We utilized an OpenStack based cloud network comprising of two Dell R620 servers and two Dell R710 servers all hosted in the data center - Science DMZ [1]. Each Dell server has about 128 GB of RAM and 16 core CPU. The SDN controller Opendaylight-Carbon was provided network management and orchestration in our framework. The VNFs Web Server, OpenVPN, Firewall (netfilter), and load balancer (nginx) were used for evaluation of SFC composition and conflict analysis. The flow rules were installed on OpenFlow switches managed by SDN controller.

6.2 SFC Flow Composition Analysis

6.3 Composition Time Comparative Analysis

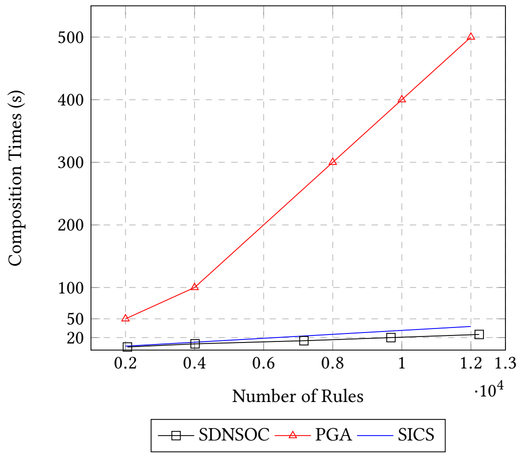


Figure 4: Number of Rules vs Composition Time - SDNSOC, PGA [15], SICS [19]

We performed a comparative analysis of composition time for SDNSOC against policy composition time of PGA [15] and SICS framework [19]. We use rules as a generic term to define PGA nodes, SICS rules, and OpenFlow rules, and to have a common comparison format. We observed that SDNSOC achieves faster composition time - 20s for 10k rules, and 25s for about 12k rules. The composition time for SICS was slightly higher than our framework, i.e., 31.5s for 10k rules and 37.5s for 12k rules. The composition time for PGA scales poorly with the number of rules as can be seen in the Figure 4. PGA takes about 400s for the composition of 10k rules and 500s for 12k rules. The performance degradation in SICS can be attributed to encryption overhead, whereas in the case of PGA, the poor scaling is because of duplication of SFs across the network. The comparison of SDNSOC with these frameworks shows that SDNSOC will scale well with the number of policy rules.

6.4 Flow Rule Conflict Analysis

We performed experiment to analyze the number of conflicts - *Inheritance (IN)*, *Polymorphism (P)*, *Aggregation (A)* and *Composition (C)* in the translated OpenFlow rules. The x-axis in the Figure 5 denotes the number of OpenFlow rules - 5k, 16k, and 25k. As the number of OpenFlow rules increased, we observed an increase in the number of conflicts. For the dataset with 5k OpenFlow rules, we identified 484 conflicts because of inheritance dependency, 936 polymorphism related conflicts, 9 aggregation conflicts, and 24 composition conflicts. Our conflict checking algorithm identified 1041 inheritance conflicts, 1989 polymorphism, conflicts, 49 aggregation conflicts and 336 composition conflicts in 25k OpenFlow rules.

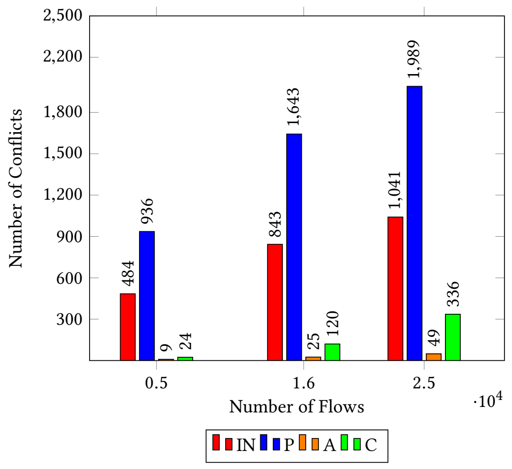


Figure 5: Number of Conflicts in OpenFlow Rules

The experiment demonstrates that managing conflicts for even few thousand rules manually can be quite challenging. Hence we use an automated detection and resolution framework for flow rule conflicts.

We performed a comparison of the flow rule conflict detection algorithm with OpenFlow conflict checking research works Brew [13] and Flowguard [7]. Object-oriented conflict detection is able to detect transitive conflicts using multi-level inheritance, which have not been considered by both works. Additionally, our framework is generalizable to many different VNFs, whereas Brew and Flowguard only considered policy conflict issues in a firewall. Our experimental results show that there can be large a number of conflicts in flow rules that can be identified only by automated composition and conflict analysis using an object-oriented framework.

6.5 Flow Rule Conflict Analysis Scalability

In this experiment, we utilized the Stanford University backbone network topology [9] for analyzing the scalability of conflict detection algorithm. The network consists of multiple layers of switches and routers, about 13k routes and 757k forwarding rules, 100 VLANs, and 900 ACL rules. The network was simulated using mininet, routers and switches were replaced with OVS, and the flow rules from the actual network were inserted using a python script.

We compared the running time for detecting conflicts of the object-oriented policy conflict detection method, with existing policy conflict detection works, Brew and Flowguard, which utilize Stanford topology for experimental analysis. The performance of SDNSOC is slightly slower than Brew for 10k rules ~9ms, but as the size of the flow rules increases, the SDNSOC performs better than both Brew and Flowguard. Flowguard only considered conflict detection for about 40k rules, the performance of our conflict checking procedure (19ms) is significantly better than Brew (22ms) and Flowguard (39ms) for 40k rules. The results from the Figure 6, shows that the running time for 50k flow rules is 25ms, and about 45ms for 100k flow rules, which is clearly 18% performance gain over Brew (55ms). Hence the flow rule conflict detection algorithm based on object-oriented principles scales well on the large network. The performance gain of SDNSOC can be attributed to the fact that

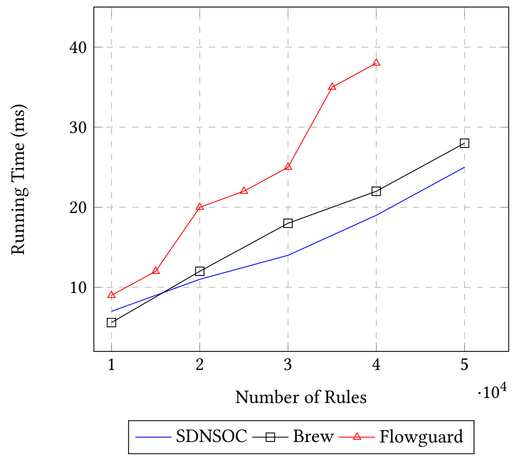


Figure 6: Number of Flow Rules vs Policy Conflict Detection Time - SDNSOC, Brew [13], Flowguard [7]

once VNF-Graph is constructed, the search for conflicts in hierarchical structure when a new rule is added, is a trivial operation, compared to the matching of new rule against every other rule in case of Brew, and Flowguard.

7 RELATED WORK

Policy Aware automatic composition of multiple independent network policies and Access Control Lists (ACLs) using graph-based expression has been discussed by Prakash *et al* [15]. FlowTags [4] extends *SDN* architecture for adding tags to outgoing packets. This provides a necessary context for policy enforcement. These works do not, however, consider possible overlaps between network policies based on packet header match. Works that focus on policy safety and efficiency of SFC like SDN based virtual firewall discussed by Deng *et al* [2] consider issues like semantic consistency, buffer overflow avoidance, and scalability but their application is limited to the firewall VNF. In our work, we use an object-oriented design that achieves optimal SFC composition and policy conflict resolution to allow different VNFs to provide a seamless service function chain (SFC).

Flow Rule Conflict analysis based on SDN flow rules has been considered by Pisharody *et al* [12, 13]. The research work only considers layer 2-4 flow rules and focus on traditional north-south traffic in a data-center. Our work considers more detailed policy conflicts at application as well as OpenFlow switch level. Veriflow [11] and NetPlumber [10] lack automatic real-time security policy resolution mechanism. The research work, however, does not consider indirect security policy violation detection. The object-oriented framework proposed in our work identifies the indirect violations by a notion of inheritance and generalizations between the dependent VNFs.

8 CONCLUSION

We discuss the problems associated with policy composition and flow rule conflict in this paper because of different administrative domains, and multi-tenancy in the SDN-managed cloud network. We utilized object-oriented principles to identify the policy dependencies at application tier and flow rule conflict issues at the infrastructure layer. The SDNSOC's architecture is able to provide scalable and faster policy composition, flow-rule conflict detection compared to existing works. As an extension of this research work, we plan to check application and effectiveness of SDNSOC on other platforms, which are not SDN-based such as Amazon cloud, and Google cloud platform.

ACKNOWLEDGMENT

All authors are gratefully thankful for research grants from Naval Research Lab N00173-15-G017 and National Science Foundation US DGE-1723440, OAC-1642031, SaTC-1528099.

REFERENCES

[1] Chowdhary, A., Dixit, V. H., Tiwari, N., Kyung, S., Huang, D., and Ahn, G.-J. Science dmz: Sdn based secured cloud testbed. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017 IEEE Conference on* (2017), IEEE, pp. 1–2.
[2] Deng, J., Li, H., Hu, H., Wang, K.-C., Ahn, G.-J., Zhao, Z., and Han, W. On the safety and efficiency of virtual firewall elasticity control. In *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS 2017)* (2017).
[3] Eppstein, D., and Muthukrishnan, S. Internet packet filter management and rectangle geometry. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms* (2001), Society for Industrial and Applied Mathematics, pp. 827–835.
[4] Fayazbakhsh, S. K., Sekar, V., Yu, M., and Mogul, J. C. 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* (2013), ACM, pp. 19–24.
[5] Foster, N., Harrison, R., Freedman, M. J., Monsanto, C., Rexford, J., Story, A., and Walker, D. Frenetic: A network programming language. *ACM Sigplan Notices 46*, 9 (2011), 279–291.
[6] Frahim, J., and Santos, O. *Cisco ASA: All-in-One Firewall, IPS, Anti-X, and VPN Adaptive Security Appliance.* Pearson Education, 2009.
[7] Hu, H., Han, W., Ahn, G.-J., and Zhao, Z. Flowguard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking* (2014), ACM, pp. 97–102.
[8] Huang, D., Chowdhary, A., and Pisharody, S. *Software-Defined Networking and Security: From Theory to Practice.* CRC Press, 2018.
[9] Kazemian, P. Header space analysis: Static checking for networks.
[10] Kazemian, P., Zeng, H., Varghese, G., McKeown, N., and Whyte, S. Real time network policy checking using header space analysis.
[11] Khurshid, A., Zhou, W., Caesar, M., and Godfrey, P. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks* (2012), ACM, pp. 49–54.
[12] Pisharody, S., Chowdhary, A., and Huang, D. Security policy checking in distributed sdn based clouds. In *Communications and Network Security (CNS), 2016 IEEE Conference on* (2016), IEEE, pp. 19–27.
[13] Pisharody, S., Natarajan, J., Chowdhary, A., Alshalan, A., and Huang, D. Brew: A security policy analysis framework for distributed sdn-based cloud environments. *IEEE Transactions on Dependable and Secure Computing* (2017).
[14] Porras, P., Yegneswaran, V., Shin, S., and Gu, G. A security enforcement kernel for openflow networks hotsdn 2012.
[15] Prakash, C., Lee, J., Turner, Y., Kang, J.-M., Akella, A., Banerjee, S., Clark, C., Ma, Y., Sharma, P., and Zhang, Y. Pga: Using graphs to express and automatically reconcile network policies. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 29–42.
[16] Quinn, P., and Nadeau, T. Problem statement for service function chaining. Tech. rep., 2015.
[17] Sherry, J., Ratnasamy, S., and At, J. S. A survey of enterprise middlebox deployments.
[18] Shin, S. W., Porras, P., Yegneswara, V., Fong, M., Gu, G., and Tyson, M. Fresco: Modular composable security services for software-defined networks. In *20th Annual Network & Distributed System Security Symposium* (2013), NDSS.
[19] Wang, H., Li, X., Zhao, Y., Yu, Y., Yang, H., and Qian, C. Sics: Secure in-cloud service function chaining. *arXiv preprint arXiv:1606.07079* (2016).