Mitigating Threats Emerging from the Interaction between SDN Apps and SDN (Configuration) Datastore

Sana Habib
Arizona State University
shabib3@asu.edu

Tiffany Bao
Arizona State University
tbao@asu.edu

Yan Shoshitaishvili
Arizona State University
yans@asu.edu

Adam Doupé
Arizona State University
doupe@asu.edu

ABSTRACT

Software-defined networking (SDN) has established itself in networking and standardization efforts are under way to strengthen the next generation of this essential technology. The Network Management Datastore Architecture (NMDA), RFC 8342, is the notable achievement in this regard, which standardizes the two vital SDN datastores: configuration and operational. Even though the configuration datastore itself has been standardized, the guidelines for addressing its security as well as safeguarding interactions between SDN apps and SDN configuration datastore are hazy, which leaves room for security vulnerabilities. Both industry and academia have realized the threats that arise due to the interactions between SDN apps and the SDN configuration datastore. But, to date only partial solutions exist for the problem.

In this paper, we focus on mitigating such threats by proposing four security design principles that we believe should be uniformly used across all SDN platforms: (i) authentication (of SDN apps), (ii) authorization (of SDN apps), (iii) accountability (of SDN apps), (iv) real-time conflict detection and resolution of configuration rules (belonging to the same/different SDN app/s). Based on these four security design principles, we develop and present a prototype implementation of the Eirene framework, an open-source vendor independent system for ensuring secure interactions between SDN apps-SDN configuration datastore. We then evaluate the security of the Eirene framework using two datasets: (i) real-world complicated cases of rule conflicts, (ii) 50,000+ real-world configuration (attack) rules. We believe that this work, by quantifying the security design principles for the next generation of SDN apps-SDN configuration datastore interactions, will assist in reaping the true benefits of the SDN powerhouse and ultimately inspire movements in the future generation of this critical technology.

CCS CONCEPTS

• **Security and privacy** → *Security services*; *Access control*; *Information accountability and usage control*; *Authorization*; *Authentication.*

CCSW '22, November 7, 2022, Los Angeles, CA, USA.

ACM ISBN 978-1-4503-XXXX-X/22/11…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

KEYWORDS

Software Defined Networking (SDN); SDN Configuration Datastore; SDN Apps; Configuration Rules.

ACM Reference Format:
Sana Habib, Tiffany Bao, Yan Shoshitaishvili, and Adam Doupé. 2022. Mitigating Threats Emerging from the Interaction between SDN Apps and SDN (Configuration) Datastore. In *Proceedings of the 2022 Cloud Computing Security Workshop (CCSW'22), November 7, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION: FUTURE OF NETWORKS

The SDN technology is considered one of the most disruptive technologies to networking that we have seen in decades. By making networks programmable and automated, SDN has redefined the future of networking. This is one of the reasons that the global SDN market value continues to grow, from around USD 13.7 billion in 2020 to approximately USD 32.7 billion by 2025 and to almost USD 117.27 billion by 2030 [4, 5]. SDN, almost 13 years old now, is mature with a standardized architecture (RFC 7426 [29]) that provides guidelines to vendors and enterprises for developing their own SDN solutions [5] (e.g., Hewlett Packard Enterprise Development [9], Huawei Technologies [11], Nokia Corporation [12]).

The SDN datastores are a vital entity in the overall SDN architecture, which store information about the desired and current state of the network. The Network Management Datastore Architecture (NMDA), standardized as RFC 8342 [50] in 2018, provides guidelines to enterprises who develop their own SDN controllers such as OpenDayLight (ODL) [15], Open Network Operating System (ONOS) [13], Ericsson Cloud SDN [7], FloodLight [8], etc.

The NMDA design specifies two key datastores [50]: (i) Configuration datastore, which contains the desired state of the network. (ii) Operational datastore, which contains the actual state of the network. The desired state of the network is defined as policies, converted into configuration rules, and then entered in the SDN configuration datastore by SDN apps or by the network administrator via the northbound API. Proper management of information inside the configuration datastore is crucial as it is translated into the network. The operational datastore contains statistics about the network. In this paper, we focus on mitigating the threats that arise due to the interaction between SDN apps and the SDN configuration datastore (a subset of these threats are summarized in Table 1). The root cause of these threats lies in the obscurity of universal design principles to be used across all SDN platforms for safeguarding

interactions between SDN apps and the SDN configuration datastore. For example, missing or poorly implemented authentication scheme for SDN apps can give a malicious SDN app access to the configuration datastore. The malicious SDN app can then hinder the smooth operation of the network by fraudulently inserting rules that tamper with the security policies of the network, exhaust datastore resources, etc. Similarly, missing or poorly implemented authorization can cause an SDN app to escalate its privileges and use more than its rightful or authorized share of datastore resources. Absence of or improperly implemented accountability causes a similar problem (i.e., one SDN app using more than its rightful share and depriving other legitimate SDN apps). Finally, absence of real-time conflict detection and resolution of configuration rules belonging to same/different SDN app/s can lead to misconfigurations in the datastore and consequently in the network.

Both industry and academia have partially realized this problem and proposed temporary solutions and services (such as SE-FloodLight [46], ODL AAA [14], ONOS AAA [6], SM-ONOS [57]). However, to date the security design principles for safeguarding interactions between SDN apps and SDN configuration datastore that could be uniformly used as a guideline across all SDN platforms are yet to be quantified. One of the reasons behind the understudied problem of securing the interaction between SDN apps and the SDN configuration datastore is the fast paced nature of industry, where new version of SDN controllers (such as ODL [15]) are rolled out quarterly and more time and energy is spent on the correct operation of the controller than its security. Even when the security issues are raised, the developers in industry use temporary fixes to deal with the problem and the underlying issues remain unaddressed. Realizing this need, this paper makes the following contributions:

(1) We quantify four security design principles to be uniformly used across multiple SDN platforms for mitigating threats that arise from the interactions between SDN apps and the SDN configuration datastore: (i) authentication of SDN apps, (ii) authorization of SDN apps, (iii) accountability of SDN apps, (iv) real-time conflict handling (detection and resolution) of configuration rules among different/same SDN app/s.
(2) Based on this quantification, we present a prototype implementation of the Eirene framework, which is an open-source vendor independent Authentication, Authorization, Accountability, and Conflict Handling (AAAC) service that can be used across multiple SDN platforms (with varying implementation details).
(3) We test the security of the Eirene framework on a small and a large scale using two real-world datasets (i) dataset containing real-world complicated cases of rule conflicts, (ii) 50,000+ real-world configuration (attack) rules, and demonstrate that the desired security goals are achieved with an acceptable performance overhead.

The accomplishment of this work required overcoming two key challenges. The *first* challenge was extracting security design principles that should be uniformly used across all SDN platforms. We did that by first defining our threat model, then categorizing the threats that emerge from the interaction between SDN apps and SDN configuration datastore. Following that, we synthesized and examined existing security services/solutions offered by industry based SDN controllers/academia against the extent to which they address and mitigate the problem. Once the security design principles were quantified, we designed and developed a prototype implementation of the Eirene framework on a widely used industry based open-source SDN controller, OpenDayLight (ODL) [15]. The *second* challenge was obtaining real-world datasets for testing the Eirene framework. This problem was solved using a real-world campus network dataset for small scale and 50,000+ configuration (attack) rules from [30] for large scale security evaluation of the Eirene framework.

Table 1: Motivating Threats.

	Attack Title	Year	Security Issue/CVE
1.	Unauthenticated Upload [1]	2017	CVE-2017-1000081
2.	Cross-App Poisoning [53]	2018	Privilege Escalation
3.	DoS [30]	2018	CVE-2017-1000411
4.	Mis-configurations [2]	2018	Rule Database Inconsistency
5.	Rule Conflicts [2]	2018	Rule Database Inconsistency
6.	Fraudulent Rule Insertion [2]	2018	Rule Database Tampering

2 BACKGROUND

This section provides the necessary background on SDN architecture. Fig. 1 graphically shows the SDN architecture.

2.1 SDN Architecture: The Nits and Grits

SDN has a centralized architecture that decouples the control plane (i.e., forwarding decisions) from the data plane (i.e., forwarding traffic). The centralized controller exposes an application programming interface (APIs) to (first- or third-party) apps, which is referred to as the northbound API. The northbound SDN apps define network policies in the form of forwarding rules that are stored inside the controller (configuration) datastore. These policies are then enforced by the controller in the data plane. The four main components of SDN architecture: (i) SDN controller, (ii) northbound and southbound APIs, (iii) core services, (iv) datastores, are discussed next.

2.1.1 SDN Controller. The controller can be thought of as a "network operating system" that centralizes control in a logically centralized control plane, which is responsible for servicing northbound SDN apps, provisioning resources to SDN apps, and enforcing policies (for security, traffic engineering, etc.) [35]. OpenDayLight (ODL) [15] and Open Network Operating System (ONOS) [13] are the two most widely used open-source SDN controllers.

2.1.2 Northbound and Southbound API. SDN apps interact with the controller via the northbound API. There is no standard northbound API or unified controller-to-app interface among different SDN frameworks. Applications can either be implemented as *internal* modules within the controller or as *external* processes decoupled from the controller. The SDN controller interacts with the network via the southbound API. The most commonly used protocol for southbound communication is OpenFlow [40].

2.1.3 Core Services. The core services inside the controller are responsible for reading, writing, modifying, and deleting data from the datastores. For example, *SAL Add-flow* service is responsible

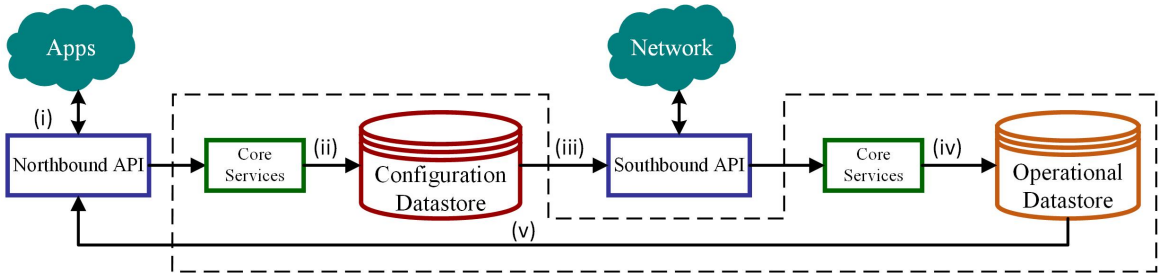

Figure 1: SDN Architecture. (i) An SDN app requests a configuration rule insertion/deletion in/from the configuration datastore via the northbound API. (ii) The request is met using core services in the configuration datastore. (iii) Configuration rules are executed in the network via southbound interface. (iv) The information about network traffic statistics is stored in the operational datastore using core services. (v) SDN apps can read operational statistics from operational datastore via the northbound API.

for adding configuration rules to the configuration datastore. *Flow programmer* is the service responsible for adding configuration rules in switches (via the southbound API). *OpenFlow Plugin* adds the rules to the operational datastore.

2.1.4 Datastores. The Network Management Datastore Architecture (NMDA) is the standard datastore architecture used by most SDN controllers [50]. It specifies two datastores: *configuration* and *operational*. The configuration datastore is a read and write datastore that contains aspirations about the network (i.e., the controller's desired state of the network). The operational datastore is a read only datastore that contains truth about the state of the network. SDN apps, via the northbound interface, can read, write, modify, and delete configuration rules inside the SDN configuration datastore. These configuration rules are later enforced in the network via the southbound interface. The network statistics are collected via the southbound interface and stored inside the operational datastore.

3 THREAT MODEL: MOTIVATING THREATS

In our threat model, we assume that the northbound channel (i.e., connection between a controller and SDN apps) is secure using HTTPS, etc. We assume the southbound channel (i.e., communication channel between a controller and network) is secure using OpenFlow, SSL, TLS, etc. We consider any interaction that an external entity (such as an SDN app) can make with the SDN configuration datastore as a threat. Consequently, we study four types of threats that arise due to the interaction between SDN apps and the SDN configuration datastore.

3.0.1 Threat 1 – Malicious SDN apps. The SDN apps establish a communication channel with the SDN configuration datastore prior to using the datastore resources. Dixit et al. [30] showed that two industry-based open-source SDN controllers (ODL and ONOS) do not inherently require authentication of SDN apps, which enables unauthenticated/malicious SDN apps to access the SDN configuration datastore, and tamper with the smooth operation of the network by messing with the existing rule database or exhausting the controller datastore resources or storing mutated configurations, etc. Both ODL and ONOS have realized this problem and ODL AAA [14] and ONOS AAA [6] aim to address and mitigate this issue. However, this security principle of authentication of SDN apps is yet to be quantified so that it is used uniformly across all SDN platforms.

3.0.2 Threat 2 – Privilege Escalation. The SDN apps interact with the SDN controller regarding the percentage capacity of configuration datastore that they are authorized to use. Dixit et al. [30] showed that two industry-based open-source controllers (ODL and ONOS) do not inherently authorize SDN apps to use a certain percentage of datastore resources. In the absence of proper authorization, an SDN app can raise its privilege and use more than its authorized share of datastore resources. Both ODL and ONOS later addressed this issue with their ODL AAA [14] and ONOS AAA [6]/SM-ONOS [57] services respectively. However, both ODL AAA and ONOS AAA/SM-ONOS do not cater for varying SDN app traffic and consequently varying network traffic in the case of small, medium, large, special purpose networks (Table 2). Again, the security design principle of authorization is yet to be quantified so that it can be uniformly used across all SDN platforms.

3.0.3 Threat 3 – Controller Overload. Ensuring that the SDN apps are not using more than their rightful authorized share of datastore resources require an accountability mechanism that dynamically updates each SDN apps' storage capacity after every rule insertion/deletion. Dixit et al. [30] showed an absence of such a mechanism for two SDN controllers (ODL and ONOS), which resulted in exhausting the configuration datastore resources and making the controller unavailable for further use. Both ODL AAA [14] and ONOS AAA/SM-ONOS [3, 57] have realized this problem but proposed partial accountability solutions with missing details on datastore capacity updates and testing of accountability function. Again, this security design principle of accountability is yet to be quantified so that it can be used across all SDN vendors.

3.0.4 Threat 4 – Information Inconsistency. Inconsistent information inside the configuration datastore leads to policy and configuration vulnerabilities, misconfigurations, and, in combination with missing authentication can lead to fraudulent rule insertion by a malicious SDN app [2]. The misconfigurations in the network can be intentional (by a malicious SDN app) or unintentional (due to conflicts among configuration rules entered by diverse SDN apps).

Table 2: Assessment of SDN Platforms for their Coverage on Mitigation Strategies for Threats 1, 2, 3, & 4.

	SDN Controller	Baseline Security Service	Mitigation Strategy (Threat 1)	Mitigation Strategy (Threat 2)	Mitigation Strategy (Threat 3)	Mitigation Strategy (Threat 4)
1.	FloodLight	SE-FloodLight [46]	Covered	Partly Covered	Partly Covered	Partly Covered
2.	OpenDayLight (ODL)	ODL AAA [14]	Covered	Partly Covered	Partly Covered	Not Covered
3.	Open Network Operating System (ONOS)	ONOS AAA [3]	Covered	Missing Details	Missing Details	Not Covered
		SM-ONOS [57]	Not Covered	Partly Covered	Not Covered	Partly Covered
4.	Ericsson Cloud SDN [7]	ODL AAA [14]	Covered	Partly Covered	Partly Covered	Not Covered
5.	Huawei Agile [10]	ODL/ONOS AAA	Covered	Partly Covered	Partly Covered	Not Covered
6.	Open Networking Platform	ODL AAA [14]	Covered	Partly Covered	Partly Covered	Not Covered
	OpenDayLight (ODL)	Eirene	Covered	Covered	Covered	Covered

We study the problem of inconsistent information in the SDN configuration datastore using four types of real-world complicated cases of rule conflicts.

(i) *Duplicate rules* (i.e., the rules belonging to Traffic Engineering (TE) and Load Balancer (LB) in Table 3). The duplicate rules match on all six fields that are source port, destination port, source IP, destination IP, priority, and action. The presence of duplicate rules, belonging to TE and LB, in the SDN configuration datastore can create confusion about the ownership of rules as well as damage rule database consistency. For example, if either one of the SDN apps (i.e., TE or LB) deletes the duplicate rule from the configuration datastore, it is kept and obeyed in the network due to the other SDN app (i.e., TE or LB).

(ii) *Rules conflicting on priority and action* (i.e., the rule belonging to FireWall (FW), which *conflicts on priority and action* with TE and LB in Table 3). The presence of FW's rule creates conflict in the rule database. Now, how to resolve this conflict? Should rule-based priority be used or another level of SDN app priority be used as FW rules usually supersede TE or LB.

(iii) *Rules conflicting on priority* (i.e., ARP proxy (ARP) rule, which *conflicts on priority* with FW and on *priority and action* with TE & LB in Table 3). These conflicts need to be resolved. (iv) *Rules conflicting on action* (i.e., Learning Switch (LS) rule that *conflicts on action* with ARP, *on priority* with TE & LB, and *on priority and action* with FW).

Such rule conflicts must be resolved so that the network operates in an intended manner and save the network administrator from hours of manual effort to debug any undesired behavior/traffic that may result from unresolved rule conflicts. The issue of detecting and handling rule conflicts in switch's TCAM table have been partly addressed by the SE-FloodLight [46] and SM-ONOS [57] service. At the time of this writing, there is no formal documentation regarding addressing rule conflicts in the SDN configuration datastore that could be uniformly used across all SDN platforms.

4 THE PROBLEM: OBSCURITY

Due to obscure definition of security design principles for safeguarding interactions between SDN apps and the SDN configuration datastore, the industry-based efforts have been unguided. Table 2 summarizes the extent to which mitigation strategies for the four threats emerging from the interactions between SDN apps-SDN configuration datastore have been addressed by multiple SDN platforms.

Refer to OpenDayLight (ODL) row in Table 2. The ODL AAA [14] service covers authentication of SDN apps so that only authenticated SDN apps are allowed access to the SDN configuration datastore; thus, the mitigation scheme for threat 1 is marked as covered. The ODL AAA [14] has an authorization function that authorizes SDN apps to use a certain percentage of SDN configuration datastore resources but does not cater for varying SDN app traffic and consequently changing network traffic requirements; thus, mitigation strategy for threat 2 is marked as partly covered. The ODL AAA [14] partly mitigates threat 3 by having an accountability part that keeps track of the API calls by an SDN app, the sessions maintained, etc.; but the exact details on how the configuration datastore capacity is allocated to an SDN app and updated with every configuration rule addition/deletion is missing. Thus, the mitigation scheme for threat 3 is marked as partly covered. Finally, the ODL AAA [14] service does not handle and resolve conflicts that arise among configuration rules belonging to the same/different SDN app/s in the datastore. Thus, mitigation scheme for threat 4 is marked as not covered in the ODL AAA [14] row of Table 2. The security assessment for other SDN platforms in Table 2 follow suit.

From Table 2, there is a lack of a baseline security service that completely addresses and proposes mitigation strategies for all four threats emerging from SDN apps-SDN configuration datastore interaction. The Eirene framework, based on four security design principles of AAAC, implemented and tested on ODL controller, addresses, and mitigates all four threats to ensure safe SDN apps-SDN configuration datastore interactions.

Table 3: Rule Conflicts.
(*Keys*: Src → source, Dst → Destination, Prt → Priority.)

SDN Apps	Src Port	Dst Port	Src IP	Dst IP	Prt	Action
TE	1212	1234	10.0.0.1/23	10.0.0.2/23	1000	Allow
LB	1212	1234	10.0.0.1/23	10.0.0.2/23	1000	Allow
FW	1212	1234	10.0.0.1/23	10.0.0.2/23	500	Deny
ARP	1212	1234	10.0.0.1/23	10.0.0.2/23	200	Deny
LS	1212	1234	10.0.0.1/23	10.0.0.2/23	200	Allow

5 RELATED WORK

Table 4 performs an assessment of notable academic efforts with respect to their completeness in covering mitigation strategies for threats 1, 2, 3, 4; and the assessment scale consists of three levels: covered (if the mitigation strategy for the respective threat has been covered with respect to SDN apps-SDN configuration datastore), partly covered (if the mitigation strategy has been covered with respect to the interaction between some other SDN components but can be extended to SDN apps-SDN configuration datastore interaction OR mitigation strategy has been covered but with missing details), and not covered (if the mitigation strategy for a particular threat has not been covered).

Table 4: Assessment of Notable Academic Efforts for their Coverage on Mitigation Strategies for Threats 1, 2, 3, & 4.

Year		Baseline Security Service	Proposed Approach	Threat 1 Mitigation	Threat 2 Mitigation	Threat 3 Mitigation	Threat 4 Mitigation
2015	1.	Secure Northbound Interface [22]	Trust Manager, Permission based Access Control	Covered	Partly Covered	Partly Covered	Not Covered
	2.	Oktian et al. [42]	Token Authentication for Application & User	Partly Covered	Not Covered	Not Covered	Not Covered
	3.	Floodguard [54]	Proactive Flow Rule Analyzer	Not Covered	Not Covered	Partly Covered	Not Covered
2016	1.	AuthFlow [39]	Host & Credential-based Authentication	Partly Covered	Partly Covered	Partly Covered	Not Covered
	2.	Fast Authentication Scheme [31]	Weighted Secure Context Information (SCI) Transfer	Partly Covered	Not Covered	Not Covered	Not Covered
	3.	AEGIS [44]	Dynamic Access Control	Not Covered	Partly Covered	Partly Covered	Not Covered
	4.	State-based Permission [41]	Permission Structure	Not Covered	Partly Covered	Not Covered	Not Covered
	5.	SDNShield [55]	Permission Structure	Not Covered	Partly Covered	Not Covered	Partly Covered
	6.	Lee et al. [38]	Permission Structure	Not Covered	Partly Covered	Not Covered	Not Covered
	7.	SDNsec [48]	Symmetric Key Cryptography	Not Covered	Partly Covered	Partly Covered	Not Covered
2017	1.	Application Authentication System [27]	Permission Authentication	Partly Covered	Partly Covered	Partly Covered	Partly Covered
	2.	Controller DAC [52]	Permission Structure, Dynamic Access Control	Not Covered	Partly Covered	Partly Covered	Not Covered
	3.	HanGuard [28]	Trust Model	Partly Covered	Partly Covered	Not Covered	Not Covered
	4.	Brew [45]	Detecting & Resolving Cross-layer Conflicts, Flow Rule Conflicts	Not Covered	Not Covered	Not Covered	Partly Covered
2018	1.	PROVSDN [53]	Data Provenance	Not Covered	Partly Covered	Not Covered	Not Covered
	2.	OAuthkeeper [43]	REST Access Parameter Conflict Algorithm	Not Covered	Partly Covered	Not Covered	Partly Covered
	3.	Synaptic [49]	Formal checker for SDN-based Security Policies	Not Covered	Not Covered	Not Covered	Partly Covered
	4.	Fine-grained Permission Management System [58]	User Isolation, Three level Permission Abstraction	Not Covered	Partly Covered	Not Covered	Not Covered
	5.	BENBI [56]	Scalable & Dynamic Access Control using Broadcast Encryption	Partly Covered	Partly Covered	Partly Covered	Not Covered
	6.	SecSDN-cloud [16]	User Authentication, Third-party Monitoring	Partly Covered	Not Covered	Partly Covered	Not Covered
2019	1.	THP (The Hidden Pattern) [32]	Graphics Password + Digital Challenge	Partly Covered	Not Covered	Not Covered	Not Covered
	2.	SDN-RBAC [17]	Formal Role-based Access Control Model	Not Covered	Partly Covered	Not Covered	Not Covered
	3.	Controller Oblivious Dynamic Access Control [33]	Flow Isolation based Model	Not Covered	Partly Covered	Not Covered	Not Covered
	4.	SDNSOC [25]	Flow Rule Conflict Detection & Resolution	Not Covered	Not Covered	Not Covered	Partly Covered
	5.	BEAM [51]	Behavior-based Access Control	Not Covered	Partly Covered	Not Covered	Not Covered
	6.	Formal Role-based Access Control [19]	Formalized Access Control Model	Not Covered	Partly Covered	Not Covered	Not Covered
2020	1.	BACC-SDN [24]	Block-chain based Access Control	Not Covered	Partly Covered	Not Covered	Not Covered
	2.	ParaSDN [18]	Parameterized Roles and Permission-based Access Control Model	Not Covered	Partly Covered	Not Covered	Not Covered
	3.	SDN-RBACa [20]	Custom Permission based Access Control	Not Covered	Partly Covered	Not Covered	Not Covered
	4.	Audi-SDN [37]	Automatic Detection of Network Policy Inconsistencies	Not Covered	Not Covered	Not Covered	Partly Covered
2021	1.	ROCA [21]	Auto-resolving Overlapping & Conflicts in ACL policies	Not Covered	Not Covered	Not Covered	Partly Covered
	2.	SEAPP [34]	REST API Based Access Control	Not Covered	Partly Covered	Not Covered	Not Covered
	3.	SILedger [47]	Block-chain & Attribute-based Encryption Access Control	Not Covered	Partly Covered	Not Covered	Not Covered
	4.	Private Block-chain based Access Control [23]	Attribute Based Encryption, Certificate-Based Access Control Protocol	Not Covered	Partly Covered	Not Covered	Not Covered
2022	1.	Lee et al. [36]	Automated Fuzz-Testing Framework	Not Covered	Not Covered	Not Covered	Partly Covered
		Eirene	128-bit Encryption Key, Three Different Modes for Resource Allocation, Conflict Detection and Resolution	Covered	Covered	Covered	Covered

Refer to Controller DAC [52] (proposed in 2017) in Table 4. It does not propose a mitigation strategy for malicious SDN apps (threat 1) and information inconsistency (threat 4). Thus, threat 1 and threat 4 mitigation is marked as not covered in the Controller DAC [52] row of Table 4. Authorization has been covered to mitigate threat 2; however, varying application traffic and network traffic requirements have not been covered. Thus, threat 2 mitigation strategy has been marked as partly covered in the Controller DAC [52] row of Table 4. Next, the mitigation strategy of threat 3 is marked as partly covered because the exact details of datastore capacity updates, etc. are missing. The security assessment of other notable academic efforts in Table 4 follow suit (with more details in Appendix E). Table 4 indicates a lack of a security service that addresses and proposes mitigation strategies for all four threats that arise due to the interaction between SDN apps and the SDN configuration datastore.

6 SYSTEM DESIGN

The absence of universal security design principles and consequently a security service that addresses and mitigates all four threats associated with the interaction between SDN apps and SDN configuration datastore provides the motivation for the design of the Eirene framework. The Eirene framework is a AAAC app that: mitigates *threat 1* using the security design principle of authentication of SDN apps (to block malicious SDN apps from accessing datastore), *threat 2* using the security design principle of authorization of SDN apps, *threat 3* by keeping SDN apps accountable for the amount of SDN datastore resources that they use (i.e., the security design principle of accountability), *threat 4* by detecting and resolving conflicts among rules belonging to same/different SDN app/s. The framework has four corresponding components.

First, the Eirene framework has a *resource allocation* component that is based on the security design principles of authorization and accountability of SDN apps. The resource allocation component supports three different modes of operation to handle the varying SDN app traffic and consequently varying network traffic requirements for small, medium, large scale, and special purpose networks. For example, a small campus network may benefit from an equal or fair resource allocation while a science demilitarized network may require role-based resource allocation. To handle different kinds of networks (small, medium, large, special purpose, etc.) and different SDN apps requirements, the framework has a *resource allocation* component that distributes datastore resources among SDN apps

using three different modes of operation with proper authorization and accountability.

Second, when allocating resources, several parameters such as datastore capacity, total number of SDN apps accessing the datastore, etc., need to be set by the network practitioner. This functionality is provided by the *network practitioner* component, which has five sub-parameters that are set by the network practitioner depending on the mode of operation used. *Authentication* is provided by this component using a 128-bit encryption key generator that generates a unique key for each legitimate SDN app. This key is shared by the network practitioner with SDN apps beforehand.

The *third* important thing is ensuring that the requirements of SDN apps are met in the best possible way. This is accomplished by the *SDN apps* component, which uses five sub parameters to measure and evaluate the most suited mode of resource allocation for SDN apps.

The *fourth* and final security design principle is ensuring consistency of information in the SDN configuration datastore. The *conflict handling* component of the Eirene framework performs this task and maintains consistent information in the datastore by resolving conflicts among rules belonging to same/different SDN app/s.

6.1 Resource Allocation Component

The resource allocation component of the Eirene framework consists of three modes of operation to cater for varying SDN app traffic and consequently varying network traffic requirements.

6.1.1 Fair or Equal Resource Allocation (FRA). Fair or equal resource allocation fairly or equally divides resources among all the SDN apps accessing the SDN configuration datastore. "Fairness" is achieved by statically giving a unified *threshold* to all SDN apps and then keeping them accountable by dynamically updating the allocated storage capacity upon every configuration rule insertion/deletion Eq. 1 in Table 5 formally shows resource allocation for this mode of operation. This mode is preferred when all SDN apps have roughly equal storage requirements (e.g., a small campus network).

6.1.2 Role-based Resource Allocation (RBRA). Role-based resource allocation allocates varying amount of datastore resources to SDN apps depending on their roles. This mode gives three different roles to SDN apps: *Tier1*, *Tier2*, and *Tier3* with *Tier1* apps having the highest priority and *Tier3* apps having the lowest priority [26, 52]. This mode allocates resources using Eq. 2 in Table 5. This mode is used when it is desired to give different roles to SDN apps due to unequal storage requirements (e.g., a science demilitarized zone setting).

6.1.3 Role-based Resource Allocation as an Optimization Problem (RBRAOP). This mode of operation uses the concept of resource pooling and revolves around the principle of maximizing the utility of SDN apps subject to datastore capacity constraints. Eq. 3 in Table 5 shows the mathematical representation for this mode. In this mode, all SDN apps share a dynamic *threshold*, which is updated with every configuration rule insertion/deletion inside the configuration datastore. This mode of operation uses the module *active rule deletion manager* (the details are in Appendix A), which carefully deletes rules (if there is no more space in the datastore) to make room for more important rules (i.e., the rules with either high rule priority or belonging to SDN apps that have high app priority/App-ID). This is the preferred mode of operation for very large networks with dynamically changing SDN app traffic as well as network traffic requirements.

6.2 Network Practitioner Component

In this subsection, we propose five parameters for the network practitioner component of the Eirene framework. These parameters are set by the network practitioner depending on the mode of operation (i.e., FRA, RBRA, RBRAOP) used.

6.2.1 Datastore Capacity (C). The datastore capacity, C, is set by the network practitioner to an optimal value (such that the SDN controller neither runs out of memory nor undergoes performance degradation). This upper limit on the SDN configuration datastore capacity is estimated by the network practitioner prior to distributing resources among SDN apps.

6.2.2 Number of Applications (N). The network practitioner is responsible for setting the total/anticipated number of SDN apps accessing the configuration datastore to a value N. The datastore capacity, C, is divided among N SDN apps in some ratio depending on the mode of operation used. FRA uses this parameter to equally divide resources among all SDN apps. RBRA uses a variation of this parameter such that the total number of SDN apps (N) is divided based on the role given to each SDN app. Eq. 2 in Table 5 shows N_{Tier1}, which is the total number of SDN apps with app role Tier1, N_{Tier2} is the total number of SDN apps with app role Tier2, and N_{Tier3} is the total number of SDN apps with app role Tier3. This parameter is not used by RBRAOP because datastore capacity is dynamically allocated in that mode.

6.2.3 Key (K). The network practitioner assigns a unique 128-bit encryption key (i.e., K_i) to each SDN app i using an encryption key generator and maintains a dictionary of keys (that is used to authenticate an SDN app). The Eirene framework has the flexibility to use any other sophisticated authentication scheme (e.g., 256-bit, 512-bit, 1024-bit encryption key) for SDN apps. The assignment of keys to SDN apps is done via a 2-way communication between an SDN app and network practitioner; this process of key assignment is considered secure.

6.2.4 Application Identifier (App-ID). The network practitioner assigns a unique App-ID based priority to each SDN app to track ownership of rules. The process of App-ID assignment is made flexible by using sliding window technique that is capable of accommodating new SDN apps with varying levels of priority.[1] This

[1]The sliding window technique is as follows. Consider four SDN apps accessing the configuration datastore: (i) FW, (ii) LB, (iii) TE, (iv) ARP. Here, N = 4 and the range for App-ID is (1, ..., *total window size*), where total window size can be N*10 or N*100 or N*1000 or N*(any other constant) depending on the size of the network and the number of existing plus anticipated SDN apps accessing configuration datastore. Suppose network practitioner has assigned an App-ID $= 50$ to LB. Next, 10 new SDN apps arrive at the datastore and need a higher priority than LB. These 10 new apps are catered for by giving them an App-ID < 50. Similarly, if 10 more apps arrive at the datastore that have lower priority than LB then they are assigned an App-ID > 50. The exact details of flexible App-ID allocation is omitted from the paper for brevity. We further assume that App-ID allocation is a secure process between network practitioner and SDN apps. Thus, it is not possible to spoof App-ID.

Table 5: The formal details for the four components of the Eirene framework in three different modes.

		Fair Resource Allocation (FRA)	Role-based Resource Allocation (RBRA)	Role-based Resource Allocation as an Optimization Problem (RBRAOP)
1.	Resource Allocation Component	$AC_i = \frac{C \ (\#\ \text{of rules})}{N \ (\#\ \text{of applications})}$ (1) where AC_i is the allocated capacity for an application i, C is datastore capacity, and N is the number of SDN applications accessing datastore resources.	$AC_i = \frac{jg\%\ \ of\ \ C \ \ (\#\ \text{of rules})}{N_g \ (\#\ \text{of}\ N_g\ \text{applications})}$ (2) where g = Tier1, Tier2, and Tier3; $j_{\text{Tier1}} = x$, $j_{\text{Tier2}} = y$, and $j_{\text{Tier3}} = z$; x, y, and z is the % of C that is equally divided among Tier1, Tier2, and Tier3 applications;N_{Tier1} = Number of Tier1 applications, N_{Tier2} = Number of Tier2 applications, and N_{Tier3} = Number of Tier3 applications.	$Threshold = C$ maximize $\sum_i \gamma_i$ subject to $S \leqslant C$ (3) where S is the sum of the total number of rules in configuration datastore; γ_i is application satisfaction. The aim is to have high γ_i for all applications.
2.	Network Practitioner Component	(i) Datastore Capacity (C). (ii) Number of Applications (N). (iii) Application Identifier (App-ID).	(i) Datastore Capacity (C). (ii) Number of Applications (N). (iii) Application Identifier (App-ID). (iv) Application Role (App-Role).	(i) Datastore Capacity (C). (ii) Application Identifier (App-ID).
3a.	Residual Storage Capacity (α_i)	$AC_i = Threshold$ (initially computed using Eq. 1 or Eq.2) $\alpha_i = AC_i - DS_i$ (where $DS_i \rightarrow$ Desired Storage capacity for an application i) $AC_i = \begin{cases} \alpha_i & \text{if } \alpha_i \geqslant 0 \\ 0 & \text{otherwise} \end{cases}$ (4) such that $0 \leqslant AC_i \leqslant C$ and $DS_i \geqslant 0$		$Threshold = C$ (initially) $\alpha_i = Threshold - DS_i$ $Threshold = \begin{cases} \alpha_i & \text{if } \alpha_i \geqslant 0 \\ 0 & \text{otherwise} \end{cases}$ (5) such that $0 \leqslant Threshold \leqslant C$ and $DS_i \geqslant 0$
3b.	Unmet Storage Requirement (β_i)	$\beta_i = \mu \cdot DS_i + \beta_i$ where $\mu = \begin{cases} 1 & \text{if } \alpha_i < 0 \\ 0 & \text{otherwise} \end{cases}$ (6) and $\beta_i = 0$ (initially) where DS_i is the Desired Storage capacity for an application i.		
3c.	Actively Deleted Rules (ζ_i)	$\zeta_i = \zeta_i + \eta_i$ and $\eta_i = 0$ and $\zeta_i = 0$ (initially) (7) where η_i is the number of actively deleted rules.		$\zeta_i = \zeta_i + \eta_i$ where $\zeta_i = 0$ (initially) (8) $Threshold = Threshold + \eta_i$ ($\because$ space created)
3d.	Application Satisfaction (γ_i)	$\gamma_i = k_o + \alpha_i - \beta_i - \zeta_i$ where $k_0 = 100$ is a constant. (9) (If $\alpha_i \geqslant 0$ then γ_i increases given that $\beta_i = 0$ and $\zeta_i = 0$. Similarly, if $\beta_i > 0$ and $\zeta_i > 0$, then γ_i decreases).		
3e.	Overall Success	Overall Success $= \sum_{i=1}^{N} \gamma_i$ where N is the total number of applications and γ_i is the application satisfaction for an application i. (10)		
4.	Conflict Handling Component	$\alpha_i = \alpha_i + \sigma_i + \kappa_i$ $AC_i = \begin{cases} \alpha_i & \text{if } \alpha_i \geqslant 0 \\ 0 & \text{otherwise.} \end{cases}$ $\beta_i = \beta_i + \sigma_i$ (11) $\zeta_i = \zeta_i + \kappa_i$ where σ_i is a conflicting rule that cannot be stored and κ_i is a conflicting rule that is actively deleted.		$\alpha_i = Threshold + \sigma_i + \kappa_i$ $Threshold = \begin{cases} \alpha_i & \text{if } \alpha_i \geqslant 0 \\ 0 & \text{otherwise.} \end{cases}$ (12) $\beta_i = \beta_i + \sigma_i$ $\zeta_i = \zeta_i + \kappa_i$

parameter is used to resolve rule conflicts and maintain information consistency inside the configuration datastore (the details are in Appendix B, Appendix C) as well as used to evict rules (if needed) to make room for more important rules by *active rule deletion manager* (Appendix A).

6.2.5 Application Role (App-Role). This parameter is used only by RBRA, and it gives three different roles to SDN apps. (i) Tier1 by setting App-Role = 0. (ii) Tier2 by setting App-Role = 1. (iii) Tier3 by setting App-Role = 2. The SDN apps with App-Role = 0 have the highest priority and are allocated a greater portion of the datastore resources while apps with App-Role = 2 have the lowest priority and are allocated a lesser portion of the datastore resources.

The parameters that the network practitioner needs to set in each mode of operation is summarized in the Network Practitioner Component row of Table 5.

6.3 SDN Apps Component

In this subsection, we introduce five parameters for the SDN apps component of the Eirene framework, which are used to evaluate the most suitable mode of operation for a particular type of network with a specific SDN app traffic.

6.3.1 Residual Storage Capacity (α). The residual storage capacity for an SDN app i, denoted as α_i, is a measure of the remainder of memory resources of an SDN app. Depending on the mode of operation used, it is formally defined by Eq. 4 or Eq. 5 in Table 5. It is *desirable* to have $\alpha_i \geqslant 0$ as it ensures that an app's storage requirements have been met.

6.3.2 Unmet Storage Requirement (β). This parameter measures the unmet storage request of an SDN app due to lack of space (i.e., $\alpha_i < 0$) or to maintain consistent information inside the configuration datastore. Mathematically, it is given by Eq. 6 in Table 5. The *goal* is to always have $\beta_i = 0$ as it guarantees that there is no unmet storage requirement.

6.3.3 Actively Deleted Rules (ζ). Actively deleted rules, represented as ζ_i for an SDN app i, are the rules that are deleted from the datastore either to make room for more important rules (i.e., *active rule deletion manager*, Appendix A) or to maintain information consistency (i.e., *duplicate and conflicting action rule manager*, Appendix B and *conflicting priority and priority action rule manager*, Appendix C). The formal representation is given by Eq. 7 and Eq. 8 in Table 5. The aim is to have $\zeta_i = 0$, which is intuitive as SDN apps typically do not want their rules to be deleted from the SDN configuration datastore without their willingness.

6.3.4 Application Satisfaction (γ). This is a measure of how satisfied an SDN app is after using datastore resources. It is formally given by Eq. 9 in Table 5. Intuitively, application satisfaction is high if an SDN app has sufficient memory resources to fulfill its storage requirements (i.e., $\alpha_i \geqslant 0$), all storage requirements are met (i.e., $\beta_i = 0$), and there is no active rule deletion (i.e., $\zeta_i = 0$).

6.3.5 Overall Success. The overall success is a measure of total application satisfaction. Based on this overall success, a decision about the most suitable mode of operation (to be used in a particular situation) is made. For example, let overall success be 100, 200, and 300 for FRA, RBRA, and RBRAOP respectively. Then the preferred

Algorithm 1 Pseudocode for *Conflict Handling Component.*

```
def DuplicateAndConflictingActionRuleManager
  if (rules belong to different SDN apps) // Principle 1
    Keep the rule with lower App-ID.
  else if (rules belong to same SDN app) // Principle 2
    Keep the latest rule.
def ConflictingPriorityAndPriorityActionRuleManager
  if (rules belong to different SDN apps) // Principle 3
    Keep the rule with lower App-ID.
  else if (rules belong to same SDN app) // Principle 4
    Keep the rule with high rule priority.
```

mode of operation is RBRAOP (because it has the highest overall success). The formal representation is given by Eq. 10 in Table 5.

6.4 Conflict Handling Component

The conflict handling component of the Eirene framework consists of two sub components and four corresponding principles, which are summarized by Algorithm 1.

6.4.1 Duplicate and Conflicting Action Rule Manager. This module resolves conflicts between duplicate rules and rules that conflict on action using *principle 1* and *principle 2* of Algorithm 1. *Principle 1* resolves such a conflict between two different SDN apps by keeping/storing the rule belonging to the SDN app with lower App-ID. This is intuitive because an SDN app with lower App-ID has high priority. By *principle 2*, for same SDN apps, the conflict is resolved by keeping the latest rule. Again, this is intuitive as the latest input represents latest requirement. The formal details are in Appendix B.

6.4.2 Conflicting Priority and Priority Action Rule Manager. This module settles the battle between rules that either conflict only on priority or on both priority and action. Algorithm 1 explains the operating principles for this module (formal details are in Appendix C). According to *principle 3*, if such a conflict exists between different SDN apps, the rule belonging to an SDN app with lower App-ID is kept (which is intuitive as an SDN app with lower App-ID has higher priority). For conflict between rules belonging to the same SDN app, the rule with higher rule priority is kept (according to *principle 4*). Again, this is intuitive as the rule with high rule priority is more important.

6.5 Implementation

We implemented a prototype of the Eirene framework as a Java application on ODL controller. The minimum memory (space) and run-time (time) requirements for the Eirene framework are:

$$\begin{aligned} &\text{Space Complexity} : O \text{ (Number of Rules)} \\ &\text{Time Complexity} : O \text{ (Number of Rules)} \end{aligned} \tag{13}$$

Both space and time complexity of the Eirene framework is a direct function of the number of configuration rules in the SDN configuration datastore, which is intuitive because every incoming rule is compared with the existing rule database. It is important to mention that the Eirene framework has been implemented as a Java application on the ODL controller, but the design can be ported to any other controller. As part of future work, we plan to use hash tables to reduce the latency of the system. However, worst-case complexity of the system, using hash tables, would still be O (Number of Rules).

7 FUNCTIONALITY DEMONSTRATION

We use two practical use case scenarios to explain the working of the Eirene system.

7.1 Scenario 1: A Multi-SDN Apps AAA Use Case

We consider a practical use case scenario where multiple SDN apps can access the SDN configuration datastore, which has a capacity to store 50,000 configuration rules. This capacity is to be divided among 50 SDN apps, note that the network practitioner typically over provisions the number of SDN apps so that the new SDN apps can be accommodated. The ODL controller is a Java Virtual Machine (JVM), and it is possible to store 50,000 configuration rules by setting JVM heap size to 4GB. The network practitioner sets the datastore capacity at C = 50,000 configuration rules and the number of SDN apps at N = 50. The App-IDs are assigned using a sliding window technique such that it is possible to cater for new SDN apps. Table 6 shows the details of parameter assignment for this multi-SDN apps scenario for each mode of operation.

7.1.1 Authentication of SDN Apps. All SDN apps including the 5 SDN apps shown in Table 6: TE, LB, FW, ARP, LS; are given a unique 128-bit encryption key by the network practitioner. The unique keys are used for authentication of each SDN app before every rule insertion/deletion inside the SDN configuration datastore.

7.1.2 Authorization of SDN Apps. The authenticated SDN apps are authorized to use a certain percentage of datastore resources by the network practitioner depending on the mode of operation used. For FRA, the datastore resources are fairly or equally distributed among 50 SDN apps. Using Eq. 1 in Table 5, the capacity allocated to all 50 SDN apps is 1,000 configuration rules. Table 6 shows 5 such SDN apps and the capacity allocated to them for this mode. However, datastore resources are provisioned such that an additional 45 SDN apps can be accommodated, and all 50 apps have a capacity to store a maximum of 1,000 configuration rules.

The mode RBRA assigns different roles to different SDN apps, based on which datastore resources are allocated. For a total number of 50 SDN apps, the case study depicted in Table 6, assigns tier1 role to 20 SDN apps, tier2 role to 10 SDN apps, and tier3 role to 20 SDN apps. Thus, Table 6 sets tier1 role for FW by setting App-Role = 0; tier2 role for TE and LB by setting App-Role = 1 for both; tier3 role for ARP and LS by setting App-Role = 2 for both. Using Eq. 2 in Table 5, 50% of datastore resources are allocated to tier1 SDN apps, which is a capacity of 25,000 configuration rules to be divided among 20 SDN apps (i.e., 1,250 configuration rules for each SDN app with App-Role = 1). Next, 30% (i.e., 15,000 configuration rules) of datastore resources are allocated cumulatively to 10 tier2 SDN apps (i.e., 1,500 configuration rules for each SDN app with App-Role = 2). Thus, Table 6 shows a storage capacity of 1,500 configuration rules for both TE and LB. Finally, 20% (i.e., 10,000 configuration rules) of datastore resources are allocated to 20 SDN apps with tier3

Table 6: Scenario 1 – A Multi-SDN Apps AAA Use Case (with $C = 50,000$ configuration rules and $N = 50$ SDN apps).

	Mode of Operation	Traffic Engineering (TE)	Load Balancer (LB)	Firewall (FW)	ARP Proxy (ARP)	Learning Switch (LS)
		App-ID$_{TE}$ = 200	App-ID$_{LB}$ = 300	App-ID$_{FW}$ = 100	App-ID$_{ARP}$ = 400	App-ID$_{LS}$ = 500
1.	Fair Resource Allocation (FRA)	AC$_{TE}$ = 1,000	AC$_{LB}$ = 1,000	AC$_{FW}$ = 1,000	AC$_{ARP}$ = 1,000	AC$_{LS}$ = 1,000
2.	Role-based Resource Allocation (RBRA)	AC$_{TE}$ = 1,500 App-Role = 1	AC$_{LB}$ = 1,500 App-Role = 1	AC$_{FW}$ = 1,250 App-Role = 0	AC$_{ARP}$ = 500 App-Role = 2	AC$_{LS}$ = 500 App-Role = 2
3.	Role-based Resource Allocation as an Optimization Problem (RBRAOP)	AC$_{TE}$ = 50,000	AC$_{LB}$ = 50,000	AC$_{FW}$ = 50,000	AC$_{ARP}$ = 50,000	AC$_{LS}$ = 50,000

Table 7: Scenario 2 – A Multi-SDN Apps Conflict Handling Use Case. (*Keys*: ✕ → Rule is not stored/deleted, ❙ → Rule is partially stored/deleted, ✓ → Rule is stored.)

	SDN App	App-ID	Src Port	Dst Port	Src IP	Dst IP	Priority	Action	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
R1.	TE	200	1212	1234	10.0.0.1/23	10.0.0.*/23	1000	Deny	✓									
R2.	TE	200	1212	1234	10.0.0.1/23	10.0.0.2/23	65535	Allow	❙	✓								
R3.	LB	300	1212	1234	10.0.0.1/23	10.0.0.2/23	1000	Allow	❙	✓	✕							
R4.	FW	100	1212	1234	10.0.0.1/23	10.0.0.2/23	500	Deny	❙	✕	✕	✓						
R5.	LB	300	4444	Any	172.168.0.1/32	172.168.0.2/32	65535	Deny	❙	✕	✕	✓	✓					
R6.	TE	200	1212	1234	10.0.0.1/23	10.0.0.2/23	1000	Deny	❙	✕	✕	✓	✓	✕				
R7.	ARP	400	1212	1234	10.0.0.1/23	10.0.0.2/23	500	Allow	❙	✕	✕	✓	✓	✕	✕			
R8.	LS	500	1212	1234	10.0.0.1/23	10.0.0.2/23	200	Allow	❙	✕	✕	✓	✓	✕	✕	✕		
R9.	FW	100	1212	1234	10.0.0.1/23	10.0.0.*/23	50000	Deny	✕	✕	✕	✕	✓	✕	✕	✕	✓	
R10.	FW	100	1212	1234	10.0.0.1/23	10.0.0.2/23	65535	Allow	✕	✕	✕	✕	✓	✕	✕	✕	❙	✓

role. Thus, an allocated memory capacity of 500 configuration rules each for both ARP and LS for this mode of operation.

The mode RBRAOP dynamically allocates resources. Therefore, an initial maximum datastore capacity of 50,000 configuration rules is allocated to all 50 SDN apps and the capacity is dynamically updated/reduced as the resources gets used up.

7.1.3 Accountability of SDN Apps. The accountability function of the Eirene system adjusts the allocated capacity corresponding to each SDN app. For example, AC$_{TE}$ = 1,000 rules for FRA (Table 6). If TE has stored 100 rules in the configuration datastore then its corresponding allocated capacity changes from 1,000 to 900 rules. Similarly, rule deletion increases capacity for the corresponding SDN app.

7.2 Scenario 2: A Multi-SDN Apps Conflict Handling Use Case

In this scenario, we use real-world complicated cases of rule conflicts to explain the conflict handling part of the Eirene system. Table 7 shows 10 such configuration rules belonging to 5 SDN apps.

Refer to Table 7, R1, a rule belonging to TE app with App-ID = 200, forbids communication between host with IP address 10.0.0.1/23 and all hosts with IP address in the range 10.0.0.*/23. Initially the datastore is empty, therefore the rule is stored. Hence (✓) for R1 in the R1 row of Table 7.

Next, R2, a rule belonging to TE app, allows communication between host with IP address 10.0.0.1/23 and host with IP address 10.0.0.2/23. This rule R2 varies on priority and action with R1. The conflicting priority and priority action rule manager module of the conflict handling component resolves this conflict (using *principle 4* as given by Algorithm 1) by partially deleting R1 such that traffic between host with IP address 10.0.0.1/23 and 10.0.0.2/23 is allowed but blocked between 10.0.0.1/23 and 10.0.0.*/23 (with the exception of 10.0.0.2/23). Therefore, a partial deletion (❙) for R1 and storage (✓) for R2 in the R2 row of Table 7.

The rule R3 belonging to LB conflicts on priority with R2. The conflict is resolved by the conflicting priority and priority action rule manager module of the conflict handling component using *principle 3* of Algorithm 1. Because LB has a high App-ID than TE, R3 is not stored as indicated by (✕) in the R3 row of Table 7. Next, rule R4, which belongs to FW conflicts on priority and action with R2. The problem is handled by the conflicting priority and priority action rule manager module of the conflict handling component using *principle 3* (Algorithm 1). As a result, R2 is deleted and R4 is stored as indicated by (✕) for R2 and (✓) for R4 in the R4 row of Table 7.

The rule R5, belonging to LB, does not conflict with the existing rules in the configuration datastore (i.e., R1 and R4) so it is stored as indicated by (✓) in the R5 row of Table 7. Next, R6, belonging to TE, conflicts on priority with R4. The conflicting priority and priority action rule manager module of the conflict handling component of the Eirene framework resolves this issue using *principle 3* (Algorithm 1). Because TE has a high App-ID than FW, this rule is not stored as indicated by (✕) in R6 row of Table 7.

Next, R7, belonging to ARP, conflicts on action with an existing rule in the configuration datastore (i.e., R4). The duplicate and conflicting action rule manager module of the conflict handling component of the Eirene framework uses *principle 1* (Algorithm 1) to resolve the issue. As a result, R7 is not stored because it has a high App-ID than FW. This is indicated by (✕) in the R7 row of Table 7. The rule R8, belonging to LS conflicts on priority and action with R4. It is handled using *principle 1* (Algorithm 1) of the conflict handling component. As a result, R8 is not stored (✕).

The rule R9, belonging to FW, conflicts on priority with R1, on priority and action with R2, and on priority with R4. The conflict between R1 and R2 is handled using *principle 1* of the conflicting priority and priority action rule manager. As a result, R1 and R2 are deleted from the datastore. The conflict with R4 is handled using *principle 2* of the conflicting priority and priority action rule manager and is resolved by deleting R4. Thus, R9 is stored inside the datastore as indicated by (✓) in the R9 row of Table 7. Finally, R10, belonging to FW conflicts on priority and action with one of the rules stored as R9. Using *principle 4*, the rule that denies communication between hosts with IP address 10.0.0.1/23 and 10.0.0.2/23

Table 8: Statistics for A Multi-SDN Apps Small Scale Security Evaluation (using real-world dataset containing 36 rules).

	Fair Resource Allocation (FRA)	Role-based Resource Allocation (RBRA)	Role-based Resource Allocation as an Optimization Problem (RBRAOP)
Firewall (FW)	App-ID$_{FW}$ = 1	App-ID$_{FW}$ = 1, App-Role$_{FW}$ = 0	App-ID$_{FW}$ = 1
Traffic Engineering (TE)	App-ID$_{TE}$ = 2	App-ID$_{TE}$ = 2, App-Role$_{TE}$ = 2	App-ID$_{TE}$ = 2
Load Balancer (LB)	App-ID$_{LB}$ = 3	App-ID$_{LB}$ = 3, App-Role$_{LB}$ = 1	App-ID$_{LB}$ = 3
Allocated Capacity	AC$_{FW}$ = 10 rules (Eq. 1)	AC$_{FW}$ = 15 rules (Eq. 2)	AC$_{FW}$ = 30 rules (Eq. 3)
	AC$_{TE}$ = 10 rules (Eq. 1)	AC$_{TE}$ = 9 rules (Eq. 2)	AC$_{TE}$ = 30 rules (Eq. 3)
	AC$_{LB}$ = 10 rules (Eq. 1)	AC$_{LB}$ = 6 rules (Eq. 2)	AC$_{LB}$ = 30 rules (Eq. 3)
Overall Success	293	293	319

Table 9: A Multi-SDN Apps Small Scale Security Evaluation using Real-World Dataset.

	Averaged Response Time (ns)																	
Mode	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18
FRA	77.64	16.17	10.44	15.92	14.62	12.07	23.02	26.82	77.49	72.53	∞	∞	∞	∞	∞	11.9	18.14	18.50
RBRA	86.69	11.03	17.19	97.99	50.68	20.20	16.87	18.07	35.67	13.32	34.76	18.23	13.81	47.67	∞	14.18	48.92	26.67
RBRAOP	81.05	16.54	15.12	19.35	17.48	16.34	35.76	38.88	19.23	29.24	22.55	17.76	15.52	11.71	∞	21.98	121.18	109.48
Without Eirene	53.50	19.22	12.84	15.63	22.75	15.53	47.04	69.96	29.24	79.72	66.30	39.46	26.73	57.85	53.31	41.75	75.62	41.12
	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30	R31	R32	R33	R34	R35	R36
FRA	28.43	∞	26.68	24.35	81.23	∞	∞	18.53	14.33	31.80	68.20	78.63	∞	14.23	56.27	60.19	90.16	48.91
RBRA	38.57	∞	12.29	38.68	33.10	∞	∞	∞	11.96	16.21	8.52	33.67	∞	6.37	13.41	∞	∞	∞
RBRAOP	76.95	∞	16.53	45.43	59.68	∞	∞	∞	10.40	7.74	8.65	38.34	∞	32.58	53.59	75.47	41.01	59.30
Without Eirene	36.36	81.66	33.45	21.72	21.22	12.72	41.99	28.66	25.03	34.76	27.29	12.15	8.50	15.41	8.51	10.75	15.75	11.91

with a priority of 50,000 is deleted (thus partial deletion (◧) for R9) and R10 is stored (✓) as indicated by R10 row of Table 7.

8 SECURITY EVALUATION

In this section, we evaluate the security of the Eirene framework on a small as well as on a large scale.

8.1 A Multi-SDN Apps Small Scale Security Evaluation

This subsection explains the dataset used for experimentation, the experimental setup, the testing conditions, and the results.

Real-World Dataset. The dataset used for small scale security evaluation of the Eirene framework consists of 36 real-world configuration rules, which belong to 3 SDN apps (FW, TE, LB), and the dataset includes complicated cases of rule conflicts. The rule composition in the dataset is such that 14 rules belong to the FW app (shown as R1–R14 in Table 9), 10 rules that belong to the TE app (shown as R15–R24 in Table 9), and 12 rules that belong to the LB app (shown as R25–R36 in Table 9). The composition of complicated cases of rule conflicts in the dataset is such that:

$$\begin{aligned} &\textsc{Duplicate Rules}: \text{R1} \,\&\, \text{R15}, \text{R3} \,\&\, \text{R24}, \text{R17} \,\&\, \text{R31}, \\ &\textsc{Conflicting Rules}: \text{R5} \,\&\, \text{R20}, \text{R8} \,\&\, \text{R25}, \text{R14} \,\&\, \text{R26} \end{aligned} \tag{14}$$

This dataset has been obtained from the Research Computing Department in our university, which is responsible for implementing Access Control Lists (ACL) and firewall rules on the traffic entering campus network as well as applying load balancing and traffic engineering policies to effectively manage the traffic throughout the campus network.

Experimental Setup. The controller used for experimentation is ODL, which is a Java Virtual Machine (JVM) and its JVM heap size is fixed at 4GB. The configuration datastore capacity of the controller is set at 30 configuration rules. The rule requests arrive at the datastore in a sequential manner such that R1 comes first, then rule R2, and so on. We next test the authentication, authorization, accountability, and conflict handling functions of the Eirene framework.

Test 1 – Threat 1 Mitigation. The Eirene framework uses authentication to filter out malicious SDN apps by using a 128-bit unique key generator (the unique key is shared by the network practitioner with SDN apps). We tested the authentication function of the Eirene system by verifying that SDN apps are allowed to insert or delete a configuration rule in the SDN configuration datastore iff they use the correct 128-bit encryption key. Failure to do so results in denying access to utilize configuration datastore resources by the SDN controller.

Test 2 – Threat 2 Mitigation. The Eirene framework uses authorization to mitigate threat 2. To test the correct working of the authorization function of the Eirene framework, we tested if the SDN apps are allocated datastore capacity based on their roles (depending on the mode of operation used, i.e., FRA, RBRA, RBRAOP). Table 8 shows the allocated capacity for each SDN app in a particular mode of operation along with the values for overall success. The SDN apps can then use the correct 128-bit encryption key to use this allocated capacity for rule insertion or deletion. Refer to Table 9, a response time of ∞ indicates that the configuration rule is not stored. Furthermore, the response time measurements shown in Table 9 are averaged over 10 iterations.

The averaged response time measurements for FRA in Table 9 shows ∞ at R11, R12, R13, and R14 because FW is not allowed to store more than 10 rules (since AC$_{FW}$ = 10 rules for FRA in Table 8). This indicates correct operation of authorization function of the Eirene system for FRA mode. For RBRA and RBRAOP, the configuration datastore capacity allocated to FW app is 15 configuration rules and 30 configuration rules respectively (Table 8). Therefore, all 14 configuration rules (i.e., R1 – R14) are stored in the configuration datastore when using the modes RBRA and RBRAOP (indicated by RBRA and RBRAOP rows of Table 9).

Test 3 – Threat 3 Mitigation. The Eirene framework keeps SDN apps accountable for the datastore resources that they utilize. To

verify that the accountability function of the Eirene framework is working as intended, refer to Table 8. The TE app is allowed to store 9 configuration rules in the RBRA mode of operation. Note that R15 is a duplicate of R1 (Eq. 14), and FW app has a lower App-ID than TE app (Table 8) so, the conflict is resolved by not storing R15 (as indicated by row RBRA in Table 9). Then, rules R16 – R19 are stored and $AC_{TE} = 9 - 4 = 5$ configuration rules. Next, R20 is not stored due to conflict resolution (Eq. 14), R21 – R23 gets stored as they are not conflicting rules, and the TE app has sufficient memory resources. This updates $AC_{TE} = 5 - 3 = 2$ configuration rules. Next, R24 is not stored due to conflict resolution (from Eq. 14, R3 & R24 are conflicting rules). This demonstrates the correct working of the accountability function of the Eirene system.

The LB app has $AC_{LB} = 6$ configuration rules for RBRA mode (Table 8). From RBRA row in Table 9, the rule R25 and R26 are not stored inside the SDN configuration datastore because both are conflicting rules (from Eq. 14). The rules R27 – R30 are stored as these are neither duplicates nor conflicting rules. Thereby, making $AC_{LB} = 6 - 4 = 2$ configuration rules. Then, R31 is not stored as it is a duplicate of R17 (Eq. 14). Next, R32 and R33 are stored, which makes $AC_{LB} = 2 - 2 = 0$ configuration rules. Therefore, R35 and R36 are not stored as LB has used all of the datastore capacity allocated to it in the RBRA mode. The fact that the space allocated to LB app is adjusted with incoming rules and the LB app cannot store any more rules after reaching the allocated storage capacity verifies the correct operation of the accountability function of the Eirene framework.

Test 4 – Threat 4 Mitigation. The conflict handling component of the Eirene framework is responsible for detecting and handling rule conflicts in real-time and mitigate threat 4. The FRA row in Table 9 shows that R15 is not stored as it is a duplicate of R1 (Eq. 14). Since FW has a lower App-ID than TE so, the conflict is resolved by keeping R1 as per *principle 1* (Algorithm 1). To resolve the problem of duplicate rules: R3 & R24 and R17 & R31; again, *principle 1* from Algorithm 1 is used. Next, *principle 1* and *principle 3* from Algorithm 1 are used to resolve the conflicts between R5 & R20, R8 & R25, with the result that R5 and R8 are stored in the datastore, R14 is not stored as $AC_{FW} = 0$ configuration rules after the storage of R10. Since R14 is not in the configuration datastore so, R26 gets stored. The fact that R24 and R31 are not stored in the SDN configuration datastore when using FRA mode shows that real-time conflict detection and resolution function of the Eirene system is working. The same principles from Algorithm 1 apply to the RBRA and RBRAOP modes and details are omitted purely for the purpose of saving space.

The dataset we have used for small scale security evaluation does not include attack (configuration) rules because the Eirene system blocks an SDN app without proper credentials.[2]

8.2 A Single-SDN App Large Scale Security Evaluation

In this subsection, we test the Eirene system for its security on a large scale.

[2]An in-depth analysis on the strength of the authentication scheme is considered out of scope for this work and left as a future exercise.

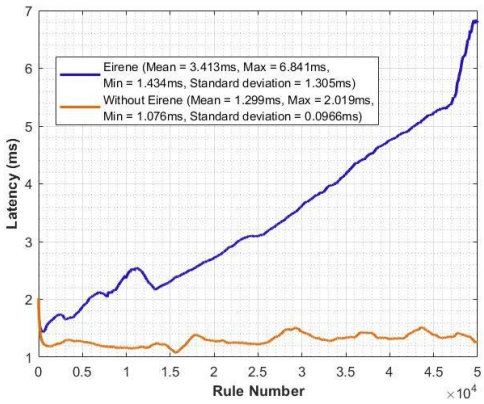

Figure 2: A Single-App Large Scale Security Evaluation.

Real-World Dataset. For large scale security evaluation, we have used real-world configuration rules that belong to the LB app. These real-world (55,000) configuration (attack) rules have been obtained from [30]. Dixit et al. [30] used these rules to carryout Denial of Service (DoS) attack on SDN controller by exhausting its resources.

Experimental Setup. The ODL controller is used for experimentation, its JVM heap size is fixed at 4GB, and datastore capacity is set at $C = 50,000$ configuration rules. The mode of operation used here is RBRAOP, just for simplicity but the experiment can be repeated using FRA and RBRA.

Test 5 – Threat 1 Mitigation. We tested the authentication function by verifying that the LB app is allowed to insert/delete a configuration rule in the SDN configuration datastore only after entering the correct 128-bit encryption key (that has been shared by the network practitioner with the LB app beforehand). This shows the correct working of the authentication function of the Eirene framework.

Test 6 – Threat 2 Mitigation. We tested the working of the authorization function of the Eirene system by verifying the datastore capacity that has been allocated to the LB app. The fact that the LB app is allocated a capacity to store 50,000 configuration rules (where C = 50,000 configuration rules) indicates the correct working of the authorization function of the Eirene framework.

Test 7 – Threat 3 Mitigation. We have verified the accountability function of the Eirene framework by observing that the storage capacity allocated to the LB app correspondingly decreases as more and more rules are added in the SDN configuration datastore. After reaching an upper limit of 50,000 rules, no more rules can be added in the configuration datastore unless some old rules have been deleted. This shows the working of the accountability function of the Eirene system.

Test 8 – Threat 4 Mitigation. The blue curve in Fig. 2 shows linearly increasing one-time latency as rules are added in the configuration datastore. This is because every incoming rule is compared with the existing rule database so that the conflict handling component of the Eirene system can detect and resolve rule conflicts in real-time if need arises. An incoming rule is stored in the datastore iff it is

not a duplicate or a conflicting rule and the corresponding SDN app (to which the rule belongs) has memory resources to fulfill the rule request.

9 DISCUSSION AND LIMITATIONS

In this section, we discuss important aspects, extensions, and limitations of the Eirene framework.

9.1 Performance Overhead (One-time Delay)

The Eirene framework incurs a maximum one-time delay of ≈7ms for the insertion of 50,000th rule in the datastore in the presence of 49,999 rules (Fig. 2). This ≈7ms is not a per-packet but a one-time delay (as once the rule is in the datastore, it gets installed in the switches and then packets are forwarded based on matching packet header). From the prior research on acceptable delay duration (e.g., an acceptable delay duration of ≈300ms (for tapping tasks) and ≈170ms (for dragging tasks) of mobile apps, acceptable ≈13ms delay for realistic gaming environments) and the fact that to date there is no universal limit on acceptable latency for SDN datastores; we conclude that the security achieved by the Eirene framework with a ≈7ms one-time delay is an acceptable trade-off (as the rules in the datastore implement security policies, forwarding decisions, traffic engineering details, etc.; and the correct storage and consequently correct execution of security policies, forwarding rules, etc., is essential for the network to operate in an intended manner).

9.2 Run-time considerations

The Eirene framework may require full or partial adjustments in a few situations while running. While the current Eirene framework prototype does not handle them, we discuss them here.

9.2.1 App-IDs (Unavailable). The Eirene framework uses a sliding window technique to assign App-IDs to SDN apps. The sliding window technique does give the flexibility to cater for new SDN apps with varying levels of priority. If all App-IDs are taken, the network practitioner may need to do some adjustments (such as increasing range of App-IDs, reassigning App-IDs to existing SDN apps, adjusting for corresponding changes in the rule database).

9.2.2 Network Practitioner Parameters (Modified). The Eirene framework requires the network practitioner to set two important parameters: (i) datastore capacity, (ii) number of SDN apps. A change or modification in these parameters leads to corresponding adjustments in the system. For example, assume that the datastore capacity was set at 50,000 configuration rules and the network grew that led to adding external memory resources to increase the datastore capacity (to cater for an additional 20,000 configuration rules). Then, this new datastore capacity of 70,000 configuration rules is to be divided among SDN apps and the ratio of division varies depending on the mode of operation used.

9.2.3 Denial of Service – Possibility. The Eirene system authenticates an SDN app, authorizes it with a role, and then holds it accountable for information stored by it in the configuration datastore. Once an SDN app has used up all its rightful share, the app is denied service by the controller and cannot store any more rules (unless some old rules are deleted by the respective SDN app). This is by design to ensure that one SDN app does not overload the entire configuration datastore.

9.2.4 Permission-based System. The Eirene framework authorizes SDN apps using access control that could be equal (FRA), role based (RBRA), or dynamic (RBRAOP). Each authenticated SDN app can only read, write, modify, or delete its entries and cannot access the data entered by other SDN apps. The structure of permissions can be modified such that some SDN apps can only read while others can read and write.

9.2.5 Credential-based Authentication. The Eirene system requires each SDN app to enter correct credentials (in the form of a 128-bit encryption key) before every rule insertion/deletion. Additionally, periodically changing passwords and correspondingly updating password (128-bit encryption key) database adds a second layer of security against malicious/invalid SDN apps. A detailed discussion on the resilience of the proposed scheme against different types of adversaries is considered out of scope for the purpose of this paper and left as a future exercise.

9.2.6 Scalability. We have evaluated the performance of the Eirene framework using 50,000 configuration rules, which is typically the case for a small to medium sized network. A medium to large scale network may contain rules on the order of hundreds of thousands to millions/billions. We argue that the evaluation of the Eirene framework using 50,000 configuration rules provide a good baseline for any large-scale network using the notion of resource pooling (the details are in Appendix D).

10 CONCLUSION

With this work, we introduce the notion of quantifying security design principles for safeguarding the interactions between SDN apps and the SDN configuration datastore. Based on this quantification, we present a prototype implementation of the Eirene framework. We test the system's security on a small scale and a large scale using two real-world datasets: (i) real-world campus network dataset consisting of real-world complicated cases of configuration rule conflicts, (ii) 50,000+ real-world configuration (attack) rules. We believe that this work, by introducing fundamental definitions for securing interactions between SDN apps and the SDN configuration datastore will serve as a universal baseline security service/guideline to be used across multiple SDN platforms. Thereby, providing a reliable backbone and assisting in reaping the true benefits of the SDN powerhouse.

11 ACKNOWLEDGEMENTS

This work was partially supported from the Center for Cybersecurity and Trusted Foundations at Arizona State University. Sana Habib was supported by the Fulbright scholarship. We would like to thank Jedidiah R. Crandall for inspiring the title of our paper, commenting on our draft, and his guidance on the interpretation of the one-time delay of the system. We are also grateful to the anonymous CCSW'22 reviewers for their valuable feedback.

REFERENCES

[1] 2017. Uauthenticated Upload of Applications. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000081* (2017).

[2] 2018. A comprehensive 3-dimensional security analysis of a controller in software-defined networking. *https://onlinelibrary.wiley.com/doi/epdf/10.1002/spy2.21* (2018).

[3] 2020. Demo AAA/RADIUS Server Testing. *https://wiki.onosproject.org/pages/viewpage.action?pageId=6357336* (2020).

[4] 2020. MarketsandMarkets. *https://www.prnewswire.com/news-releases/software-defined-networking-market-worth-32-7-billion-by-2025–exclusive-report-by-marketsandmarkets-301104606.html* (2020).

[5] 2021. Global SDN Orchestration Market to Reach 117.27 Billion by 2030: Allied Market Research. *https://www.yahoo.com/now/global-sdn-orchestration-market-reach-071500264.html* (2021).

[6] 2021. ONOS. Demo AAA/RADIUS Server Testing. *https://wiki.onosproject.org/pages/viewpage.action?pageId=6357336* (2021).

[7] 2022. Cloud SDN - Ericsson. *https://www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure/cloud-sdn* (2022).

[8] 2022. FloodLight. *https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview* (2022).

[9] 2022. Hewlett Packard Enterprise Development. *https://www.hpe.com/us/en/home.html* (2022).

[10] 2022. Huawei Agile Campus Network Solution Brochure. *https://e.huawei.com/en/related-page/solutions/technical/agile-networking/agile-campus-solutions/agile-campus/brochure/solutions*$_{c}ampus_{n}etwork$(2022).

[11] 2022. Huawei Technologies. *https://www.huawei.com/en/* (2022).

[12] 2022. NEC Corporation. *https://www.nec.com/* (2022).

[13] 2022. Open Network Operating System (ONOS). *https://opennetworking.org/onos/* (2022).

[14] 2022. OpenDayLight. Authentication, Authorization and Accounting (AAA) Services. *https://docs.opendaylight.org/projects/aaa/en/latest/dev-guide.html* (2022).

[15] 2022. OpenDayLight (ODL). *https://www.opendaylight.org/* (2022).

[16] Ihsan H Abdulqadder, Deqing Zou, Israa T Aziz, Bin Yuan, and Weiming Li. 2018. SecSDN-cloud: defeating vulnerable attacks through secure software-defined networks. *IEEE Access* 6 (2018), 8292–8301.

[17] Abdullah Al-Alaj, Ram Krishnan, and Ravi Sandhu. 2019. SDN-RBAC: An access control model for SDN controller applications. In *2019 4th International Conference on Computing, Communications and Security (ICCCS)*. IEEE, 1–8.

[18] Abdullah Al-Alaj, Ram Krishnan, and Ravi Sandhu. 2020. ParaSDN: An Access Control Model for SDN Applications based on Parameterized Roles and Permissions. In *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 107–116.

[19] Abdullah Al-Alaj, Ravi Sandhu, and Ram Krishnan. 2019. A formal access control model for SE-Floodlight controller. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. 1–6.

[20] Abdullah Al-Alaj, Ravi Sandhu, and Ram Krishnan. 2020. A Model for the Administration of Access Control in Software Defined Networking using Custom Permissions. In *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 169–178.

[21] Awais Bin Asif, Muhammad Imran, Nadir Shah, Mehtab Afzal, and Hasnat Khurshid. 2021. ROCA: Auto-resolving overlapping and conflicts in Access Control List policies for Software Defined Networking. *International Journal of Communication Systems* 34, 9 (2021), e4815.

[22] Christian Banse and Sathyanarayanan Rangarajan. 2015. A secure northbound interface for SDN applications. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 834–839.

[23] Durbadal Chattaraj, Basudeb Bera, Ashok Kumar Das, Joel JPC Rodrigues, and Young Ho Park. 2021. Designing Fine-grained Access Control for Software Defined Networks using Private Blockchain. *IEEE Internet of Things Journal* (2021).

[24] Durbadal Chattaraj, Sourav Saha, Basudeb Bera, and Ashok Kumar Das. 2020. On the Design of Blockchain-Based Access Control Scheme for Software Defined Networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 237–242.

[25] Ankur Chowdhary, Dijiang Huang, Gail-Joon Ahn, Myong Kang, Anya Kim, and Alexander Velazquez. 2019. SDNSOC: Object oriented SDN framework. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. 7–12.

[26] Sean Convery. 2004. *Network security architectures*. Cisco Press.

[27] Hongyan Cui, Zunming Chen, Longfei Yu, Kun Xie, and Zongguo Xia. 2017. Authentication mechanism for network applications in SDN environments. In *2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 1–5.

[28] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. 2017. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 122–133.

[29] Spyros Denazis, Evangelos Haleplidis, Jamal Hadi Salim, Odysseas Koufopavlou, David Meyer, and Kostas Pentikousis. 2015. Software-defined networking (SDN): Layers and architecture terminology. (2015).

[30] Vaibhav Hemant Dixit, Adam Doupé, Yan Shoshitaishvili, Ziming Zhao, and Gail-Joon Ahn. 2018. AIM-SDN: Attacking Information Mismanagement in SDN-datastores. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 664–676.

[31] Xiaoyu Duan and Xianbin Wang. 2016. Fast authentication in 5G HetNet through SDN enabled weighted secure-context-information transfer. In *2016 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[32] Liming Fang, Yang Li, Xinyu Yun, Zhenyu Wen, Shouling Ji, Weizhi Meng, Zehong Cao, and Muhammad Tanveer. 2019. THP: A novel authentication scheme to prevent multiple attacks in SDN-based IoT network. *IEEE Internet of Things Journal* 7, 7 (2019), 5745–5759.

[33] Steven R Gomez, Samuel Jero, Richard Skowyra, Jason Martin, Patrick Sullivan, David Bigelow, Zachary Ellenbogen, Bryan C Ward, Hamed Okhravi, and James W Landry. 2019. Controller-oblivious dynamic access control in software-defined networks. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 447–459.

[34] Tao Hu, Zhen Zhang, Peng Yi, Dong Liang, Ziyong Li, Quan Ren, Yuxiang Hu, and Julong Lan. 2021. SEAPP: A secure application management framework based on REST API access control in SDN-enabled cloud environment. *J. Parallel and Distrib. Comput.* 147 (2021), 108–123.

[35] Diego Kreutz, Fernando MV Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2015. Software-defined networking: A comprehensive survey. *Proc. IEEE* 103, 1 (2015), 14–76.

[36] Seungsoo Lee, Seungwon Woo, Jinwoo Kim, Jaehyun Nam, Vinod Yegneswaran, Phillip Porras, and Seungwon Shin. 2022. A Framework for Policy Inconsistency Detection in Software-Defined Networks. *IEEE/ACM Transactions on Networking* (2022).

[37] Seungsoo Lee, Seungwon Woo, Jinwoo Kim, Vinod Yegneswaran, Phillip Porras, and Seungwon Shin. 2020. AudiSDN: Automated detection of network policy inconsistencies in software-defined networks. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1788–1797.

[38] Seungsoo Lee, Changhoon Yoon, and Seungwon Shin. 2016. The smaller, the shrewder: A simple malicious application can kill an entire sdn environment. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. 23–28.

[39] Diogo Menezes Ferrazani Mattos and Otto Carlos Muniz Bandeira Duarte. 2016. AuthFlow: authentication and access control mechanism for software defined networking. *annals of telecommunications* 71, 11 (2016), 607–615.

[40] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[41] Jiseong Noh, Seunghyeon Lee, Jaehyun Park, Seungwon Shin, and Brent Byunghoon Kang. 2016. Vulnerabilities of network OS and mitigation with state-based permission system. *Security and Communication Networks* 9, 13 (2016), 1971–1982.

[42] Yustus Eko Oktian, SangGon Lee, HoonJae Lee, and JunHuy Lam. 2015. Secure your northbound SDN API. In *2015 Seventh International Conference on Ubiquitous and Future Networks*. IEEE, 919–920.

[43] Yustus Eko Oktian, Sang-Gon Lee, and JunHuy Lam. 2018. Oauthkeeper: An authorization framework for software defined network. *Journal of Network and Systems Management* 26, 1 (2018), 147–168.

[44] Hitesh Padekar, Younghee Park, Hongxin Hu, and Sang-Yoon Chang. 2016. Enabling dynamic access control for controller applications in software-defined networks. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*. ACM, 51–61.

[45] Sandeep Pisharody, Janakarajan Natarajan, Ankur Chowdhary, Abdullah Alshalan, and Dijiang Huang. 2017. Brew: A security policy analysis framework for distributed sdn-based cloud environments. *IEEE transactions on dependable and secure computing* 16, 6 (2017), 1011–1025.

[46] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. 2015. Securing the software defined network control layer.. In *NDSS*.

[47] Wei Ren, Yan Sun, Hong Luo, and Mohsen Guizani. 2021. SILedger: A Blockchain and ABE-based Access Control for Applications in SDN-IoT Networks. *IEEE Transactions on Network and Service Management* (2021).

[48] Takayuki Sasaki, Christos Pappas, Taeho Lee, Torsten Hoefler, and Adrian Perrig. 2016. SDNsec: Forwarding accountability for the SDN data plane. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–10.

[49] Nicolas Schnepf, Rĕmi Badonnel, Abdelkader Lahmadi, and Stephan Merz. 2018. Synaptic: A formal checker for SDN-based security policies. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–2.

[50] Philip Shafer, Martin Bjorklund, Robert Wilton, and Kent Watsen. 2018. Network Management Datastore Architecture (NMDA). *Network* (2018).

[51] Bhavesh Toshniwal, Kalpana D Joshi, Pragati Shrivastava, and Kotaro Kataoka. 2019. BEAM: Behavior-based access control mechanism for SDN applications.

In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–2.
[52] Yuchia Tseng, Montida Pattaranantakul, Ruan He, Zonghua Zhang, and Farid Naït-Abdesselam. 2017. Controller DAC: Securing SDN controller with dynamic access control. In *2017 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.
[53] Benjamin E Ujcich, Samuel Jero, Anne Edmundson, Qi Wang, Richard Skowyra, James Landry, Adam Bates, William H Sanders, Cristina Nita-Rotaru, and Hamed Okhravi. 2018. Cross-App Poisoning in Software-Defined Networking. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 648–663.
[54] Haopei Wang, Lei Xu, and Guofei Gu. 2015. Floodguard: A dos attack prevention extension in software-defined networks. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 239–250.
[55] Xitao Wen, Bo Yang, Yan Chen, Chengchen Hu, Yi Wang, Bin Liu, and Xiaolin Chen. 2016. Sdnshield: Reconciliating configurable application permissions for SDN app markets. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 121–132.
[56] Jia-Si Weng, Jian Weng, Yue Zhang, Weiqi Luo, and Weiming Lan. 2018. BENBI: Scalable and dynamic access control on the northbound interface of SDN-based VANET. *IEEE Transactions on Vehicular Technology* 68, 1 (2018), 822–831.
[57] Changhoon Yoon, Seungwon Shin, Phillip Porras, Vinod Yegneswaran, Heedo Kang, Martin Fong, Brian O'Connor, and Thomas Vachuska. 2017. A security-mode for carrier-grade SDN controllers. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 461–473.
[58] Deqing Zou, Yu Lu, Bin Yuan, Haoyu Chen, and Hai Jin. 2018. A fine-grained multi-tenant permission management framework for SDN and nfv. *IEEE Access* 6 (2018), 25562–25572.

A ACTIVE RULE DELETION MANAGER: RULE SLAYER

Active rule deletion manager dynamically evict rules to make room for more important rules, hence the name *"rule slayer."* This module is activated *iff* the following two conditions are met:

(1) The mode of operation is Role-based Resource Allocation as an Optimization Problem (Section 6.1.3).
(2) Datastore can *not* entertain any more storage requests (i.e., $Threshold = 0$).

Let A_I denote an incoming application (having App-ID$_{A_I}$) with a storage requirement R_{Iy}[3] and A_L (having App-ID$_{A_L}$) be an application with lowest *application priority* amongst all applications in datastore. Let R_{Ll}[4] be the rule with lowest priority amongst all rules belonging to A_L. This gives rise to three cases.[5]

A.0.1 Case 1 (App-ID$_{A_I}$ > App-ID$_{A_L}$). This means that A_I has lower priority than A_L. In such a case, the incoming rule (denoted as R_{Iy}) is not stored because it belongs to an application that has lower priority than the lowest priority application inside datastore. Since there is no active rule deletion so, there is no change in the parametric values for A_L. Due to no space, A_I has an unmet storage requirement (i.e., R_{Iy}), which causes a change in the parametric values of A_I as shown by Eq. 15 in Table 10.

A.0.2 Case 2 (App-ID$_{A_I}$ < App-ID$_{A_L}$). This means that A_I has higher priority than A_L. In such a case, the incoming rule (denoted as R_{Iy}) is stored at the expense of R_{Ll}. Consequently, there is a change in the parametric values for both A_I and A_L as shown by Eq. 17 in Table 10.

A.0.3 Case 3 (App-ID$_{A_I}$ = App-ID$_{A_L}$). There are three sub-cases if this condition is met.

[3]Here $y \geqslant 1$ and the subscript I is used to indicate that the rule belongs to A_I.
[4]Here $l \geqslant 1$ and the subscript L is used to show that the rule belongs to A_L.
[5]The parameters σ_i and κ_i are untouched by this box and assumed to be 0. The initial value for $\beta_{A_L} = \beta_{A_I} = \zeta_{A_L} = \zeta_{A_I} = 0$.

(a) Priority of rule R_{Iy} < Priority of rule R_{Ll}.
In this case, the incoming rule (i.e., R_{Iy}) is not stored. Since $A_L = A_I$, so A_L is the notation used in Eq. 16 in Table 10.
(b) Priority of rule R_{Iy} > Priority of rule R_{Ll}.
In such a case, R_{Ll} is slayed to make room for R_{Iy} as R_{Iy} has more priority than R_{Ll}. Note that $A_I = A_L$ so, the notation A_L is used in Eq. 18 in Table 10.
(c) Priority of rule R_{Iy} = Priority of rule R_{Ll}.
In such a case, R_{Iy} is disregarded and computations are done using Eq. 16. Note that the decision to disregard R_{Iy} as it has same priority as R_{Ll} is purely a judgement call (both rules have same priority so, there is no way to decide which one should be kept).

B DUPLICATE AND CONFLICTING ACTION RULE MANAGER: THE JUDGE

Let A_1 (with App-ID$_1$) and A_2 (with App-ID$_2$) be two applications such that App-ID$_1$ < App-ID$_2$. Let R_{11} and R_{21} be two duplicate rules/rules conflicting on action that belong to A_1 and A_2 respectively. Moreover, R_{15} and R_{16} are two duplicate rules/rules conflicting on action that belong to A_1. The parameter η_i (i.e., active rule deletion in order to make space) is untouched by this module and assumed to be 0. Further assume that initially $\beta_1 = \beta_2 = \zeta_1 = \zeta_2 = 0$.

B.0.1 Case 1 (Rule R_{21} arrives after Rule R_{11}). Suppose R_{11} arrives at datastore. This causes a change in the parametric values for A_1, depending on the mode of operation used (i.e., Eq. 4 - Eq. 9). The rule passes through *duplicate and conflicting action rule manager* and is stored in datastore. Next, R_{21} arrives and the parametric values for A_2 are adjusted using Eq. 4 - Eq. 9. When R_{21} enters *duplicate and conflicting action rule manager*, Eirene realizes that R_{21} conflicts with or is a duplicate of R_{11}. Thus, R_{21} is not stored in datastore as App-ID$_1$ < App-ID$_2$. Thus,

$$\sigma_2 = 1 \ (\because R_{21} \text{ cannot be stored}) \ \& \ \kappa_2 = 0 \tag{25}$$

where σ_2 is a conflicting rule that cannot be stored while κ_2 is a conflicting rule that is actively deleted. The corresponding change in the parametric values is given by Eq. 19 and Eq. 20 in Table 11.

B.0.2 Case 2 (Rule R_{11} arrives after Rule R_{21}). Suppose R_{21} arrives at datastore, and gets stored in datastore. Next, R_{11} arrives and is recognized as a duplicate rule or rule that conflicts on action with R_{21}. As App-ID$_1$ < App-ID$_2$ so, R_{21} is actively deleted and R_{11} is stored. Therefore,

$$\sigma_2 = 0 \ \& \ \kappa_2 = 1 \ (\because R_{21} \text{ is actively deleted}) \tag{34}$$

Eq. 21 and Eq. 22 in Table 11 formally represent the changes in parametric values of A_1 and A_2.

B.0.3 Case 3 (Rule R_{16} arrives after Rule R_{15}). Suppose R_{15} arrives at the datastore, and gets stored. Next, R_{16} arrives and is detected as a conflicting on action or duplicate rule. Thus, R_{15} is replaced with R_{16}. Formally,

$$\sigma_1 = 0 \ \& \ \kappa_1 = 1 \ (\because R_{15} \text{ is actively deleted}) \tag{35}$$

Eq. 23 and Eq. 24 in Table 11 represent the changes in parametric values.

Table 10: Active Rule Deletion Manager (*Keys*: App-ID$_i$ → Application Identifier for an SDN app *i*, η_i → number of actively deleted rules, DS_i → Desired Storage capacity, ζ_i → Active Rule Deletion Variable, α_i → Residual Storage Capacity, β_i → Unmet Storage Requirement).

	Mathematical Representation		Mathematical Representation
Case 1 (App-ID$_{A_I}$ > App-ID$_{A_L}$, appendix A.0.1)	$\eta_{A_I} = 0$ (i.e., no active rule deletion) $\zeta_{A_I} = \zeta_{A_I} + \eta_{A_I} = 0 + 0 = 0$ ($\because \eta_{A_I} = 0$, Eq. 8) $Threshold = Threshold + \eta_{A_I} = 0 + 0 = 0$ $\alpha_{A_I} = Threshold - DS_{A_I} = 0 - 1 = -1$ ($\because DS_{A_I} = R_{Iy}$) $Threshold = 0$ ($\because \alpha_{A_I} < 0$, Eq. 5) $\beta_{A_I} = \mu \cdot DS_{A_I} + \beta_{A_I} = 1 + 0 = 1$ ($\because \alpha_{A_I} < 0$, Eq. 6) (15)	Case 3a (Priority of rule R_{Iy} < Priority of rule R_{LI}, appendix A.0.3)	$\eta_{A_L} = 0$ (i.e., no active rule deletion) $\zeta_{A_L} = \zeta_{A_L} + \eta_{A_L} = 0 + 0 = 0$ ($\because \eta_{A_L} = 0$, Eq. 8) $Threshold = Threshold + \eta_{A_L} = 0 + 0 = 0$ $\alpha_{A_L} = Threshold - DS_{A_L} = 0 - 1 = -1$ ($\because DS_{A_L} = R_{Iy}$) $Threshold = 0$ ($\because \alpha_{A_L} < 0$, Eq. 5) $\beta_{A_L} = \mu \cdot DS_{A_L} + \beta_{A_L} = 1 + 0 = 1$ ($\because \alpha_{A_L} < 0$, Eq. 6) (16)
Case 2 (App-ID$_{A_I}$ < App-ID$_{A_L}$, appendix A.0.2)	$\eta_{A_L} = 1$ ($\because R_{LI}$ is actively deleted) $\zeta_{A_L} = \zeta_{A_L} + \eta_{A_L} = 0 + 1 = 1$ ($\because \eta_{A_L} = 1$, Eq. 8) $Threshold = Threshold + \eta_{A_L} = 0 + 1 = 1$ $\alpha_{A_L} = Threshold - DS_{A_L} = 1 - 0 = 1$ ($\because DS_{A_L} = 0$) $Threshold = 1$ ($\because \alpha_{A_L} \geqslant 0$, Eq. 5) $\beta_{A_L} = \mu \cdot DS_{A_L} + \beta_{A_L} = 0 + 0 = 0$ ($\because \alpha_{A_L} \geqslant 0$, Eq. 6) Now, there is space for one more rule. $\alpha_{A_I} = Threshold - DS_{A_I} = 1 - 1 = 0$ ($\because DS_{A_I} = R_{Iy}$) $Threshold = \alpha_{A_I} = 0$ ($\because \alpha_{A_I} \geqslant 0$, Eq. 5) $\beta_{A_I} = 0$ ($\because \alpha_{A_I} \geqslant 0$, Eq. 6) $\eta_{A_I} = 0$ (i.e., no active rule deletion) $\zeta_{A_I} = \zeta_{A_I} + \eta_{A_I} = 0 + 0 = 0$ ($\because \eta_{A_I} = 0$, Eq. 8) $Threshold = Threshold + \eta_{A_I} = 0 + 0 = 0$ (17)	Case 3b (Priority of rule R_{Iy} > Priority of rule R_{LI}, appendix A.0.3)	$\eta_{A_L} = 1$ (i.e., R_{LI} is actively deleted) $\zeta_{A_L} = \zeta_{A_L} + \eta_{A_L} = 0 + 1 = 1$ ($\because \eta_{A_L} = 1$, Eq. 8) $Threshold = Threshold + \eta_{A_L} = 0 + 1 = 1$ $\alpha_{A_L} = Threshold - DS_{A_L} = 1 - 1 = 0$ ($\because DS_{A_L} = R_{Iy}$) $Threshold = 0$ ($\because \alpha_{A_L} \geqslant 0$, Eq. 5) $\beta_{A_L} = \mu \cdot DS_{A_L} + \beta_{A_L} = 0 + 0 = 0$ ($\because \alpha_{A_L} \geqslant 0$, Eq. 6) (18)

Table 11: Duplicate and Conflicting Action Rule Manager (*Keys*: App-ID$_i$ → Application Identifier for an SDN app *i*, AC_i → Allocated Capacity, α_i → Residual Storage Capacity, β_i → Unmet Storage Requirement, ζ_i → Active Rule Deletion, σ_i → Conflicting Rule that cannot be stored, κ_i → Conflicting Rule that has been actively deleted).

	Fair Resource Allocation & Role-based Resource Allocation (Eq. 11)	Role-based Resource Allocation as an Optimization Problem (Eq. 12)
Case 1 (Rule R_{21} arrives after Rule R_{11}, appendix B.0.1)	$\alpha_2 = \alpha_2 + \sigma_2 + \kappa_2 = \alpha_2 + 1 + 0 = \alpha_2 + 1$ (Eq. 25) $AC_2 = \alpha_2 = \alpha_2 + 1$ (assuming $\alpha_2 \geqslant 0$) $\beta_2 = \beta_2 + \sigma_2 = \beta_2 + 1$ ($\because \sigma_2 = 1$) $\zeta_2 = \zeta_2 + \kappa_2 = \zeta_2 + 0 = \zeta_2$ ($\because \kappa_2 = 0$) (19)	$\alpha_2 = Threshold + \sigma_2 + \kappa_2 = Threshold + 1 + 0$ (Eq. 25) $Threshold = \alpha_2 = Threshold + 1$ (assuming $\alpha_2 \geqslant 0$) $\beta_2 = \beta_2 + \sigma_2 = \beta_2 + 1$ ($\because \sigma_2 = 1$) $\zeta_2 = \zeta_2 + \kappa_2 = \zeta_2 + 0 = \zeta_2$ ($\because \kappa_2 = 0$) (20)
Case 2 (Rule R_{11} arrives after Rule R_{21}, appendix B.0.2)	$\alpha_2 = \alpha_2 + \sigma_2 + \kappa_2 = \alpha_2 + 0 + 1 = \alpha_2 + 1$ (Eq. 34) $AC_2 = \alpha_2 = \alpha_2 + 1$ (assuming $\alpha_2 \geqslant 0$) $\beta_2 = \beta_2 + \sigma_2 = \beta_2 + 0 = \beta_2$ ($\because \sigma_2 = 0$) $\zeta_2 = \zeta_2 + \kappa_2 = \zeta_2 + 1$ ($\because \kappa_2 = 1$) (21)	$\alpha_2 = Threshold + \sigma_2 + \kappa_2 = Threshold + 0 + 1$ (Eq. 34) $Threshold = \alpha_2 = Threshold + 1$ (assuming $\alpha_2 \geqslant 0$) $\beta_2 = \beta_2 + \sigma_2 = \beta_2 + 0 = \beta_2$ ($\because \sigma_2 = 0$) $\zeta_2 = \zeta_2 + \kappa_2 = \zeta_2 + 1$ ($\because \kappa_2 = 1$) (22)
Case 3 (Rule R_{16} arrives after Rule R_{15}, appendix B.0.3)	$\alpha_1 = \alpha_1 + \sigma_1 + \kappa_1 = \alpha_1 + 0 + 1 = \alpha_1 + 1$ (Eq. 35) $AC_1 = \alpha_1 = \alpha_1 + 1$ (assuming $\alpha_1 \geqslant 0$) $\beta_1 = \beta_1 + \sigma_1 = \beta_1 + 0 = \beta_1$ ($\because \sigma_1 = 0$) $\zeta_1 = \zeta_1 + \kappa_1 = \zeta_1 + 1$ ($\because \kappa_1 = 1$) (23)	$\alpha_1 = Threshold + \sigma_1 + \kappa_1 = Threshold + 0 + 1$(Eq. 35) $Threshold = \alpha_1 = Threshold + 1$ (assuming $\alpha_1 \geqslant 0$) $\beta_1 = \beta_1 + \sigma_1 = \beta_1 + 0 = \beta_1$ ($\because \sigma_1 = 0$) $\zeta_1 = \zeta_1 + \kappa_1 = \zeta_1 + 1$ ($\because \kappa_1 = 1$) (24)

Table 12: Conflicting Priority and Priority Action Rule Manager (*Keys*: App-ID$_i$ → Application Identifier for an SDN app *i*, AC_i → Allocated Capacity, α_i → Residual Storage Capacity, β_i → Actively Deleted Rules, ζ_i → Conflicting Rule that cannot be Stored, σ_i → Conflicting Rule that has been Actively Deleted).

	Fair Resource Allocation and Role-based Resource Allocation (Eq. 11)	Role-based Resource Allocation as an Optimization Problem (Eq. 12)
Case 1 (Rule R_{41} arrives after Rule R_{31}, appendix C.0.1)	$\alpha_4 = \alpha_4 + \sigma_4 + \kappa_4 = \alpha_2 + 1 + 0 = \alpha_4 + 1$ (Eq. 36) $AC_4 = \alpha_4 = \alpha_4 + 1$ (assuming $\alpha_4 \geqslant 0$) $\beta_4 = \beta_4 + \sigma_4 = \beta_4 + 1$ ($\because \sigma_4 = 1$) $\zeta_4 = \zeta_4 + \kappa_4 = \zeta_4 + 0 = \zeta_4$ ($\because \kappa_4 = 0$) (26)	$\alpha_4 = Threshold + \sigma_4 + \kappa_4 = Threshold + 1 + 0$ (Eq. 36) $Threshold = \alpha_4 = Threshold + 1$ (assuming $\alpha_4 \geqslant 0$) $\beta_4 = \beta_4 + \sigma_4 = \beta_4 + 1$ ($\because \sigma_4 = 1$) $\zeta_4 = \zeta_4 + \kappa_4 = \zeta_4 + 0 = \zeta_4$ ($\because \kappa_4 = 0$) (27)
Case 2 (Rule R_{31} arrives after Rule R_{41}, appendix C.0.2)	$\alpha_4 = \alpha_4 + \sigma_4 + \kappa_4 = \alpha_4 + 0 + 1 = \alpha_4 + 1$ (Eq. 37) $AC_4 = \alpha_4 = \alpha_4 + 1$ (assuming $\alpha_4 \geqslant 0$) $\beta_4 = \beta_4 + \sigma_4 = \beta_4 + 0 = \beta_4$ ($\because \sigma_4 = 0$) $\zeta_4 = \zeta_4 + \kappa_4 = \zeta_4 + 1$ ($\because \kappa_4 = 1$) (28)	$\alpha_4 = Threshold + \sigma_4 + \kappa_4 = Threshold + 0 + 1$ (Eq. 37) $Threshold = \alpha_4 = Threshold + 1$ (assuming $\alpha_4 \geqslant 0$) $\beta_4 = \beta_4 + \sigma_4 = \beta_4 + 0 = \beta_4$ ($\because \sigma_4 = 0$) $\zeta_4 = \zeta_4 + \kappa_4 = \zeta_4 + 1$ ($\because \kappa_4 = 1$) (29)
Case 3 (Rule R_{36} arrives after Rule R_{35}, appendix C.0.4)	$\alpha_3 = \alpha_3 + \sigma_3 + \kappa_3 = \alpha_3 + 1 + 0 = \alpha_3 + 1$ (Eq. 38) $AC_3 = \alpha_3 = \alpha_3 + 1$ (assuming $\alpha_3 \geqslant 0$) $\beta_3 = \beta_3 + \sigma_3 = \beta_3 + 1$ ($\because \sigma_3 = 1$) $\zeta_3 = \zeta_3 + \kappa_3 = \zeta_3 + 0 = \zeta_3$ ($\because \zeta_3 = 0$) (30)	$\alpha_3 = Threshold + \sigma_3 + \kappa_3 = Threshold + 1 + 0$ (Eq. 38) $Threshold = \alpha_3 = Threshold + 1$ (assuming $\alpha_3 \geqslant 0$) $\beta_3 = \beta_3 + \sigma_3 = \beta_3 + 1$ ($\because \sigma_3 = 1$) $\zeta_3 = \zeta_3 + \kappa_3 = \zeta_3 + 0 = \zeta_3$ ($\because \kappa_3 = 0$) (31)
Case 4 (Rule R_{35} arrives after Rule R_{36}, appendix C.0.4)	$\alpha_3 = \alpha_3 + \sigma_3 + \kappa_3 = \alpha_3 + 0 + 1 = \alpha_3 + 1$ (Eq. 39) $AC_3 = \alpha_3 = \alpha_3 + 1$ (assuming $\alpha_3 \geqslant 0$) $\beta_3 = \beta_3 + \sigma_3 = \beta_3 + 0 = \beta_3$ ($\because \sigma_3 = 0$) $\zeta_3 = \zeta_3 + \kappa_3 = \zeta_3 + 1$ ($\because \kappa_3 = 1$) (32)	$\alpha_3 = Threshold + \sigma_3 + \kappa_3 = \alpha_3 + 0 + 1 = \alpha_3 + 1$ (Eq. 39) $Threshold = \alpha_3 = Threshold + 1$ (assuming $\alpha_3 \geqslant 0$) $\beta_3 = \beta_3 + \sigma_3 = \beta_3 + 0 = \beta_3$ ($\because \sigma_3 = 0$) $\zeta_3 = \zeta_3 + \kappa_3 = \zeta + 3 + 1$ ($\because \kappa_3 = 1$) (33)

C CONFLICTING PRIORITY AND PRIORITY ACTION RULE MANAGER: THE PEACEMAKER

Let A_3 (with App-ID$_3$) and A_4 (with App-ID$_4$) be two applications such that App-ID$_3$ < App-ID$_4$. Let R_{31} and R_{41} be two rules conflicting only on priority/both priority and action belonging to A_3

and A_4 respectively. Let R_{35} and R_{36} be two rules that conflict only on priority/both on priority and action, and belong to A_3. Further assume that R_{35} has a higher priority than R_{36}.

C.0.1 Case 1 (Rule R_{41} arrives after Rule R_{31}). Suppose R_{31} arrives at the datastore. The parametric values for A_3 are modified according to the respective equations (i.e., Eq. 4 - Eq. 9). The rule, R_{31}, passes through *conflicting priority and priority action rule manager* and gets stored. Later, R_{41} arrives at the datastore and the parametric values for A_4 are modified. On passing through *conflicting priority and priority action rule manager*, it is discovered that R_{41} conflicts with R_{31}. Thus, R_{41} is discarded because App-ID$_3$ < App-ID$_4$. Thus,

$$\sigma_4 = 1 \ (\because R_{41} \text{ cannot be stored}) \ \& \ \kappa_4 = 0 \tag{36}$$

The corresponding changes in the parametric values is given by Eq. 26 and Eq. 27 in Table 12.

C.0.2 Case 2 (Rule R_{31} arrives after Rule R_{41}). Suppose R_{41} arrives at the datastore, and gets stored. Afterwards, R_{31} arrives and is detected to be in conflict with R_{41}. Thus, R_{41} is deleted from the datastore and R_{31} is stored ($\because$ App-ID$_3$ < App-ID$_4$). Therefore,

$$\sigma_4 = 0 \ \& \ \kappa_4 = 1 \ (\because R_{41} \text{ is actively deleted}) \tag{37}$$

This leads to a change in the parametric values of A_3 and A_4, as given by Eq. 28 and Eq. 29 in Table 12.

C.0.3 Case 3 (Rule R_{36} arrives after Rule R_{35}). Suppose R_{35} arrives at datastore, and gets stored. Afterwards, R_{36} approaches and is marked as a conflicting rule. Since R_{35} has a higher priority than R_{36} so, R_{36} is not stored. Thus,

$$\sigma_3 = 1 \ (\because R_{36} \text{ cannot be stored}) \ \& \ \kappa_3 = 0 \tag{38}$$

The corresponding changes in the parametric values are given by Eq. 30 and Eq. 31 in Table 12.

C.0.4 Case 4 (Rule R_{35} arrives after Rule R_{36}). Suppose R_{36} arrives at the datastore, and gets stored. Afterwards, R_{35} arrives and is identified as a rule that conflicts with R_{36}. Since R_{36} has lower priority than R_{35} so, R_{36} is deleted from the datastore and R_{35} is stored. Formally,

$$\sigma_3 = 0 \ \& \ \kappa_3 = 1 \ (\because R_{36} \text{ is actively deleted}) \tag{39}$$

The corresponding changes in the parametric values is given by Eq. 32 and Eq. 33 in Table 12.

D EIRENE: SCALABILITY

Consider a medium to large scale network with a storage requirement of 100,000 configuration rules. The SDN controllers are horizontally as well as vertically scalable. Thus, the case of 100,000 rules can be catered for by breaking the storage problem into two parts such that there are three separate SDN datastore resources: DR1, DR2, DR3; each with a capacity to store up to 50,000 rules. Assume DR1 and DR2 are storing rules using the Eirene framework (with identical settings). Next, DR3 is used to do comparison of rules stored in DR2 with DR1. Since the worst case comparison is still with 49,999 rules so the one-time worst case delay stays at ≈7ms.

The delay can be further reduced if the resources DR1, DR2, and DR3 are subdivided; such that DR1 is divided into DR11 & DR12; DR2 is divided into DR21 & DR22; DR3 is divided into DR31 & DR32. Each one of the sub resources is capable of storing up to 25,000 rules. Using Fig. 2, the one-time delay for the insertion of 25,000$^{\text{th}}$ rule in the presence of 24,999 rules is under 4ms. Hence, DR11 and DR12 store 25,000 rules each. A comparison is done between the rules stored in DR11 and DR12 with the resulting rules stored in DR31. Now, DR11 and DR31 collectively contain 50,000 unique rules (with no duplicates or conflicts) while DR12 is empty (as the rules from DR12 after comparison with DR11 rule database got stored in DR31). Next, DR21 and DR22 store 25,000 rules, each using the Eirene framework. The rule database of DR21 is compared with DR22 with the resulting rules stored in DR32. This way, DR32 and DR21 collectively contain 50,000 rules while DR22 is empty. We refer to the collective 50,000 rule storage capacity of DR12 & DR31 as DR1231, DR21 & DR32 as DR2132, and DR12 & DR22 as DR1222. The resulting rules after comparison between DR1231 and DR2132 are stored using the collective storage capacity of DR1222. This way, unique 100,000 rules are collectively stored inside DR1222 and DR1231 while DR2132 is empty.

E RELATED WORK: ASSESSMENT METHODOLOGY

We have used the following security assessment methodology for notable academic research to mark whether the mitigation strategy for a threat is *covered*, *partly covered*, or *not covered*.

The mitigation strategy for threat 1 in Table 4 is marked as *covered* if authentication of SDN apps has been proposed, marked as *partly covered* if the authentication scheme is proposed for some SDN component other than SDN apps but can be extended to the authentication of SDN apps, marked as *not covered* if the issue has not been addressed.

The mitigation strategy for threat 2 in Table 4 is marked as *covered* if some authorization scheme for SDN apps is proposed while being mindful of varying SDN app traffic and consequently varying network traffic, marked as *partly covered* if only authorization strategy for SDN apps has been proposed, and *not covered* if the problem is not addressed.

The mitigation strategy for threat 3 in Table 4 is marked as *covered* if the exact details of updating datastore capacity associated with each SDN app is explained, *partly covered* if exact details are missing or the accountability mechanism is proposed for maintaining logs of API calls, etc.; it is marked as *not covered* if the problem has not been addressed.

The mitigation strategy for threat 4 in Table 4 is marked as *covered* if real-time conflict detection and resolution inside the SDN configuration datastore has been covered, marked as *partly addressed* if rule conflict resolution in switch's TCAM table and the conflict resolution strategy can be extended to resolving conflicts inside the SDN configuration datastore, and marked as *not covered* if the problem has not been addressed.

Refer to Secure Northbound Interface [22] in Table 4. Secure Northbound Interface [22] presents a trust model that only allows authenticated SDN apps to use datastore resources thus, the mitigation scheme for threat 1 is marked as covered. Secure Northbound Interface [22] proposes a permission based access control model for authorization and accountability. However, the authorization scheme does not cover the aspects of varying application traffic

and varying network traffic while the accountability scheme has some missing details on how datastore capacity is updated with every rule insertion/deletion. Thus, threat 2 and threat 3 mitigation scheme is marked as partly covered in Secure Northbound Interface [22] row of Table 4. Finally, a mitigation scheme to resolve the problem of information inconsistency by using some form of real-time detection and resolution of configuration rules conflicts has not been covered. Thus, the mitigation strategy for threat 4 is marked as not covered.