

A STATE-BASED PROACTIVE APPROACH TO  
NETWORK ISOLATION VERIFICATION IN CLOUDS

GAGANDEEP SINGH CHAWLA

A THESIS  
IN  
THE DEPARTMENT  
OF  
CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF MASTER OF APPLIED SCIENCE  
IN INFORMATION SYSTEMS SECURITY AT  
CONCORDIA UNIVERSITY  
MONTRÉAL, QUÉBEC, CANADA

AUGUST 2019

© GAGANDEEP SINGH CHAWLA, 2019

CONCORDIA UNIVERSITY  
School of Graduate Studies

This is to certify that the thesis prepared

By: **Gagandeep Singh Chawla**

Entitled: **A State-Based Proactive Approach to Network Isolation  
Verification in Clouds**

and submitted in partial fulfillment of the requirements for the degree of

**Master of Applied Science (Information Systems Security)**

complies with the regulations of this University and meets the accepted standards  
with respect to originality and quality.

Signed by the final examining committee:

Dr. Arash Mohammadi \_\_\_\_\_ Chair

Dr. Amr Youssef \_\_\_\_\_ Examiner

Dr. Ali Akgunduz \_\_\_\_\_ External Examiner

Dr. Lingyu Wang \_\_\_\_\_ Supervisor

Approved \_\_\_\_\_

Chair of Department or Graduate Program Director

\_\_\_\_\_ 2019 \_\_\_\_\_

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

# Abstract

## A State-Based Proactive Approach to Network Isolation Verification in Clouds

Gagandeep Singh Chawla

The multi-tenancy nature of public clouds usually leads to cloud tenants' concerns over network isolation around their virtual resources. Verifying network isolation in clouds faces unique challenges. The sheer size of virtual infrastructures paired with the self-serviced nature of clouds means the verification will likely have a high complexity and yet its results may become obsolete in seconds. Moreover, the fine-grained and distributed network access control (e.g., per-VM security group rules) typical to virtual cloud infrastructures means the verification must examine not only the events but also the current state of the infrastructures. In this thesis, we propose *VMGuard*, a state-based proactive approach for efficiently verifying large-scale virtual infrastructures against network isolation policies. Informally, our key idea is to proactively trigger the verification based on predicted events and their simulated impact upon the current state, such that we can have the best of both worlds, i.e., the efficiency of a proactive approach and the effectiveness of state-based verification. We implement and evaluate VMGuard based on OpenStack, and our experiments with both real and synthetic data demonstrate the performance and efficiency.

# Acknowledgments

At the very beginning, I would like to thank my adviser Dr. Lingyu Wang. His continuous availability and guidance helped me the most to finish this thesis work. I am grateful to him for introducing me to the research world and feeding me the basics of this world. He always listens to my problems patiently and tries his best to rescue me with his inspiring speeches. Moreover, he gave his attention to almost all of my requests, which shows his great kindness. I feel very lucky to have him as my supervisor.

I also like to thank all other faculty members of the CIISE department. I really enjoyed and learned a lot from the courses that I attended during my master's program. I would like to show my gratitude to Mengyuan Zhang and Suryadipta Majumdar for their insightful advice, along with my all fellow labmates for their enthusiastic discussions.

At the end, I would like to acknowledge the unconditional affection and continuous support of my parents and sister in my life. They always fulfill all my silly demands and keep faith on my ability. They have always been the best inspiration in my life.



# Contents

<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivating Example . . . . .	3
1.2 Contributions . . . . .	4
1.3 Outline . . . . .	5
<b>2 Preliminaries</b>	<b>6</b>
2.1 State-Based Verification . . . . .	6
2.2 Proactive Verification . . . . .	7
2.3 Threat Model . . . . .	8
<b>3 Methodology</b>	<b>10</b>
3.1 Challenges . . . . .	10
3.2 Overview . . . . .	12
3.3 Proactive Verification . . . . .	14
3.3.1 Pre-Computation . . . . .	14
3.3.2 Proactive verification . . . . .	15
3.4 Incremental Verification . . . . .	16

3.4.1	Fork . . . . .	16
3.4.2	Merge . . . . .	16
3.5	Pruning . . . . .	18
<b>4</b>	<b>Implementation and Experiments</b>	<b>20</b>
4.1	Implementation . . . . .	20
4.1.1	Initialization Phase . . . . .	21
4.1.2	Run-Time Phase . . . . .	22
4.1.3	Audit Phase . . . . .	24
4.2	Experiments . . . . .	26
4.2.1	Experimental Settings . . . . .	26
4.2.2	Performance of VMGuard . . . . .	27
<b>5</b>	<b>Discussions and Application to NFV</b>	<b>36</b>
5.1	Discussions . . . . .	36
5.1.1	The Effect of a Wrong Prediction . . . . .	36
5.1.2	Choice of Threshold Values . . . . .	37
5.1.3	Correctness of VMGuard Inputs . . . . .	37
5.1.4	Policies Supported by VMGuard . . . . .	39
5.1.5	Cross-Platform Portability . . . . .	39
5.2	Application to NFV . . . . .	41
5.2.1	Introduction . . . . .	41
5.2.2	Preliminaries . . . . .	44
5.2.3	NFV Network Isolation with VMGuard . . . . .	51
5.2.4	Implementation . . . . .	53
<b>6</b>	<b>Related Work</b>	<b>55</b>
6.1	Network Isolation Verification . . . . .	55

6.2	Proactive Verification . . . . .	57
6.3	State-Based Proactive Verification . . . . .	59
<b>7</b>	<b>Other Contributions</b>	<b>62</b>
7.1	ProSAS: Proactive Security Auditing System for Clouds through Caching and Pre-Computation . . . . .	62
7.2	ProxiMet: Security Metrics for Evaluating and Mitigating Co-residency Threats in Public Cloud . . . . .	63
7.3	TenantGuard+: Efficient Cloud Network Policy Verification at Run- time using Formal Methods . . . . .	64
7.4	Modeling NFV Deployment to Identify the Cross-level Inconsistency Vulnerabilities . . . . .	65
7.5	ERDC: Ericsson Research Demo Cloud . . . . .	66
<b>8</b>	<b>Future Work and Conclusion</b>	<b>75</b>
8.1	Future Work . . . . .	75
8.2	Conclusion . . . . .	75
	<b>Bibliography</b>	<b>86</b>

# List of Figures

1	The motivating example . . . . .	3
2	Examples of the state-based verification by TenantGuard [69] . . . . .	7
3	Examples of the proactive verification by LeaPS [41] . . . . .	8
4	Challenges in designing a state-based proactive verification approach	11
5	An overview of VMGuard . . . . .	12
6	Applying VMGuard on the running example . . . . .	13
7	The high-level architecture of VMGuard . . . . .	21
8	Flowchart of initialization phase . . . . .	22
9	Flowchart of thread execution flow . . . . .	24
10	Flowchart of thread execution flow . . . . .	26
11	The data collection time for VMGuard and TenantGuard in the initialization phase . . . . .	28
12	The initialization time for VMGuard and TenantGuard in the initialization phase . . . . .	29
13	The performance of $VMGuard_0$ (incremental) and $VMGuard_1$ for VM level events . . . . .	30
14	The performance of $VMGuard_0$ (incremental) and $VMGuard_1$ subnet level events . . . . .	31
15	The performance of $VMGuard_0$ (incremental) and $VMGuard_1$ for router level events . . . . .	32

16	The performance for <i>VMGuard<sub>2</sub></i> and <i>VMGuard<sub>1</sub></i> vs. Time . . . . .	32
17	The performance for <i>VMGuard<sub>2</sub></i> and <i>VMGuard<sub>1</sub></i> vs. Memory (up) and CPU (bottom) . . . . .	33
18	Performance comparison by varying the # of VMs per subnet . . . . .	34
19	Performance comparison by varying the # of subnet per router . . . . .	35
20	Performance comparison by varying the # of policies per tenant . . . . .	35
21	The NFV motivating example . . . . .	43
22	ETSI NFV architecture . . . . .	45
23	Virtualization vs Containerization . . . . .	46
24	VNFD Sample TOSCA Descriptor . . . . .	49
25	Applying Network Isolation Policies with VMGuard . . . . .	51
26	VMGuard with Audit Ready Cloud tools in ERDC . . . . .	54
27	ARC tools integration reference architecture . . . . .	67
28	Cloud management using OpenStack Horizon and Audit Ready Cloud dashboard covering complete cloud environment . . . . .	69
29	Layer 2 Auditing using Isotop [38] . . . . .	70
30	Layer 3 Auditing using TenantGuard [69] . . . . .	71
31	Proactive Auditing using LeaPS [41] . . . . .	72
32	Security Metrics for Co-residency Evaluation using ProxiMet . . . . .	73
33	Log Anonymization using SegPriv . . . . .	74

# List of Tables

1	Examples of reachability-related events, their parameters to be instantiated, and the number of possible instantiations . . . . .	14
2	Statistics of the datasets . . . . .	27
3	Virtual infrastructure model portability under different cloud platforms [69] . . . . .	40
4	An excerpt of mapping operation application interfaces of different cloud platforms [41] . . . . .	40
5	Comparing network isolation solutions with VMGuard . . . . .	57
6	Comparing proactive solutions with VMGuard . . . . .	59
7	Comparing existing solutions with VMGuard . . . . .	61

# Chapter 1

## Introduction

Security and privacy issues, such as the lack of transparency, accountability and auditability, remain as the main concerns for individuals and enterprises to fully integrate their workflow into clouds [1, 16, 74]. In particular, the multi-tenancy nature of public clouds means cloud tenants would likely keep worrying about the lack of sufficient network isolation around their virtual resources. Recent studies show that almost 70% of cloud users consider security as a major issue in clouds, of which 80% agree that network isolation is the biggest obstacle to adopting clouds [2, 63]. Cloud providers often have an obligation to provide clear evidences for sufficient network isolation [4], either as part of the service level agreements, or to demonstrate compliance with security standards (e.g., ISO 27002/27017 [26, 27] and CCM 3.0.1 [15]). Moreover, cloud providers may gain a competitive edge in today's highly competitive market by providing the capability of verifying network isolation as a security service to their tenants.

However, in contrast to traditional network environments, the virtual infrastructures hosted in clouds pose unique challenges to network isolation verification.

- First, the sheer size of cloud means different network configurations may be constantly introduced by multiple tenants which can easily lead to an isolation breach. For example, a decent size cloud contains a few hundred tenants running thousands of virtual machines deployed in a cloud data center [55]. The tools verifying isolation within traditional networks [24, 23, 73, 22] cannot work with virtual infrastructures. Tools designed for clouds, such as NOD [36], takes hours to detect a breach within a data center. Contrary to which, TenantGuard [69] generates reachability for a data center within seconds. Yet compliance verification to detect an isolation breach is off-line process in [69].
- Second, the self-service nature of a cloud multiplexes the operational complexity with the number of tenants in an environment where any tenant triggered operation may invalidate existing verification results. The dynamical nature of cloud calls for proactive solutions such as LeaPS, PVSC [41, 40], that pre-computes for a critical event in advance for minimum delay to verification at run-time. But these approaches use signatures based prevention which makes them limited only to the known vulnerabilities in cloud platform.

The sheer size of such virtual infrastructures paired with their self-serviced and highly dynamic nature means most verification techniques designed for traditional networks would cause so much delays that their results may become obsolete before they are ready (a more detailed review of related work will be given in Chapter 6). To that end, TenantGuard [69] and LeaPS [41] are two promising solutions specifically designed for clouds, as demonstrated in the following.



## 1.1 Motivating Example

Figure 1 employs a sequence of administrative events (upper-left) inside a simplified virtual infrastructure (upper-right) to demonstrate the limitations of existing works and our key ideas (lower). Suppose a user Bob wants to forbid all the ingress traffic from subnet  $SN_{A1}$  to his billing server (IP: 10.1.1.7) by deleting its security group rule [52]. However, he is not aware of an OpenStack vulnerability OSSA-2015-021 [51], which causes the change of security group to fail silently on already running VMs. At a later time, another user Alice creates her own VM (10.1.5.9) and connects it to subnet  $SN_{A1}$ . At this time, the network isolation has been compromised since Alice can now access Bob’s billing server via subnet  $SN_{A1}$ .

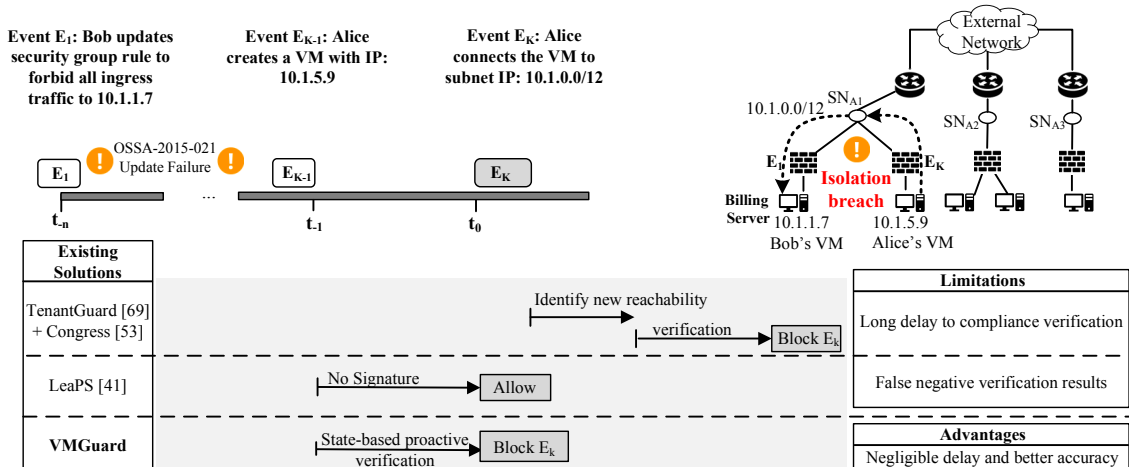


Figure 1: The motivating example

- In such a case, TenantGuard [69] could generate all-pair reachability results within seconds by examining the state of the virtual infrastructure. However, TenantGuard by itself does not support verification against isolation policies (e.g., no ingress traffic from subnet  $SN_{A1}$  to Bob’s billing server), and applying an additional verification tool on top of the all-pair reachability results may again introduce a significant delay (e.g., Congress [53] may take minutes to

hours for large clouds based on our evaluation).

- As a promising solution, LeaPS [41] would predict event  $E_K$  as soon as it sees  $E_{K-1}$  (since the next thing to do after creating a VM is typically attaching it to a subnet), and consequently start the verification for  $E_K$  long before it actually arrives. However, since we assume the vulnerability is not known, the first halve of this undesired reachability (from Bob’s billing server to  $SN_{A1}$ ) can only be detected by examining the state of the virtual infrastructure. Therefore, without looking at the state, the event-based verification in LeaPS [41] will consider  $E_K$  as normal and miss the isolation breach.

This example shows that the state-based approach and the proactive approach both have their advantages and drawbacks. A natural question is: *Can we design a state-based, and proactive approach, such that we can have the best of both worlds?*

## 1.2 Contributions

In this thesis, we present *VMGuard* as an answer to the aforementioned question, as demonstrated at the bottom of Figure 1. We tackle several unique challenges in designing VMGuard. Specifically, the state-based verification only works after an event has actually occurred (with its effect on the state materialized). However, since each event can lead to multiple predicted next events with different effects on the current state, how to verify those predicted events without adversely affecting the true state of the virtual infrastructure (since predicted events may never happen) becomes a major challenge. We will tackle this and other challenges in the remainder of this thesis.

Our main contributions in this work are as follows:

- To the best of our knowledge, VMGuard is the first state-based proactive approach to network isolation verification that excels in both effectiveness and efficiency in comparison to existing methods.
- We implement VMGuard based on OpenStack [52] and evaluate its performance through experiments with both real and synthetic data. Our results confirm the superior performance of VMGuard.

### **1.3 Outline**

The remainder of this thesis is organized as follows. Chapter 2 covers necessary background on both the state-of-art approaches. In Chapter 3, we first illustrate the main challenges for this work and dive into methodology. The implementation and experimentation details of VMGuard are elaborated in Chapter 4. Chapter 5 include discussion over features of VMGuard, its portability to different cloud platforms and application to NFV environment. Chapter 6 compare VMGuard with related works. Chapter 7 briefly introduces other contributions made during period of this thesis. Chapter 8 briefs future directions and concludes the thesis.

# Chapter 2

## Preliminaries

This chapter provides necessary background on two state-of-the-art verification approaches to facilitate further discussions and then defines the threat model.

### 2.1 State-Based Verification

As a state-based (the *state* here refers to the collection of all reachability information inside the virtual infrastructure) verification tool, TenantGuard [69] can efficiently verify all-pair VM-level reachability for large clouds (e.g., 13 seconds for 168 millions of VM pairs) with its hierarchical approach, as demonstrated in Figure 2 and detailed below.

- In [Step 1], TenantGuard checks the subnet-to-subnet reachability within the same network (using the private IPs), e.g., between  $SN_{A1}$  and  $SN_{A2}$ .
- In [Step 2], TenantGuard verifies the subnet-to-subnet reachability involving external networks (using the public IPs), e.g., between  $SN_{A1}$  and  $SN_{A3}$ .
- In [Step 3], TenantGuard only needs to check VM-level reachability (mainly based on security group rules of VMs) for the reachable subnets obtained from

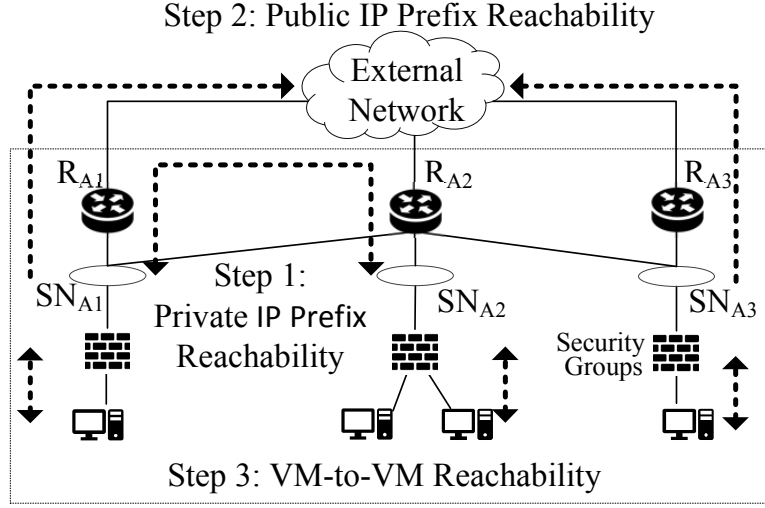


Figure 2: Examples of the state-based verification by TenantGuard [69]

the first two steps, leading to significant cost savings.

We will leverage TenantGuard in a proactive fashion in Chapter 3.

## 2.2 Proactive Verification

As a proactive verification tool, LeaPS [41] initiates the verification before an event actually arrives, leading to millisecond-level response time. This is possible due to the so-called dependency model, which captures the likelihood of next events (either by design, or extracted as frequent patterns from historical events), as demonstrated in Figure 3 and detailed below.

- When a *create VM* event occurs, LeaPS identifies a highly probable next event, *attach port*, from the dependency model.
- LeaPS verifies this predicted event (*attach port*) based on a pre-defined signature of critical events (i.e., events causing isolation breaches, e.g., plugging a port

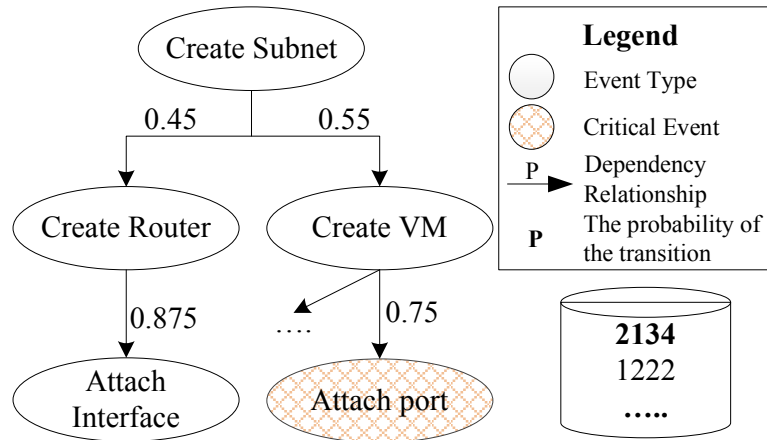


Figure 3: Examples of the proactive verification by LeaPS [41]

on another tenant’s VM under vulnerability OSSA 2014-008 [56]), and if this event causes no breach then its parameters (e.g., VM ID: *2134*) will be added to the watchlist (which is essentially a tenant-specific white list).

- Later, when an *attach port* event actually occurs, all LeaPS needs to do is to search for its parameter (e.g., VM ID) in the watchlist (a match means the event will be allowed), resulting in a negligible delay.

We will enhance LeaPS with state-based verification in Chapter 3.

## 2.3 Threat Model

The in-scope threats include any implementation flaws, misconfigurations, and vulnerabilities in the cloud platform that may be exploited by malicious cloud users to violate the network isolation policies specified by cloud tenants or the provider. Since our solution is based on the system state instead of the events, such threats are in the scope even if they are previously unknown, as long as their effect on network isolation is visible in the state of the virtual infrastructure. On the other hand, we focus on the

virtual network management layer in the cloud and only consider network isolation-related security policies. We also assume the cloud platform may be trusted for the correctness of the inputs (e.g., logs and configuration database). Any security breach that is not reflected in such inputs (either due to the nature of such breaches, such as side-channel attacks, or due to inputs tempered by attackers) is out of the scope, and potential privacy leakage from the verification results is out of the scope.

# Chapter 3

## Methodology

In this chapter, we first show the main challenges and provide an overview before delving into the details of our methodology.

### 3.1 Challenges

We first show the challenges in designing a state-based proactive verification approach, by discussing why the integration between the state-based TenantGuard [69] (see Section 2.1) and the proactive LeaPS [41] (see Section 2.2) will not be straightforward, as illustrated in Figure 4. In [Step 1], upon the occurrence of event  $E_{k-1}$  (*create VM*), we follow the LeaPS approach to predict the next event to be *attach port* based on the dependency model. In [Step 2], we obtain the updated state of the virtual infrastructure based on the effect of this predicted event. In [Step 3], we apply TenantGuard to the updated state to obtain the reachability result. In this case, since there is an isolation breach, the predicted event will not be added to the watchlist (white list), and when the actual event  $E_k$  (*attach port*) arrives it will be denied.



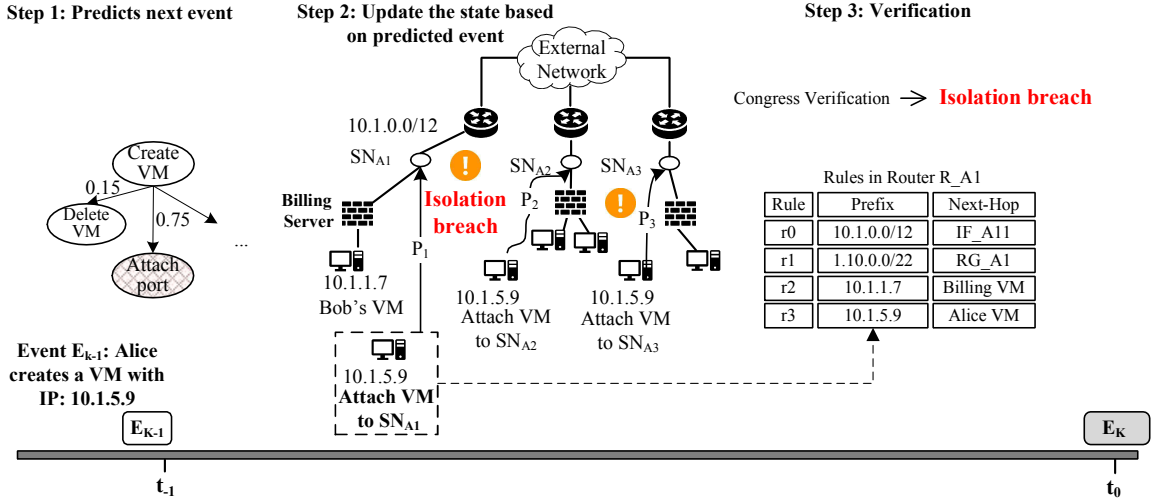


Figure 4: Challenges in designing a state-based proactive verification approach

However, the above description has omitted some major challenges. Specifically, unlike the event-based LeaPS, the state-based TenantGuard only works on the resultant state of an event (i.e., the collection of all the reachability information with the effect of that event taken into consideration). Obtaining such a state for an event that is only predicted, but has not actually occurred, poses a series of challenges as follows.

- First, the dependency model can only predict the type of events (e.g., *attach port*) but not their detailed parameters (e.g., to which subnet) [41]. However, we cannot obtain the resultant state of an event without its parameters.
- Second, even if we could predict the event parameters (e.g., *attach port* to either subnet  $SN_{A1}$ ,  $SN_{A2}$ , or  $SN_{A3}$ ), obtaining the resultant state for those events would still be a challenge. We cannot directly apply such events to the virtual infrastructure, since the predicted events may never occur, while their effects may be irrevocable (e.g., information leakage or denial of service as the result of an isolation breach).

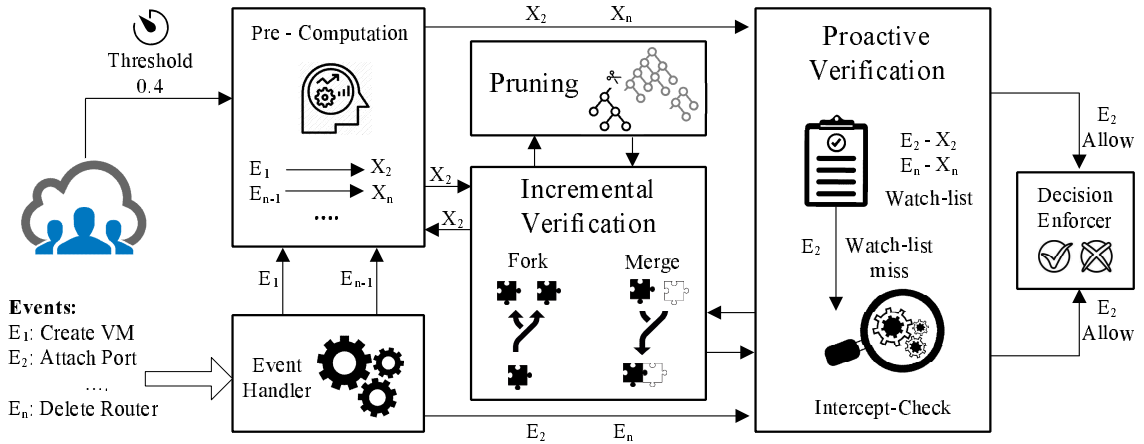


Figure 5: An overview of VMGuard

- Third, while a viable solution is to *emulate* the effect of predicted events on a new copy of the current state, creating such a copy may lead to prohibitive computational and storage overhead in clouds. To make things worse, an event may lead to multiple predicted events with different and incompatible effects on the state (e.g.,  $P_1$  and  $P_3$  lead to breaches but  $P_2$  does not) which requires creating multiple independent copies of the state.

## 3.2 Overview

To address the aforementioned challenges, we design our state-based proactive approach, namely, *VMGuard*. Figure 5 shows an overview of VMGuard with three main modules, proactive verification, incremental verification, and pruning.

- First, the proactive verification module performs pre-computation and proactive verification. In the pre-computation phase, all the predicted next events will be instantiated with potential parameters based on the current state of the virtual infrastructure.
- Second, the incremental verification module forks multiple copies of the state

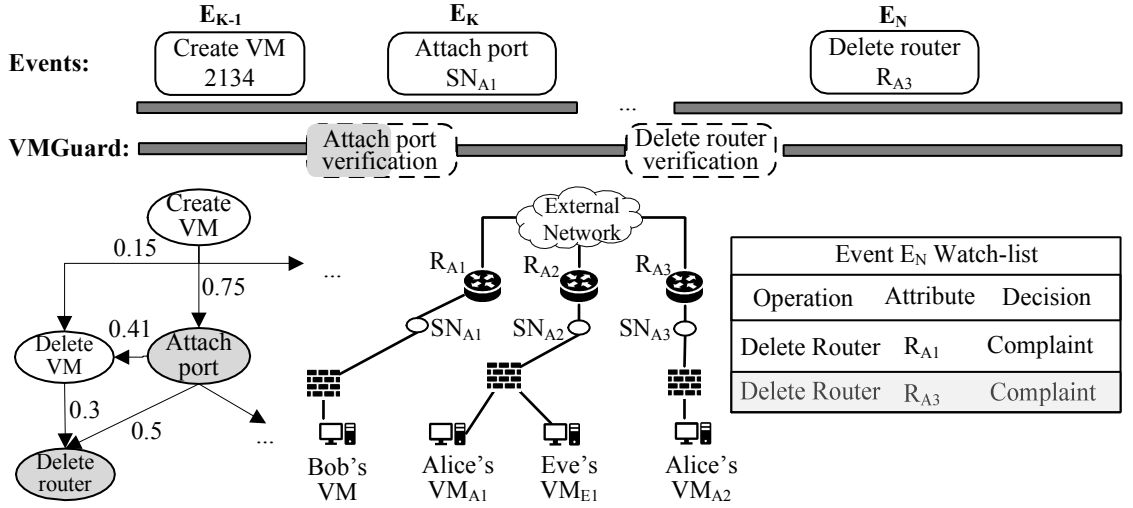


Figure 6: Applying VMGuard on the running example

each of which is used to evaluate the impact of one instantiated event. Each copy of the state is flushed immediately once the verification is done; only the compliant results are stored in a watchlist.

- Third, to minimize the overhead of incremental verification, the pruning module further limits the scope of the verification to the intersection between the set of VMs involved in the policies and the set of VMs that may be impacted by the instantiated event.
- Finally, when the actual event arrives, the proactive verification module matches it against the watchlist and the incremental verification module merges its effect to the actual state of the virtual infrastructure.

## 3.3 Proactive Verification

### 3.3.1 Pre-Computation

Each received event will trigger the pre-computation to identify potential next events based on the dependency model described in Section 2.2. Any event exceeding the pre-defined threshold becomes a candidate for pre-computation. A candidate event gets instantiated based on all possible parameters, as demonstrated through an example based on OpenStack [52] in Table 1. To illustrate this, Figure 6 shows how VMGuard works on our running example; the upper part of the figure is the timeline for incoming events and the actions taken by VMGuard, and the lower part of the figure shows the dependency model (left), the network state (center), and the watchlist for the event  $E_N$  (right).

Event	Parameters	# of Predicted Parameters
Attach Interface	Router_ID and subnet_ID*	#Routers $\times$ #Subnets
Attach Port	VM_ID and subnet_ID	#Subnets
Attach Public IP	VM_ID and unallocated public IP	#VMs $\times$ #unallocated public IP
Delete Router	Router_ID	#Routers
Detach Interface	Router_interfaces_ID	#Router Interfaces
Detach Port	VM port_ID	#Port
Detach Public IP	Allocated public IP	#Allocated public IPs
Detach Security Group Rule	Security_Group_ID	#Security group rules

\* Exception: A router should not have any overlapping subnets

Table 1: Examples of reachability-related events, their parameters to be instantiated, and the number of possible instantiations

#### Example 1

From the sequence of events in Figure 6, event  $E_{K-1}$  creates a VM (*ID: 2134*). According to the dependency model, the next event could be *Delete VM* or *Attach port*. Assume the threshold is 0.5 in all our examples. The pre-computation module selects *Attach port* as the next event to be evaluated in advance. As shown in Table 1, to instantiate this event, both *VM\_ID* and *subnet\_ID* are the required parameters.

The event generator generates three possible events based on the current network state: “*Post create 2134, SN<sub>A1</sub>*”, “*Post create 2134, SN<sub>A2</sub>*”, and “*Post create 2134, SN<sub>A3</sub>*”.

### 3.3.2 Proactive verification

Two possible scenarios may happen for proactive verification, i.e., the actual event may either occur before, or after the pre-computation process finishes. In the first case, VMGuard will switch to the intercept-check mode, which only verifies the intercepted event, e.g., *attach port verification* in Figure 6 (some other situations, e.g., a mistakenly predicted event, will also trigger the intercept-check mode). In the second case, VMGuard will trigger proactive verification which searches for the event in the watchlist, e.g., *delete router verification*. In either case, the watchlist is flushed, as it is no longer valid with the state of cloud. The verification process will be detailed in Section 3.4, and for now it can be regarded as a black-box.

#### Example 2

The next event for *attach port* is “*E<sub>N</sub>: delete router*”. After instantiating the event with the three available routers,  $R_{A1}$ ,  $R_{A2}$ , and  $R_{A3}$ , VMGuard triggers the incremental verification. The results of the verification are added to a watchlist, shown in Figure 6 (lower right). Assume the network isolation policy says “*VM<sub>E1</sub> allows all ingress traffic from the external network*”, which means Eve’s VM under  $R_{A2}$  must be reachable to the external network. Then, “*Delete router R<sub>A2</sub>*” is a non-compliant event and will be removed from the watchlist, whereas “*Delete router R<sub>A3</sub>*” is compliant and will be kept in the watchlist and allowed later when it occurs.

## 3.4 Incremental Verification

As we discussed in Section 3.1, *emulating* the effect of multiple predicted events requires creating copies of the state of the virtual infrastructure. For this purpose, we propose the **Fork** and **Merge** procedures in the *VMGuard* incremental verification module.

### 3.4.1 Fork

During the fork procedure, each instantiated event will be associated with an independent copy of the current state of the virtual infrastructure. To maintain the scalability of this solution, we will leverage the pruning module (detailed in Section 3.5) to limit the scope of verification and we will further evaluate the overhead of the fork procedure through experiments in Chapter 4.2. Once the instantiated event is applied to a copy of the state and the pruning module has been applied, we will employ the resultant state to verify the reachability against the given isolation policy, and then delete that copy of the state as soon as the verification completes. In this way, we do not incur the additional storage overhead for maintaining multiple copies of the state, and only the compliant results will be stored in the watchlist for proactive verification. The actual state of the virtual infrastructure remains intact until the merge procedure is triggered.

### 3.4.2 Merge

The merge procedure will be triggered when a compliant event is actually received. It will update the actual state of the virtual infrastructure based on the effect of this event.

---

**Algorithm 1: INCREMENTAL VERIFICATION**

---

**Input:** *Event*, *RadixTries*, *PrunedList*, *ReachabilityRelatedEvents*  
**Output:**  $result \in \{Compliant, Non-Compliant\}$

- 1 **Copy** *RadixTries* to *RadixTriesFork*
- 2 **if**  $event \in ReachabilityRelatedEvents$  **then**
- 3     **Update** *RadixTriesFork* with *event*
- 4     **for**  $VMdst \in PrunedList$  **do**
- 5         **for**  $VMsrc \in PrunedList$  **do**
- 6              $Trie_{pub} \leftarrow getBTrie(VMdst.publicIP.CIDR, VMsrc.subnet\_id)$
- 7              $Trie_{priv} \leftarrow getBtrie(VMdst.private.CIDR, router\_id)$
- 8              $routable \leftarrow Route-Lookup(Trie_{pub}, Trie_{Priv})$
- 9             **if** *routable is true* **then**
- 10                 **if** *VerifyPolicy(VMsrc, VMdst) is true* **then**
- 11                      $result \leftarrow Compliant$
- 12                 **else**
- 13                      $result \leftarrow Non-Compliant$
- 14                 **break**
- 15 **return** *result*

**Input:** *event\_response*, *Radixtries*

- 16 **if** *event\_response is success* **then**
- 17     **Update** *RadixTries* with *event*

---

In Algorithm 1, lines 10-15 present the fork procedure. It takes an event, the current radix tries (the data structure used in TenantGuard to store the state of the virtual infrastructure [69]), the *prunedList* (this list will be generated in Algorithm 2 in Section 3.5) and *ReachabilityRelatedEvents* (See Table 1 as an example). Lines 1-3 in the algorithm generate a copy of the state for a reachability event and apply the event to the state. Lines 4-5 take the pair of VMs from the prunedlist, then lines 6-8 evaluate the reachability between two VM pairs (*getBtrie*, *getBtrie*, and *Route-lookup* are the VM-level isolation verification functions from TenantGuard). Function *VerifyPolicy* checks whether the reachability between  $VM_{src}$  and  $VM_{dst}$  is allowed. Then lines 9-14 generate compliance results by comparing the policy to the reachability results of TenantGuard. If a non-compliant result is found, this instantiated event is marked as *non-compliant*. Lines 16-17 show the merge procedure

only update the state when the *event\_response* is *success*.

### Example 3

In Figure 6,  $E_{K-1}$  triggers proactive verification for event  $E_K$ . The fork procedure emulates the impacts of three events on three independent copies of the state. However, since the actual event “*attach port SN<sub>A1</sub>*” occurs before the pre-computation phase completes, the verification against other copies gets flushed immediately; the verification happens only on the copy of state with the actual event. TenantGuard will identify the new reachability in the copied state between the newly created VM (*ID: 2134*) and Bob’s VM. Assume the policy is “*VM<sub>Bob</sub>, \*, Deny*” (i.e., denying all ingress traffic to this VM), which means this actual event is non-compliant and therefore, will be blocked by *VMGuard*.

## 3.5 Pruning

To improve the scalability of our incremental verification, we present the pruning module in this section. The main purpose of this module is to reduce the number of VMs for incremental verification. Only the compliant results corresponding to the pruned list of VMs are stored in a watchlist to wait for the actual event.

In Algorithm 2 (pruning procedure), lines 1-3 list the VMs that have at least one policy associated with the input event’s tenant ID. If the list is not empty, lines 4-11 also list the VMs that might be affected by the input event by comparing the attributes of the events, e.g., the type and the parameters of the event. In the end, a pruned list is generated with the intersection of the two VM lists in line 11 and this result will be returned to the incremental verification module.

### Example 4

We illustrate how pruning works for Example 3. We first check the policy “*VM<sub>Bob</sub>, \**,



---

**Algorithm 2: PRUNING**

---

**Input:** *Event, Policies, RadixTries*  
**Output:** *PrunedList*

- 1 **for** *each Policy*  $\in$  *Policies* **do**
- 2     **if** *Policy.TenantID* == *Event.TenantID* **then**
- 3         Add *Policy.VMs* to *VMsUnderTenantPolicy*
- 4 **if** *VMsUnderTenantPolicy* is not empty **then**
- 5     **if** *event* is a *Routing event* **then**
- 6         **for** *each RouterInterface*  $\in$  *RadixTries.RouterInterfaces* **do**
- 7             **for** *each port*  $\in$  *RounterInterface* **do**
- 8                 Add *get(VM)* to *VMsUnderEventScope*
- 9     **else**
- 10         Add *get(VM)* to *VMsUnderEventScope*
- 11     *PrunedList*  $\leftarrow$  *VMsUnderTenantPolicy*  $\cap$  *VMsUnderEventScope*
- 12 **return** *PrunedList*

---

*Deny*” to identify Bob’s VM as the only VM under the policy. Since *attach port* event may affect reachability under the same subnet, which applies to Bob’s VM, the pruning module intersects the two lists and generates a pruned list with only Bob’s VM. As a result, instead of verifying all the eight pairs of reachability, the verification will only need to be performed between Bob’s VM and the newly created VM.

# Chapter 4

## Implementation and Experiments

In this chapter, firstly we detail implementation that distributes the operations of VMGuard into three phases. Secondly we present experimentation, where we first describe experimental settings and then present experiment results with both real and synthetic data.

### 4.1 Implementation

We implement VMGuard based on OpenStack [52], a widely used open source cloud management platform. Figure 7 illustrates the high-level architecture. There are mainly three phases in VMGuard. First, the initialization phase is for pre-processing the given policies and dependency models, which is conducted only once. At initiation, the compliance verification for policies is performed to check that no violation had already occurred before initialization of VMGuard. Second, the run-time phase is for event-driven verification and is conducted with each successive event. Third, the audit phase is to ensure the correctness of prediction and verification. An audit is triggered periodically in background to identify implementation flaws that can introduce errors in the accuracy of verification and improvises tenants' dependency

models as elaborated in Chapter 5.1. Each phase is detailed as follows.

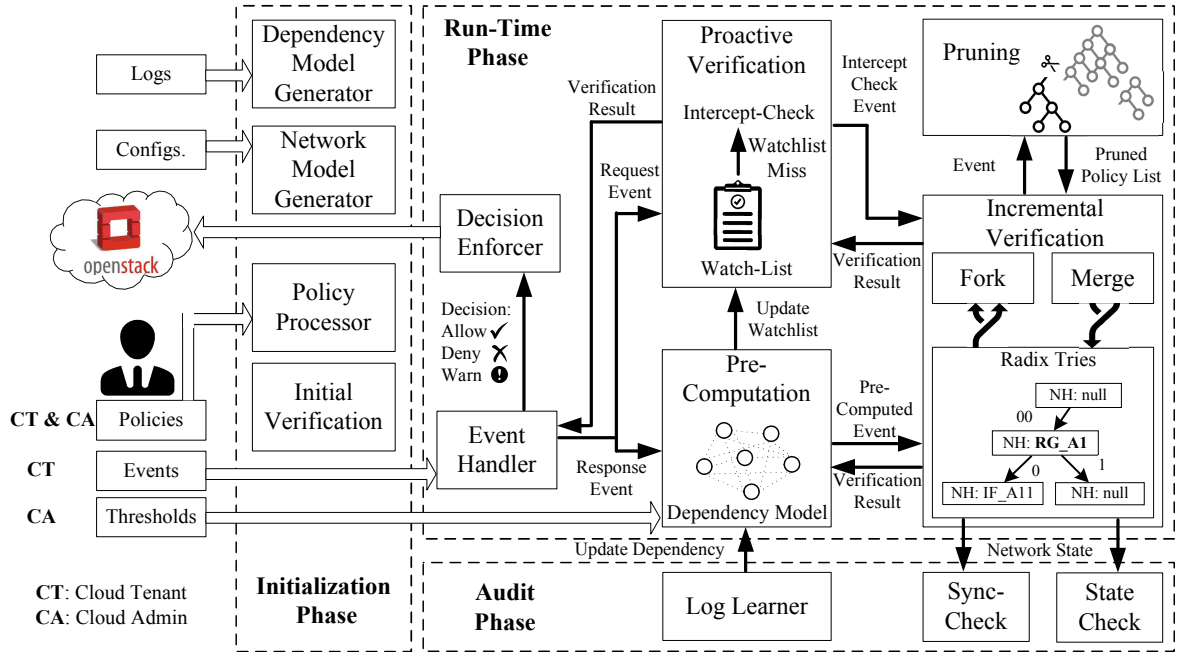


Figure 7: The high-level architecture of VMGuard

### 4.1.1 Initialization Phase

Before initialization, the dependency model is generated for each tenant with the logs collected from two OpenStack services, Nova (compute) and Neutron (network). The raw service logs comprise of all tenants requests which are first parsed into column format and then filtered sequentially for each tenant to obtain tenant’s logged requests that are used to model the tenant’s cloud usage patterns. The processed logs are fed into the Bayesian network tool, SMILE & GeNIe [8] in order to generate the dependency model for each tenant. The configuration information that summarizes virtual network infrastructure resides in Nova and Neutron databases. We fetch these databases and process them to generate the radix tries [28], which facilitates

optimized searching using IP prefixes. The policies stated by both tenants (intra-tenant) and cloud service provider (inter-tenant) are collected from databases and inter-tenant policies are processed into intra-tenant format to aid the policy pruning and verification process, as discussed in Chapter 5.1. Finally, an initial verification is conducted to check the compliance of current cloud state against specified policies. Figure 8 provides an excerpt of initialization phase.

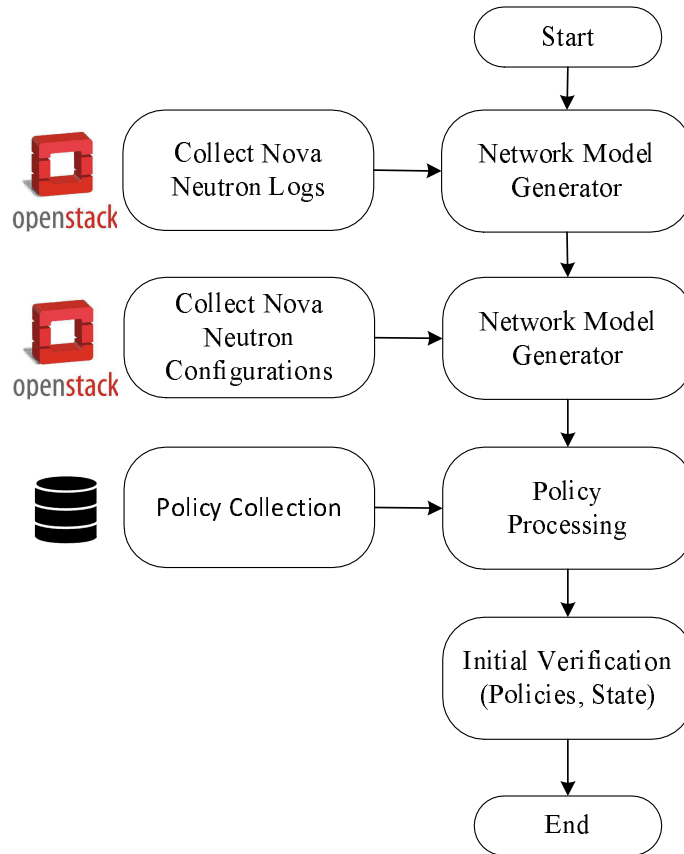


Figure 8: Flowchart of initialization phase

### 4.1.2 Run-Time Phase

The run-time phase is even-driven which is performed with each successive event to ensure continuous compliance. Each Neutron and Nova event is intercepted from the

cloud pipeline using an audit middleware [13]. The middleware redirects the event towards VMGuard that decides the compliance of event before it is actually executed within the cloud pipeline. We leverage the TenantGuard’s [69] implementation, written in Java [59]. Firstly, we add policy-based purging that shortlists policies affected by events based on affected components of virtual infrastructure. We further utilized the deep object copy mechanism [70] that helps creating an identical independent copy of existing virtual infrastructure which can be used to apply events and perform verification without affecting the original virtual infrastructure (referred as fork in incremental verification). Multi-threading [60] is leveraged for parallel verification (i.e., either pre-compute or intercept-check) where multiple events are verified triggered from different tenants. An intercept-check verification for an event occurs at a whitelist miss. Therefore, an intercept-check event requires immediate attention as it leads to direct delays in event processing time. The intercept-check event threads are executed with a higher thread priority that grants an event immediate processing resources. The pre-computation for predicted events need to verify various events with parameters. Therefore, the pre-computation might not complete before an event is triggered by the tenant. So pre-computation threads are executed as background processes with lower thread priority for freeing resources when idle. To aid the management of threads, we initialize them with corresponding tenant ID for whom the verification is being performed. Only one thread will run for each tenant at any moment, i.e., either a pre-computation or an intercept-check thread will exist for a tenant. Figure 9 illustrates the thread work-flow in the run-time phase

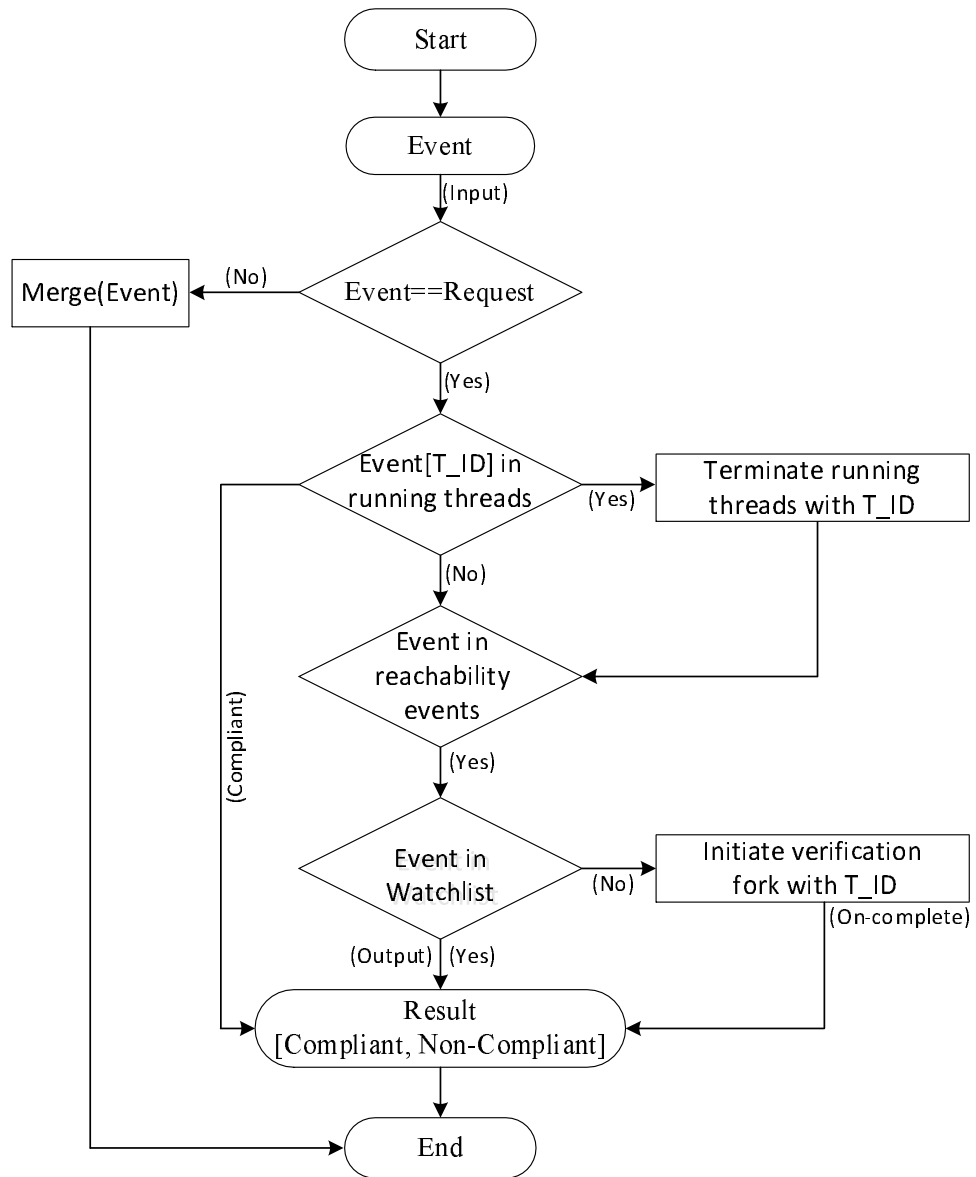


Figure 9: Flowchart of thread execution flow

### 4.1.3 Audit Phase

An audit phase devises implementation and prediction corrections for verification and dependency model. An audit phase is performed periodically as a low priority thread

and ID as *audit* to distinguish it from run-time threads. For identifying implementation flaws a two step audit is performed. Firstly, sync-check verifies synchronization of the virtual infrastructure. State derived incrementally is checked against the actual state of cloud which resides in the service databases. Secondly, state-check verifies policies against virtual infrastructure for their compliance. The interpretation of sync-check and state check outcomes to identify implementation flaws is elaborated in Chapter 5.1. Sync-check is done by graph comparison [68], using JGraphT [29], a Java graph library, to verify nodes, edges and their connections within the states. State-check uses a special incremental verification fork which takes input as both the states and policies. Compliance check is performed and variation in states result implies implementation flaws elaborated in Chapter 5.1. Audit phase also improvises the prediction ability of VMGuard with log learner which learns over simultaneous log trails. It collects Nova and Neutron logs from Ceilometer, the telemetry service in OpenStack. The logs are parsed to column format for GeNIe to update the tenant's dependency model. A few other measures are adopted to enhance event dependencies based on platform learning discussed in Chapter 5.1. Figure 10 illustrates the thread work-flow of audit phase.

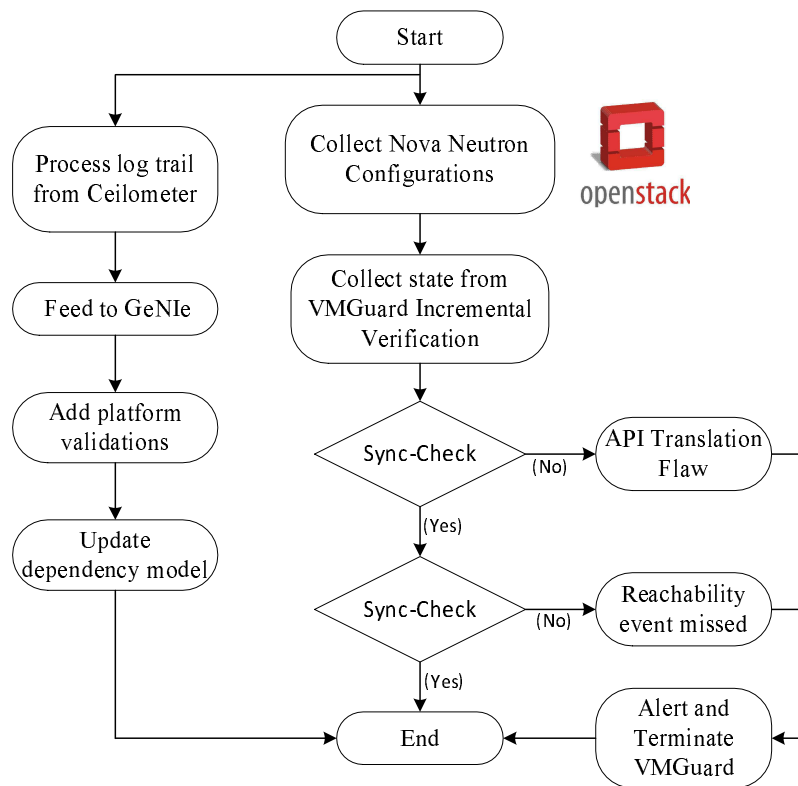


Figure 10: Flowchart of thread execution flow

## 4.2 Experiments

We perform all our experiments based on OpenStack [52], a widely used open source cloud management platform. Firstly, we describe experimental settings and then present experiment results with both real and synthetic data.

### 4.2.1 Experimental Settings

Our test cloud is an OpenStack release Mitaka with one controller node and 80 compute nodes. Each node runs Ubuntu 16.04 server on an Intel i7 dual-core CPU with 2GB memory. The neutron network driver is ML2 OpenVSwitch with L3 agent plugins, which is a popular networking deployment. We use the cloud schema presented



<b>Dataset</b>	<b>Tenants</b>	<b>VMs</b>	<b>Routers</b>	<b>Subnets</b>	<b>Policies per tenant</b>
DS1	50	4,362	300	525	100
DS2	100	10,168	600	1,288	200
DS3	150	14,414	800	1,828	300
DS4	200	20,207	1,000	2,580	400
DS5	250	25,246	1,200	3,210	500

Table 2: Statistics of the datasets

in the recent OpenStack survey [55] as a basis for our simulation where we simulate an environment with maximum 250 tenants and 25,000 VMs. We conduct the experiment on five different datasets varying the number of tenants from 50 to 250, policies per tenant from 100 to 500, subnets from 500 to 3,200, routers from 300 to 1,200, while keeping the number of VMs fixed to 100 per tenant. We believe these data sets represent a wide-range of cloud setups. Every experiment is performed over 100 iterations.

## 4.2.2 Performance of VMGuard

### The Initialization Phase

We first compare the performance of our work with the state-of-art solution, TenantGaurd [69] for the initialization phase. All the experiment results are gathered from a single machine. Parallelization can still be applied to further improve the performance, which is considered as future work. The first set of experiments compare the time consumption for the initialization phase including data collection and initial verification.

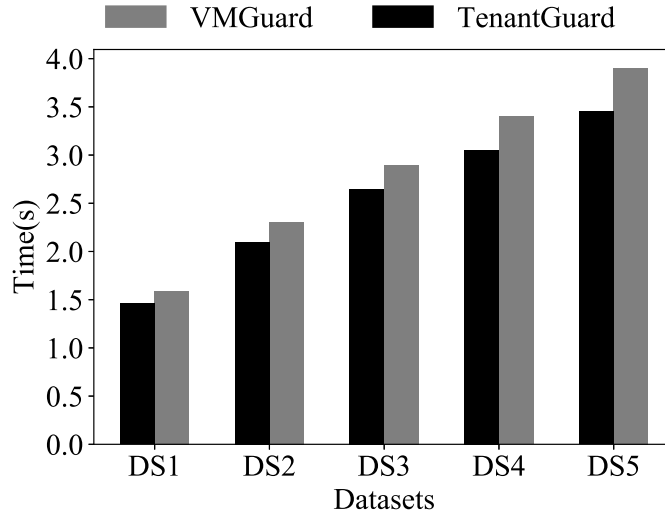


Figure 11: The data collection time for VMGuard and TenantGuard in the initialization phase

### Results and Implications for Initialization

In general, the one-time initialization phase takes longer than other phases. Due to the extra policy processing module, compare to TenantGuard, VMGuard requires a slightly longer time to finish data collection as shown in Figure 11. However, VMGuard performs much better in initialization verification. We observe that TenantGuard’s initialization time increases exponentially with the size of the cloud, whereas VMGuard shows negligible increase largely due to the pruning module which limits the scope of the verification.

### The Incremental Verification and Pruning

TenantGuard is originally designed to work on the static snapshot of the virtual infrastructure [69]. Making TenantGuard incremental and suitable for an event-driven application faces many implementation challenges (some of these are demonstrated in Chapter 3.1). In implementing VMGuard, we have tackled those challenges and, by disabling both the pruning and proactive modules of VMGuard, we basically obtain

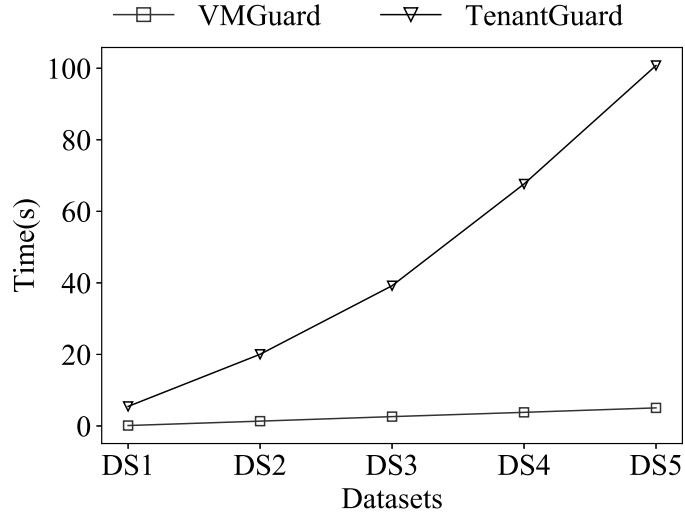


Figure 12: The initialization time for VMGuard and TenantGuard in the initialization phase

an incremental version of TenantGuard, which will be called  $VMGuard_0$  from now on. Also, the VMGuard with only the proactive module disabled (which means it has both the incremental and pruning modules) will be called  $VMGuard_1$ . In the second set of the experiments, we compare  $VMGuard_0$  (incremental) and  $VMGuard_1$  (incremental+pruning) with events selected from different hierarchical levels of the virtual infrastructure. Figure 13, 14 and 15 correspond to VM-level, subnet-level, and router-level events, respectively.

### Results and Implications for Incremental Verification and Pruning

Both  $VMGuard_0$  and  $VMGuard_1$  perform verification within a promising time range (0.4s to verify a high complexity event in the largest dataset) for different types of events. We can observe that  $VMGuard_1$  takes significant less time than  $VMGuard_0$  in all cases, which clearly demonstrates the benefit of the pruning module. When comparing between the three figures, we can see the time consumption for  $VMGuard_0$  and  $VMGuard_1$  is similar in both Figure 13 and Figure 14. This is because, although

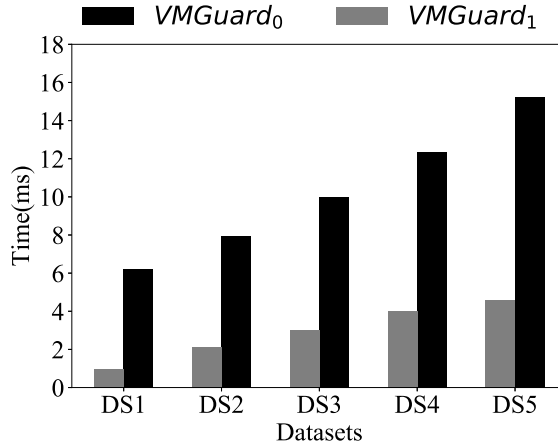


Figure 13: The performance of  $VMGuard_0$  (incremental) and  $VMGuard_1$  for VM level events

attach security group rule and attach port are the events at two different levels, the scope of these two events is similar (i.e., the VMs that directly correspond to the security group rule or the port). Other updating/deletion/addition events at these two levels are expected to share a similar trend. Also, as shown in Figure 15, delete router requires significantly longer verification time, because the verification depends on the number of subnets under the router and the number of VMs under each subnet; the scope of the verification is therefore larger than with the previous two events (even add router event would generate less verification overhead than delete router).

### The Proactive Verification

The third set of the experiments evaluates the performance of the proactive module, namely,  $VMGuard_2$  (the complete version of VMGuard with all modules enabled). Since whether the proactive verification is triggered depends on the given threshold,  $VMGuard_1$  can be considered as a special case of  $VMGuard_2$  by setting the threshold as 1 (means no event can trigger the proactive module). In this experiment, we use real data collected from a real world community cloud hosted at one of the

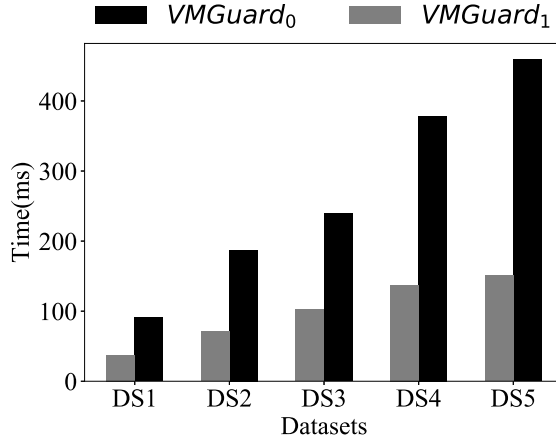


Figure 14: The performance of  $VMGuard_0$  (incremental) and  $VMGuard_1$  subnet level events

largest telecommunications vendors to obtain the dependency model and extract the sequences of events.

## Results and Implications for Proactive Verification

Figure 16 shows the verification time comparison between the intercept-check  $VMGuard_1$  and the proactive  $VMGuard_2$ . The latter one requires significantly less, sometimes negligible, verification time, which demonstrates the benefit of the proactive module. The upper figure of figure 17 shows the run-time memory consumption of  $VMGuard_2$  under two different thresholds, as well as the results of  $VMGuard_1$  and  $VMGuard_0$ . The high memory consumption for  $VMGuard_0$  is mainly due to its need to maintain a large number of pair-wise reachability. After the initialization phase, the memory consumption stays nearly plateau because each successive event will only affect a limited amount of reachability. Comparing  $VMGuard_2$  (under both threshold values) to  $VMGuard_1$ , the number of triggered pre-computations show a positive correlation with memory consumption. During the idle time,  $VMGuard_2^{0.5}$  finishes the pre-computation and shares the same memory consumption as  $VMGuard_1$ ; however, due

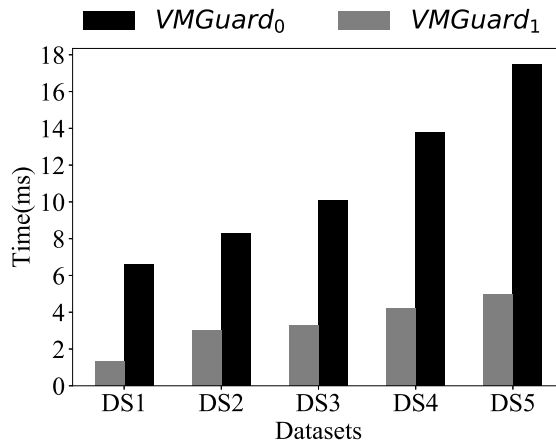


Figure 15: The performance of  $VMGuard_0$  (incremental) and  $VMGuard_1$  for router level events

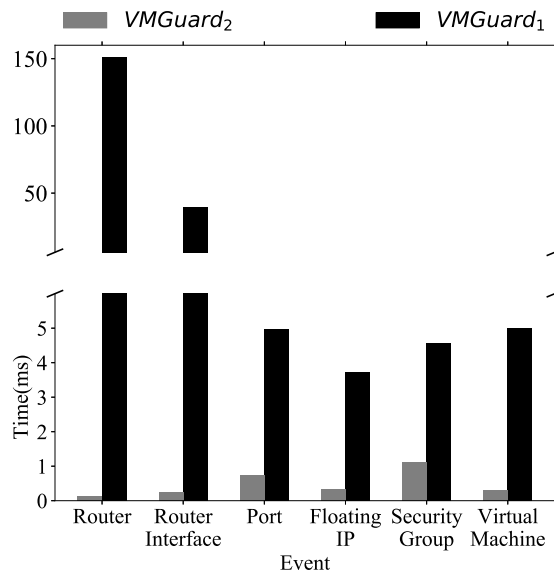


Figure 16: The performance for  $VMGuard_2$  and  $VMGuard_1$  vs. Time

to the larger number of pre-computation candidate events,  $VMGuard_2^{0.1}$  still shows a higher memory consumption during idle time to finish the ongoing pre-computation tasks. The CPU consumption shares almost the same trend as memory consumption.

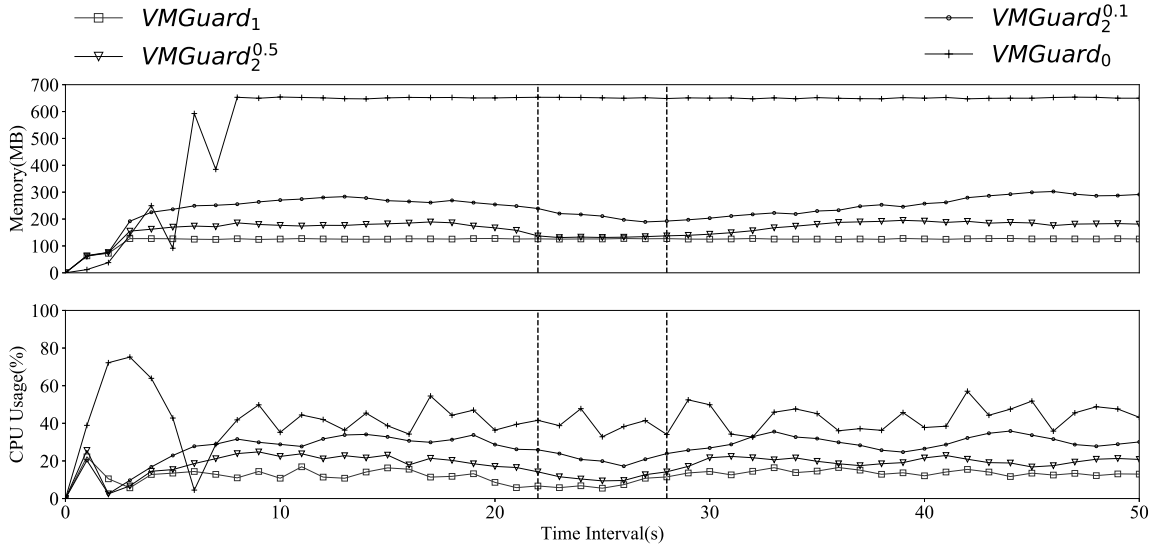


Figure 17: The performance for  $VMGuard_2$  and  $VMGuard_1$  vs. Memory (up) and CPU (bottom)

## Results and Implications for Scalability

As shown in Figure 17, the proactive module significantly reduces the performance time of VMGuard. For evaluation purposes, we disable this module in the scalability tests. Note that this configuration represents the worst case scenario.

Overall, we achieve promising results during scalability with different types of event. In this experiment, we observe that the router event and router interface event require more time during the verification. This is mainly because each event in router or router interface level is associated with a larger number of pruned list VMs due to the impacted VM list is larger than the event happen in a lower hierarchical level. In VMGuard, the intercept check module verifies all the VMs in the pruned list, which means the larger pruned list, the longer verification time; when the pruned list stays the same, the verification time would be constant as well. As we discussed in Section 3.5, the pruned list is directly associated with two lists, the number of VMs under the policy and the number of VMs under the impact of the event. In Figure 18,

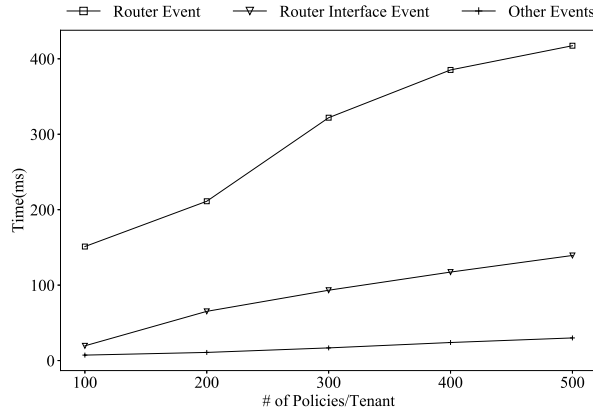


Figure 18: Performance comparison by varying the # of VMs per subnet

the number of VMs under the policy stays the same, however, the impacted number of VMs under router and router interface event increases with the number of VMs per subnet. Therefore, we can observe the increase of time consumption for both types of events. Different than Figure 19, only the impact of router event increases with the number of subnet per router; we observe the increase of router event and a constant trend for the other two events. In Figure 20, the number of VMs under the policy increases since the number of policy per tenant increases. In other words, the number of pruned list increases in all level of events; we observe the increase of time consumption for all the events.



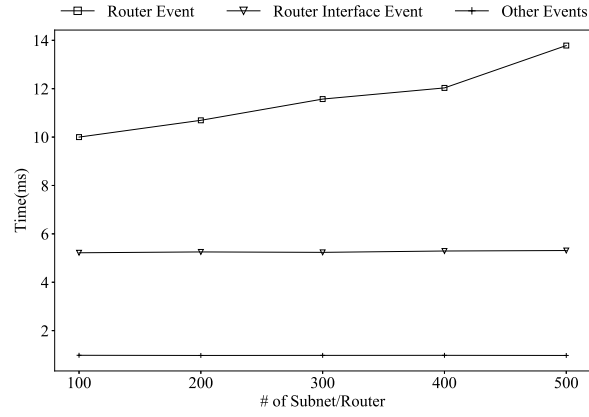


Figure 19: Performance comparison by varying the # of subnet per router

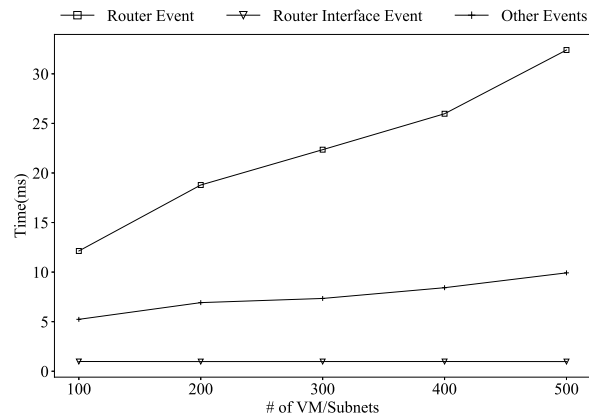


Figure 20: Performance comparison by varying the # of policies per tenant

# Chapter 5

## Discussions and Application to NFV

In this chapter, firstly we discuss effect of wrong prediction, choice of threshold value, correctness of input, policies supported by VMGuard and various cloud platform portability. Secondly, we extend our solution for NFV environment proactively enforcing network isolation within VFN's using state-based verification.

### 5.1 Discussions

#### 5.1.1 The Effect of a Wrong Prediction

We distinguish our dependency models from LeaPS [41] and PVSC [40], as they propose common dependency model for all the tenants in an environment. For our case, the dependency model for each tenant is distinct and represents the cloud operations usage patterns of the tenant. VMGuard pre-computes verification for the predicted event. If there is a wrong prediction resulted from the inaccuracy of tenants' dependency model, the pre-computed results are not useful and hence, VMGuard works

as an intercept-check solution in such circumstances. To continuously improve the prediction accuracy of the dependency model for each tenant, we periodically trigger log learner that updates this model using the tenants cloud logs in the audit phase. Cloud logs stores operations requested by all the tenants which are first segregated for each tenant and then learning is performed to update tenants dependency models. In addition, to be more accurate in the prediction, we incorporate structural dependencies specific to the cloud platform with dependency model that adjusts prediction probability for scenarios that cannot exist for the tenant (For example, In OpenStack [52], a subnet cannot be deleted by the tenant before detaching all the created ports).

### **5.1.2 Choice of Threshold Values**

In VMGuard, the amount of pre-computation effort is controlled through a threshold value. The operations with prediction probability higher than or equal to the threshold value are chosen as precomputation candidates. The verification is performed for each candidate operation with all its possible variants. The lower value of threshold results in a higher number of prediction candidates and vice versa. Therefore, precomputation is simultaneously performed for all the tenants by *VMGuard* which requires a careful estimation by the cloud service provider. The threshold provided by the cloud provider can be decided based on the service level agreement (SLA) with customers as well as based on the experiences of cloud service provider.

### **5.1.3 Correctness of VMGuard Inputs**

The correctness of state-based verification performed by VMGuard relies on two factors. First, the correctness of extracting operation parameters from the cloud management application interface which produces the events. Second, determining the event which affects reachability. The occurrence of any error in the extraction of operation

parameters will result in a deviation of incrementally produced state from the actual state of the cloud. Missing reachability can add a delay in detection (an isolation breach will still be detected but it will be retroactive). To overcome the aforementioned challenges to the correctness of verification, we perform synchronization and state check. These checks are triggered periodically in the audit phase.

- In synchronization check, the incrementally obtained virtual infrastructure (state) is checked against actual configuration (state) of the cloud. Both states (actual state of the cloud and incrementally derived state) are organized as graphs. The graph contains vertices as virtual components (e.g., router, interfaces, ports, subnets, VM, etc.) and the edges connect the components. The vertices and edges are cross-checked within both graphs. The failure in matching vertices and edges implies the existence of an error which occurred during the extraction of operation parameter from the management application interface. Further, the vertex or an edge that does not match with the actual configuration indicates the exact operation which was not correctly extracted.
- The state check is performed in addition to the synchronization check, only if no error is found. In the previous step, we determine that the state obtained incrementally is coherent with the actual cloud configurations. Now we verify the compliance of all the network isolation policies over the state (can be any state as both of them are coherent). A fork thread is assigned with a low priority and the ID *audit* to distinguish the thread from precomputation and intercept-check threads. The input to an audit fork is the state (from the previous step) and all the network isolation policies. The failure of verification implies misidentification of reachability affecting operation. The policy which is violated can be used to trace back to the reachability operation that went misidentified.

### 5.1.4 Policies Supported by VMGuard

VMGuard is designed to verify policies related to virtual network isolation. As an example, VMGuard supports the list of policies proposed in NoD [36]. The list of security policies (defined by the cloud provider and/or its tenants) is an input to the VMGuard system. These policies can vary in nature, such as inter-tenant (i.e., cross-tenant reachability policy) specified only by the cloud service provider, and intra-tenant (i.e., tenant scoped reachability policy) specified by tenants for themselves. The format for specifying policy is as follow:

$$\textit{Intra-Tenant Policy}=\{SRC_{VM},DST_{VM},T_{ID},Protocol,Action\}$$

$$\textit{Inter-Tenant Policy}=\{SRC_{VM},SRC\_T_{ID},DST_{VM},DST\_T_{ID},Protocol,Action\}$$

For the ease of pruning and verification, VMGuard processes an inter-tenant policy into two intra-tenant policies where destination in each is replaced with the external network of the corresponding tenant. Protocol in policy specifies both direction (i.e., ingress and egress), port numbers (i.e., access control list parameters) for the reachability. The action states whether the reachability amongst VMs should exist or not.

### 5.1.5 Cross-Platform Portability

Cloud platforms are developed by various vendors such as Amazon EC2 [3], Google GCP [25], Microsoft Azure [42] and VMware vCD [67] (i.e., the big four). Each vendor has their own application interfaces and virtual infrastructure model. VMGuard leverages the models from both TenantGuard [69] and LeaPS [41] and inherit their interoperability amongst different cloud platforms. In TenantGuard [69], the virtual infrastructure model comprises forwarding and filtering rules which are used with different cloud platforms as per their vendor specification. The probability of a virtual

infrastructure model is illustrated in Table 3. The inter-tenant, intra-tenant, and L3 routing uses different nomenclature for various vendor-specific cloud platforms.

Routing/ Filtering	OpenStack [52]	Amazon EC2-VPC [3]	Google GCE [25]	Microsoft Azure [42]	VMware vCD [67]
Intra-tenant routing	Host routes, routers	Routing tables	Routes	System and user-defined routes	Distributed logical routers
Inter-tenant routing	Routers, external gateways	Internet gateway/VPC peering	Internet gateway	System route to Internet	Edge gateway
L3 filtering	Security groups	Security groups	Firewall rules	Network security groups	Edge firewall service

Table 3: Virtual infrastructure model portability under different cloud platforms [69]

Operation	OpenStack [52]	Amazon EC2-VPC [3]	Google GCE [25]	Microsoft Azure [42]
create VM	POST /servers	aws opsworks --region create-instance	gcloud compute instances create	az vm create l
delete VM	DELETE /servers	aws opsworks --region delete-instance --instance-id	gcloud compute instances delete	az vm delete
update VM	PUT /servers	aws opsworks --region update-instance --instance-id	gcloud compute instances add-tags	az vm update
create security group	POST /v2.0/security-groups	aws ec2 create-security-group	N/A	az network nsg create
delete security group	DELETE/v2.0/security-groups/{security_group_id}	aws ec2 delete-security-group --group-name {name}	N/A	az network nsg delete

Table 4: An excerpt of mapping operation application interfaces of different cloud platforms [41]

Similarly, each vendor-specific cloud platform uses different nomenclature for their management application interfaces. LeaPS [41] presents the analogy in nomenclature used with various cloud platforms in contrast to OpenStack [52] as illustrated in Table 4. Hence, the interoperability of VMGuard among different vendor-specific cloud platforms can be easily derived from the interoperability of its two ancestors, i.e., TenantGuard [69] and LeaPS [41].

## 5.2 Application to NFV

### 5.2.1 Introduction

The telecommunication industry has increased over-subscribed usage of their core networks each successive year as it faces rapid increase in the consumer base with an exploding number of devices that need to communicate (e.g., Internet of things [34]), which increases the demand for data exchange, requiring a higher bandwidth to operate. According to a Cisco forecast, the global mobile data traffic is expected to reach approximately 31 Exabytes per month by 2020, i.e., roughly a seven-time increase since 2017 [12]. To cope up with market demands, the telecoms are often forced to substantially increase both CAPEX (Capital Expenditures) and OPEX (Operating Expenditures) for upgrading their physical network infrastructure. Traditionally, a network service (NS) deployed by telecoms required specialized networking hardware (known as middle-boxes or network appliances) that are vital components to operate core networks [11]. The expansion and upgrade of network services faced enormous challenges such as equipment compatibility, space to accommodate new equipment and manual efforts required to deploy and manage new network services. The aforementioned challenges of the telecom network operators to incorporate network expansions and deliver reliable services with a high bandwidth brings about

a new line of solution with ETSI NFV [18, 19]. With seven of the world's leading telecoms network operators, European Telecommunications Standards Institute (ETSI; The world standardization organization for telecommunications) is the home of industry specification group for Network Function Virtualization (NFV). NFV devises virtualization (Clouds) to decouple hardware from the software for a network functions to be deployed as a Virtual Network Function (VNF) over the Commercial Off-The-Shelf (COTS) servers.

Virtualization moves traditional data centers into the Clouds, making them resource efficient through multi-tenancy that relies on a virtualized pool of resources to reduce computation cost, aid flexibility, and provide metered services. To that end, NFV is an evolution of clouds and therefore shares its attack surface. Yet till date, from Berkeley's view of clouds, the lack of visibility into underlying infrastructure limits auditability in the cloud which makes it as one of the top ten security concerns [7]. Therefore, there is a need for auditing NFV to enhance the guarantee of stakeholders with transparency and accountability as illustrated in the motivating example below:

**Motivating Example:** Recall the attack scenario presented in Chapter 1 , consider that Alice and Bob are now considered as tenants with an NFV service provider, who is running the orchestration platform by leasing infrastructure resources from a cloud provider as a cloud tenant.



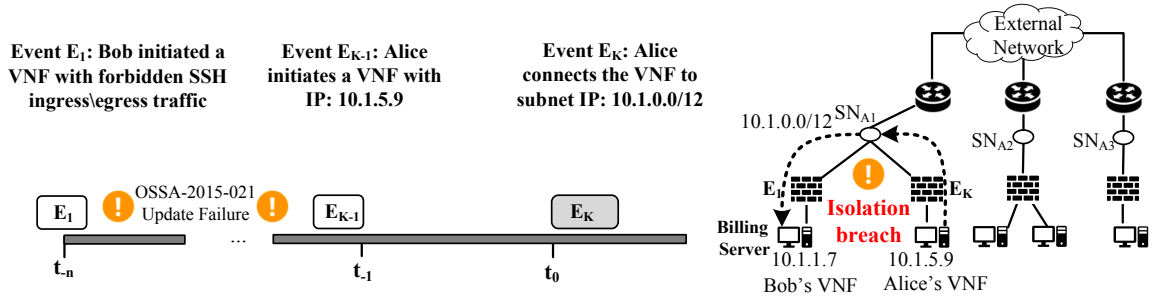


Figure 21: The NFV motivating example

Figure 21 illustrates a series of events triggered by the NFV clients Alice and Bob. With event  $E_1$ , Bob initiated a VNF having the security group which forbids SSH Ingres and Egress traffic to the VNF. Unaware of an existing vulnerability in the underlying VIM ( i.e. Cloud; discussed in a section later), Bob found himself to be secure. Alice initiated a VNF with event  $E_{K-1}$  and update the VNF to attach with the management network (subnet) of Bob, with event  $E_K$ . This created an isolation breach between Alice and Bob, which made Bob's VNF reachable to Alice's VNF. To that end, we can observe that the NFV environment resides on clouds and therefore shares a common attack surface which makes it subject to network isolation breaches. Hence, we extent our cloud-based solution VMGuard to target an NFV environment to enhance network isolation guarantee delivered by NFV. VMGuard extends its application to proactively enforcing network isolation between VNFs using state-based verification.

## 5.2.2 Preliminaries

This section reviews NFV reference architecture with a focus on the role of cloud in NFV. Later we review the interaction between cloud and NFV to orchestrate and manage network services. Finally, we present an enhanced threat model associated with the application to NFV.

### NFV Reference Architecture

Figure 22 illustrates the high-level reference architecture defined by ETSI as the guidelines to NFV. [19]. Also, known as MANO (management and orchestration architecture); it has three major components identified as the driving unit for NFV, which aid automation, orchestration and management of the network services. The three components are detailed as follows.

- **Network Function Virtualization Orchestrator (NFVO)**

NFVO acts as a management interface, where the client specifies their network service requirements. Commonly known as OSS/BSS (Business/Operations Support System), it supports end-to-end telecommunication services requirement gathering which is encoded as the descriptors [71]. A descriptor can be coded with different specification standards, out of which the most widely accepted descriptor standard is TOSCA [44] [20]. The templates are on-boarded and stored as catalogs. A client can choose an already existing catalog or can onboard a customized network services descriptor based on their specific requirements. NFVO orchestrates VNFs on the underlying cloud infrastructure and manages them by coordinating with VNFMs.

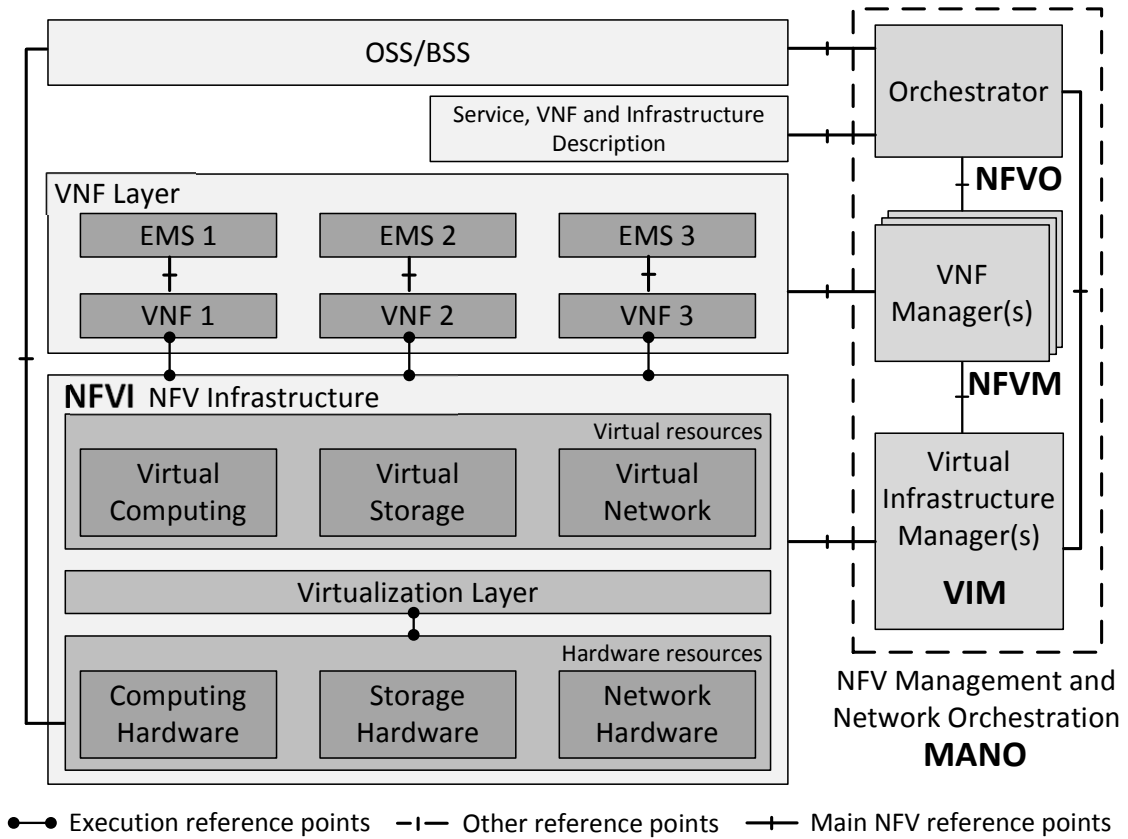


Figure 22: ETSI NFV architecture

– **Virtual Network Function Manager (VNFM)**

A VNFM manages the life-cycle for the VNFs. A VNF may exhibit operations such as query, scale, heal, subscribe, terminate, notify, etc. as a part of the life-cycle, which are common to all the VNFs (known as health monitoring). VNFM also coordinates with the EMS (Element Management Systems). An EMS is specific life-cycle manager for a VNF, which requires mission-critical health operations. An EMS is provided by vendors to assist health-checks from their VNFs requirements [21].

– **Virtual Infrastructure Manager (VIM)**

VIM manages and controls Network Function Virtualization Infrastructure (NFVI) i.e., the pool of virtualized compute, storage and network resources. The physical servers are combined in order to act as a common pool of resources, which are then logically redistributed. The resource pool can be driven either in presence of hypervisor, i.e. Virtualization, or in absence of it, i.e. Containerization. Virtualization runs guest OS on the host OS which is completely isolated. Containerization runs guest containers that are packaged container specific utilities and uses the common utility from host OS, which makes containers partially isolated from host. [17]. Figure 23 illustrates the architectural view of virtualization and containerization. The open source platforms for virtualization based VIM is OpenStack [52] and containerization based VIM is Kubernetes [14]. Note that, VMGuard is designed to work with the cloud environment, therefore, In our use-case, we only cover virtualization based VIM i.e., OpenStack.

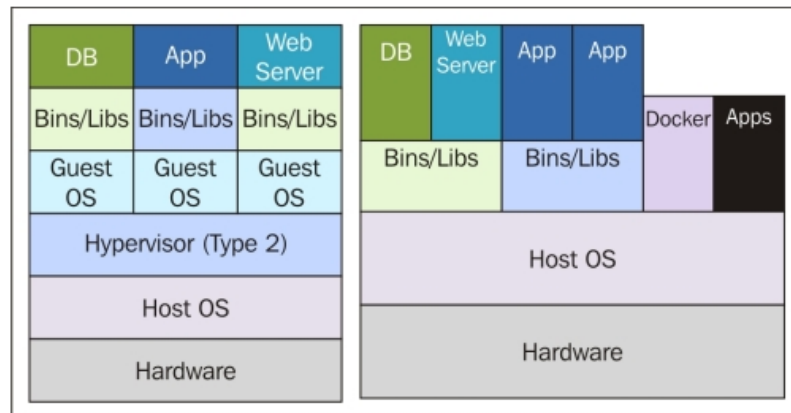


Figure 23: Virtualization vs Containerization

## Deploying a Network Service

### TOSCA

OASIS standardizes descriptors as the means to encode interoperable services and applications for enterprise workloads hosted on the cloud [43]. TOSCA, in particular, enables interoperability and portability with automated management across the cloud regardless of an underlying platform or infrastructure [44]. OASIS drafts descriptor specification for NFV using which a network service can be created, managed and updated (scaling, alarming, downgrading, etc.) [45]. The descriptors are of three types based on the targeted part of a network service.

#### - **Virtual Network Function Descriptor (VNFD)**

VNFD describes the specification of a VNF. A VNF comprises of virtual deployment units (VDUs) as the main component. A VDU corresponds to the virtual machine and its specifications for running VNF software on it. The connection points (CP) corresponds to the ports that attach VDUs to the network. The virtual links (VL) corresponds to the routing rules connecting a network to the router. The floating IPs (FIP) corresponds to the public IPs that connect VDUs to the Internet. Figure 24(left) excerpt a sample VNF descriptor.

#### - **VNF Forwarding Graph Descriptor (VNFFGD)**

VNFFGD describes the forwarding paths which connects the VNFs. The forwarding graphs require neutron-SFC (service function chaining) [57], which we

do not support with current VMGuard. It can be included in the future with upgrades in the verification algorithm.

- **Network Service Descriptor (NSD)**

NSD describes the complete end-to-end network service which is composed by connecting VNFs and the forwarding graphs.

tosca_definitions_version: tosca_nfv_1_0_0	TOSCA version
description: Sample VNFD template	VNF description
metadata: template_name: sample-tosca-vnfd-template-guide topology_template: node_templates:	
VDU: type: tosca.nodes.nfv.VDU.Tacker capabilities: nfv_compute: properties: mem_page_size: [small, large, any, custom] cpu_allocation: cpu_affinity: [shared, dedicated] thread_allocation: [avoid, separate, isolate, prefer] socket_count: any integer core_count: any integer thread_count: any integer	VDU specification
properties: image: Image to be used in VDU flavor: Nova supported flavors availability_zone: available availability zone mem_size: in MB disk_size: in GB num_cpus: any integer metadata: entry_schema: config_drive: [true, false] monitoring_policy: name: [ping, noop, http-ping] parameters: monitoring_delay: delay time count: any integer interval: time to wait between monitoring timeout: monitoring timeout time actions: [failure: respawn, failure: terminate, failure: log] retry: Number of retries port: specific port number if any config: Configure VDU as per requirements mgmt_driver: [default=noop] service_type: type of network service to be done by VDU user_data: custom commands to be executed on VDU user_data_format: format of the commands key_name: user key artifacts: VNFIimage: type: tosca.artifacts.Deployment.Image.VM file: file to be used for image	
CP: type: tosca.nodes.nfv.CP.Tacker properties: management: [true, false] anti_spoofing_protection: [true, false] type: [sriov, vnic] order: order of CP within a VDU security_groups: list of security groups requirements: - virtualLink: node: VL to link to - virtualBinding: node: VDU to bind to	Ports and Security Groups specification
VL: type: tosca.nodes.nfv.VL properties: network_name: name of network to attach to vendor: Tacker	Routing Rules with attached network
heat_template_version:	HOT specification version
description: Sample HOT	Template description
parameters:	Template Parameters
key_name: type: string description: Name of keypair to assign to servers image: type: string description: Name of image to use for servers flavor: type: string description: Flavor to use for servers public_net: type: string description: > ID/name of public network for floating IP will be allocated	
... Create, modify and delete virtual resources such as VMs, routers, routing interfaces, subnets, ports, security groups, floating IPs, etc.	
resources: private_net: type: OS::Neutron::Net properties: name: { get_param: private_net_name }	
private_subnet: type: OS::Neutron::Subnet properties: network_id: { get_resource: private_net } cidr: { get_param: private_net_cidr } gateway_ip: { get_param: private_net_gateway } allocation_pools: - start: { get_param: private_net_pool_start } end: { get_param: private_net_pool_end }	
router: type: OS::Neutron::Router properties: external_gateway_info: network: { get_param: public_net }	
router_interface: type: OS::Neutron::RouterInterface properties: router_id: { get_resource: router } subnet_id: { get_resource: private_subnet }	
server1: type: OS::Nova::Server properties: name: Server1 image: { get_param: image } flavor: { get_param: flavor } key_name: { get_param: key_name } networks: - port: { get_resource: server1_port }	
server1_port: type: OS::Neutron::Port properties: network_id: { get_resource: private_net } fixed_ips: - subnet_id: { get_resource: private_subnet }	
server1_floating_ip: type: OS::Neutron::FloatingIP properties: floating_network: { get_param: public_net } port_id: { get_resource: server1_port }	

Figure 24: VNFD Sample TOSCA Descriptor

## **HOT**

Heat [47] is an OpenStack [52] orchestration engine, which implements a service that enables the tenants to automate launching of multiple composite cloud applications that are written as HOT(Heat Orchestration Template) [48]. The templates are human and machine accessible codes written in YAML which manages the entire life-cycle of infrastructure and applications within the cloud. Figure 24 (right) presents an excerpt of a heat orchestration template.

### **Deploying a network service**

The NFV tenants (Bob and Alice in the motivating example) can either choose from an on-boarded VNFD catalog or can upload a customized VNF descriptor based on their requirements. The NFV provider hosts NFVO that facilitates the platform for NFV clients. An NFV tenant initiates VNF by choosing the corresponding catalog, VIM and NFVO orchestrate the descriptor definitions in a chosen VIM. The VNF descriptors specified using TOSCA specification is provided by the NFV tenant as illustrated in Figure 24. The NFVO parses template with TOSCA-Parser [58], which converts it to a heat orchestration template (HOT). The heat template is provided to OpenStack (VIM) for orchestrating the network service in the cloud. The heat orchestration template defines a workflow for the cloud. The workflow is then executed and NFVO is acknowledged for successful orchestration.

### **Threat Model**

We adopt a thread model targeting threats such as implementation flaws, misconfiguration, and vulnerabilities in the cloud platform which a malicious cloud user may



exploit in-order to violate the network isolation policies specified by cloud tenants or the provider. We assume data (e.g., logs and configuration databases) from cloud platform are intact and any attack prior to the current state of cloud which is not reflected in the input data is out of scope. Any potential privacy leakage from the verification results is out of the scope. Additionally, the change in policy is subjected to an action triggered only by an orchestrator or the cloud provider. An NFV client can only choose from recommended policies and may not have direct interaction to devise their custom policies. Lastly, the scope of verification is strictly restricted to be inside the VIM. Any service that stretches over more than one VIM is subjected to verification within their corresponding VIMs.

### 5.2.3 NFV Network Isolation with VMGuard

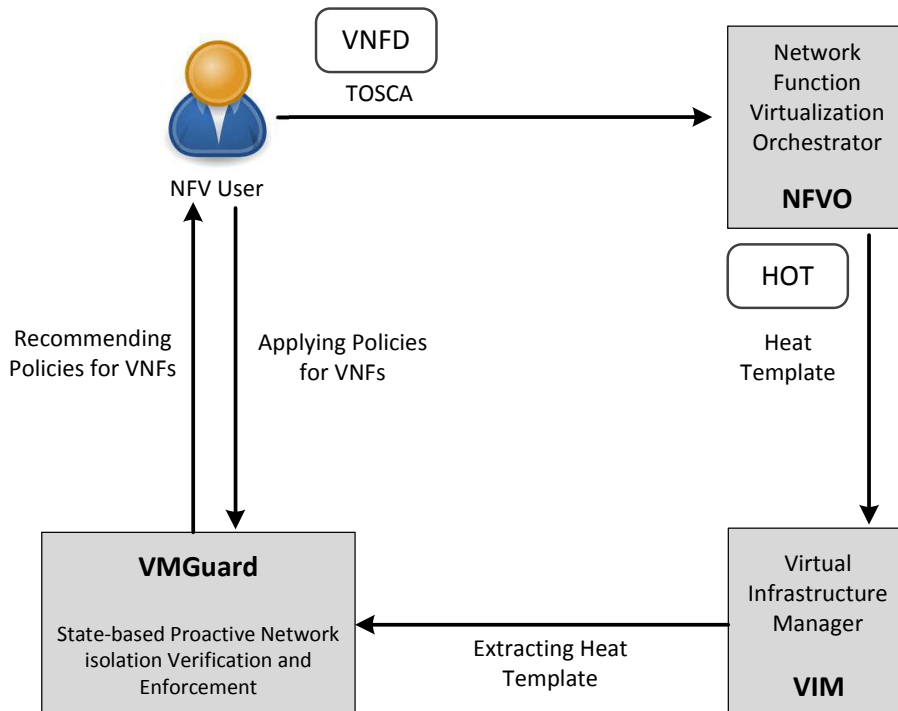


Figure 25: Applying Network Isolation Policies with VMGuard

Figure 25 shows how VMGuard interacts with NFV. The tenants deploy VNFs which are stored as TOSCA and converted to heat template as elaborated before. The HOT is passed to VIM (i.e., OpenStack), which uses heat-parser and heat-engine to deploy the template. VMGuard resides at VIM coordinates with the heat database to extract VNF definition to recommend network isolation policies.

In Algorithm 3, line 1 is to extract resources requested by heat which is currently in YAML format. Lines 2-4, shows queries to collect the requested network resources (virtual machines, ports, security group rules, routing interfaces, and router). A state fork in line 5 and all pair reachability in-scope of tenant is derived in line 6. Reachability is identified in lines 8-11, based on which policies are produced with corresponding actions. Line 12 returns the policies applicable to network isolation within the tenant. The tenant can now present isolation policies to VNF owners to proactively audit their VNFs with state-based verification.

---

**Algorithm 3:** POLICY PROCESSING

---

**Input:** *Heat Orchestration Template*  
**Output:** *Policies*

- 1 Resources = ParseYAML(HOT)
- 2 **for** *each Resource*  $\in$  Resources **do**
- 3     **if** *Resource.Type* == (*Server/Subnet/Port/Router/Interface/Security Group*) **then**
- 4         NetworkResource=Resource;
- 5     Graph = Fork(NetworkResource);
- 6     Reachability = IntraTenantAllPair(Graph);
- 7     **for** *each Result*  $\in$  Reachability **do**
- 8         **if** *Result*==*Reachable* **then**
- 9             Policies=policies(Result,Allow);
- 10         **else**
- 11             Policies=policies(Result,Deny);
- 12 **return** *Policies*

---

## 5.2.4 Implementation

### NFV Test Bed

We extend our testbed further to add NFV platform. With the existing OpenStack environment, we add OpenStack Heat [47] project that executes the workflow. We develop an NFV platform with OpenStack MANO project Tacker [61] and its prerequisites Keystone [54], Barbican [46] and Mistral [50]. Further, we modify policy processing methodology with VMGuard to support automated policy recommendation for the NFV tenants.

### Proof-Of-Concept

We integrate VMGuard with ERDC (Audit Ready Cloud platform), elaborated in section 7.5. VMGuard addresses the limitations of existing proactive and retroactive solutions with state-based proactive verification and enforcement. Figure 26 shows the VMGuard graphical interface. In Figure 26 (top), we can see the initiation button for VMGuard which performs processes prerequisites such as dependency model for tenants. Once done, it asks for policy specification mode, where the automatic mode is to extract heat policies for NFV users and manual mode for cloud tenants. On receiving the auditing of auditing policies, VMGuard is initiated.

In Figure 26 (bottom), we show alert board for VMGuard which indicates isolation breach violations. Each violation is presented with details such as tenant involved, policy violated, detection mode (i.e., proactive or intercept-check). Further, a violation prevented by VMGuard can be viewed with the state-based effect (reachability) which it would have exerted if the violating event had been executed.

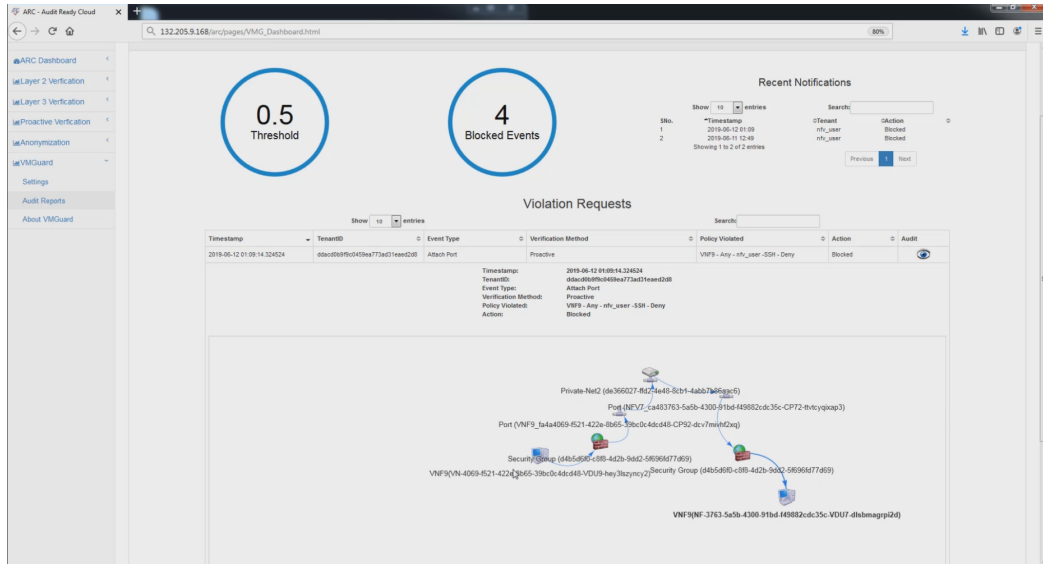
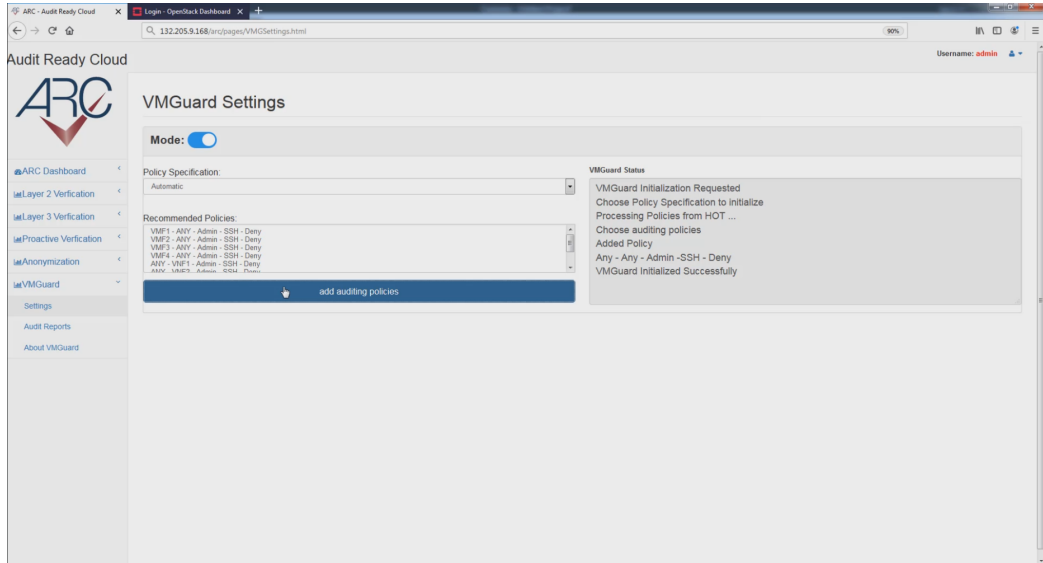


Figure 26: VMGuard with Audit Ready Cloud tools in ERDC

# Chapter 6

## Related Work

VMGuard integrates the best of both state-based verification and proactive approaches. Firstly, we review existing solutions which verify network isolation in the non-cloud or cloud environments. Secondly, we review the solutions with proactive abilities for non-cloud and cloud environments. Finally, we compare the collection of cloud-based solutions with VMGuard to showcase the advantages that our solution possesses.

### 6.1 Network Isolation Verification

Table 5 shows existing works for network isolation (i.e., State-Based) verification solutions.] The first column classifies existing works based on targeted environments, i.e., either cloud (virtual networks) or non-cloud (traditional networks). The second and third columns list existing works and indicate their verification methods, respectively. The next column compares those works based on various features, e.g., the support of parallel implementation, incremental verification, NAT, and all pairs reachability verification. The next two columns respectively compare the scope of those works, i.e., whether the work is designed for physical or virtual networks, and whether it

addresses control or data plane in such networks.

There exist several works (e.g., [32, 31, 39, 33, 72, 73, 24, 22, 23]) that target non-cloud networks (i.e., either traditional network or SDN). Works such as [32, 39, 73, 24, 22, 23] cover physical network. Most of these works face issues in scalability for the cloud environment. Other existing works [31, 33, 72, 9, 65, 38, 69], like VMGuard, covers virtual networks and can scale to the large scale cloud environments. Networking functionality is divided into planes where traffic steering is managed by the control plane and the data plane is responsible for traffic forwarding. Existing works such as [24, 22, 23, 65] focus on the control plane whereas VMGuard, TenantGuard [69] and [32, 31, 39, 33, 72, 73, 36, 64] are developed for the data plane. Based on the features incorporated by existing works, only VMGuard, TenantGuard [69], Libra [73], Anteater [39] and NetPlumer [31] can leverage parallel processing. VMGuard can parallelize state of virtual infrastructure to verify events simultaneously whereas TenantGuard [69] uses parallel processing for in-memory storage. The incremental verification helps to cope with the dynamicity of the network. Existing works such as [31, 72, 73, 33], TenantGuard [69] and VMGuard can operate incrementally. TenantGuard [69] proposes incremental verification but does not evaluate it. VMGuard enhances the proposed methodology by TenantGuard [69] with pruning and evaluates the technique. Network address translation used for public IP in the cloud is only covered in [32, 31, 39, 72, 73, 36, 64], TenantGuard [69] and VMGuard. Existing works such as [31, 23, 36, 64] and TenantGuard [69] target cloud-wide reachability which is different from VMGuard as it performs verification in the scope of network isolation policies.

Platform	Proposal	Methods	Network		Plane		Features			
			Phys.	Virt.	Control	Data	Paral.	Incr.	NAT	All Pairs Reach.
Non-Cloud	Hassel [32]	Custom algorithms	✓			✓			✓	
	NetPlumber [31]	Graph-theoretic		✓		✓	✓	✓	✓	✓
	Anteater [39]	SMT Solver	✓			✓	✓		✓	
	Veriflow [33]	Graph-theoretic		✓		✓		✓		
	AP verifier [72]	Custom algorithms		✓		✓		✓	✓	
	Libra [73]	Graph-theoretic	✓			✓	✓	✓	✓	
	ARC [24]	Graph-theoretic	✓		✓					
	ERA [22]	Custom algorithms	✓		✓					
	Batfish [23]	SMT Solver	✓		✓					✓
Cloud	NoD [36]	SMT Solver	✓			✓			✓	✓
	Plotkin et al. [64]	SMT Solver	✓			✓			✓	✓
	Cloud Radar [9]	Graph-theoretic		✓						
	Probst et al. [65]	Graph-theoretic		✓	✓					
	Madi et al. [38]	CSP Solver		✓		✓				
	TenantGuard [69]	Custom algorithms		✓		✓	✓	✓	✓	✓
	VMGuard	Custom algorithms		✓		✓	✓	✓	✓	

Table 5: Comparing network isolation solutions with VMGuard

## 6.2 Proactive Verification

Table 6 summarizes proactive solutions for cloud and non-cloud environments. The first and second columns enlist existing works and their verification methods. The

next column compares the coverage such as supported environment (cloud or non-cloud) and cloud layers (virtual infrastructure and/or user-level). 'Both' means the work supports both virtual infrastructure and user-level cloud layers. The next six columns compare these works according to different features, i.e., proactiveness, automated and dynamic dependency capturing, cloud-platform-agnostic and probabilistic dependencies.

Solutions such as Jiang et al. [30], Ligatti et al. [35] are for non-cloud environment whereas PVSC [40], Weatherman [10], Congress [53], LeaPS [41], Patron [37] and VMGuard works with cloud environment. Only VMGuard and Jiang et al. [30] work in an automatic mode. VMGuard facilitates automatic policy extraction from heat templates. VMGuard inherits the probabilistic model of LeaPS [41]. Others [30], [40] and [37] also employ probability based dependency model. VMGuard and Patron [37] are the only self-reliant works; VMGuard relies on its audit phase for the correction of implementation flaws which may pose a threat to its accuracy. Congress [53] and Ligatti et al. [35] are the only works with policy expressiveness as all other works have customized policies. Ligatti et al. [35] and Weatherman [10] can support dynamic modeling, whereas for VMGuard the model remains fixed, yet they are interoperable (manual effort required) with different cloud platforms.



Platform	Proposal	Methods	Coverage	Features					
				Proactive	Automatic	Dynamic	Probabilistic	Expressive	Self-Reliant
Non-Cloud	Jiang et al. [30]	Regression Technique	N/A	✓	✓	-	✓	-	✓
	Ligatti et al. [35]	Model Checking	N/A	✓	N/A	✓	-	✓	✓
Cloud	PVSC [40]	Custom Algorithm	Both	✓	-	-	✓	-	-
	Weatherman [10]	Graph-theoretic	Virtual Infr.	✓	-	✓	-	-	-
	Congress [53]	Datalog	Both	✓	-	-	-	✓	-
	LeaPS [41]	Custom + Bayesian	Both	✓	-	-	✓	-	-
	Patron [37]	Custom Algorithm	User-level	✓	-	-	✓	-	✓
	VMGuard	Custom Algorithm	Virtual Infr.	✓	✓	-	✓	-	✓

Table 6: Comparing proactive solutions with VMGuard

### 6.3 State-Based Proactive Verification

Table 7 summarizes the comparison between existing works on state-based proactive verification and VMGuard. The first and second columns enlist existing works and their verification methods, respectively. The next eight columns compare these works according to different features, i.e., retroactive, intercept-check (I-C), proactive, incremental (Incr.), parallel workload distribution (Paral.), pruning-based verification, and multi-threading to manage multiple requests (M-Thread). The last two columns compare the scope of network isolation works, i.e., virtual networks (Vir. Net.), and data plane (D. Plane) in such networks. The main benefit of VMGuard over those works is that VMGuard provides a real-time response while verifying virtual network isolation in the data plane. To that end, VMGuard’s unique combination of features is: proactive, incremental and pruning.

There exist several works (e.g., [36, 64, 31, 69]) for virtual network isolation verification. Among them, NoD [36], Plotkin et al. [64] and TenantGuard [69] adopt a

retroactive approach, which detects an isolation breach after the fact. Specifically, NoD [36], is a logic-based verification engine that checks reachability policies using Datalog. Plotkin et al. [64] leverage the regularities existing in data centers to lessen the verification overhead using bi-simulation and modal logic. However, both of these works may cause hours to days of delay in verifying reachability. Whereas, TenantGuard [69] achieves verification time of 18 minutes for the same dataset by performing a hierarchical verification approach. However, for policy verification, TenantGuard relies on Congress [53], which causes a significant delay (as discussed in Chapter 2). Unlike these works, VMGuard achieves a practical response time (e.g., in a few milliseconds), as reported in Chapter 4.2 by adopting a proactive approach. There exist some other works (e.g., [31, 33, 32, 24, 23, 73]) for SDN-based or traditional networks. Among them, NetPlumber [31] leverages verifying hypotheses before deploying, but it is only applicable to SDN-based networks. In contrast, *VMGuard* verifies hypothesis in the virtual network environment for clouds.

There exist some proactive verification solutions (e.g., [53, 10, 41, 40]) for clouds. Weatherman [10] performs proactive verification on the virtual infrastructure based on the future change plan. Similarly, Congress [53] performs proactive verification over the proposed hypothetical configuration for the cloud. Both of those works rely on manual identification of future plan, and otherwise, cause a significant delay as an intercept-and-check solution. Whereas, VMGuard adopts an automated proactive approach (based on dependency model), and achieves a response time of a few milliseconds. Similar to VMGuard, PVSC [40] and LeaPS [41] achieve the response time in milliseconds. However, those works rely on signatures and cannot detect many isolation breaches (as demonstrated in Chapter 3). To that end, VMGuard adopts a state-based approach and hence, overcomes this limitation.

Proposal	Methods	Feature								Scope	
		Retroactive	I-C	Proactive	Incr.	Paral.	Real-Time	Pruning	M-Thread	Vir. Net.	D. Plane
Weatherman [10]	Graph-theoretic		✓	✓	✓					✓	-
Congress [53]	Datalog			✓						-	-
PVSC [40]	Custom algorithms			✓	✓		✓			-	-
LeaPS [41]	Custom + Bayesian			✓	✓		✓			-	-
NoD [36]	Datalog	✓			✓			✓		✓	✓
Plotkin et al. [64]	SMT Solver	✓			✓					✓	✓
Madi et al. [38]	CSP Solver	✓								✓	✓
Cloud Radar [9]	Graph-theoretic	✓								✓	-
TenantGuard [69]	Custom algorithms	✓			✓	✓				✓	✓
<b>VMGuard</b>	Custom algorithms		✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 7: Comparing existing solutions with VMGuard

# Chapter 7

## Other Contributions

During this master thesis study, other than VMGuard, which is the main contribution of this master thesis project, we also contributed to other projects that are described in the following sections.

### **7.1 ProSAS: Proactive Security Auditing System for Clouds through Caching and Pre-Computation**

The dynamic nature due to self-service in clouds calls for run-time auditing to ensure continuous security compliance. However, the existing auditing approaches, intercept-and-check and proactive ones, fail to provide a practical response time due to the sheer size and multi-tenancy in most of the cloud environments. ProSAS is a proactive security auditing system, which reduces the response time and significantly improves the efficiency over the existing auditing approaches. The main idea is to perform costly verification proactively (ahead of time) to reduce run-time delay. ProSAS

builds a dependency model leveraging the relationship between cloud events to trigger pre-computation, and performs verification at the occurrence of the critical event. ProSAS further employs caching techniques to improve the efficiency of search against the stored pre-computation results.

Our contribution to this work are listed in the following.

- Implemented a cloud environment by utilizing OpenStack [52] version Mitika. In our environment, a controller and compute nodes are implemented to collect input for ProSAS. The connecting technology between different nodes is LinuxBridge.
- Implemented and evaluated the caching mechanisms using LRU (Least Recent Used) and MRU (Most Recent Used), which improved the hit rate of values searched.

ProSAS is an extension of [40] and is currently under revision for the IEEE Transactions on Dependable and Secure Computing journal.

## **7.2 ProxiMet: Security Metrics for Evaluating and Mitigating Co-residency Threats in Public Cloud**

Multi-tenancy in cloud benefits the resource optimization but also lead to several security concerns such as co-residency of tenant’s virtual infrastructures, including VMs and virtual networks. The nature of co-residency demands a quantitative approach for

measuring the relative distance between cloud resources, which is currently missing in most existing works. ProxiMet is a quantitative approach to security compliance auditing which devises security metrics to define a distance-based co-residency level of tenant’s virtual infrastructures.

Our contribution to this work are listed in the following.

- Implemented a simulator that simulated the cloud environment to distribute virtual resource. Further, implemented the suite of security metrics which calculated the distance of co-residency for a virtual resource in the environment.
- Evaluated the security metrics for two real cloud data sets. The first data set was from one of the largest telecommunication service provider and the second from Google public cloud.

ProxiMet is currently under revision for the IEEE Transactions on Dependable and Secure Computing journal.

### **7.3 TenantGuard+: Efficient Cloud Network Policy Verification at Runtime using Formal Methods**

In this work, we audit the compliance of high-level policies specified by the network operators or by the cloud tenants, using a twofold approach. First, we used state of the art network verification tool TenantGuard, to model the network behavior and extract information about the relationship between different components in the virtualized network, such as which VM is reachable from which others, the flow path

information such as, the path length from the source to the destination. Second, we leverage the results provided by the network verification tool to verify the compliance of the networking configuration against predefined high-level policies. We formalize the network policy as CSP (Constraint Satisfaction Problems), for a constraint solver, namely Sugar [66], to validate the compliance.

Our contribution to this work is listed in the following.

- Performed data collection from TenantGuard [69] for different data sets, collecting all pair reachability paths.
- Installed an SAT solver, implemented data pre-processing, parsing and data mapping scripts. Evaluated CSP files for 100 iterations executed under SAT solver to obtain timing results.

TenantGuard+ is a paper in preparation for submission to a journal.

## **7.4 Modeling NFV Deployment to Identify the Cross-level Inconsistency Vulnerabilities**

By providing network functions through software running on standard hardware, Network Functions Virtualization (NFV) brings many benefits, such as increased agility and flexibility with reduced costs, as well as additional security concerns. Although existing works have examined various security issues of NFV, such as vulnerabilities in VNF software and DoS, there has been little effort on a security issue that is intrinsic to NFV, i.e., as an NFV environment typically involves multiple abstraction levels, the inconsistency that may arise between different levels can potentially be

exploited for security attacks. In this paper, we propose the first NFV deployment model to capture the deployment aspects of NFV at different abstraction levels, which is essential for an in-depth study of the inconsistencies between such levels. Based on the model and an implemented NFV testbed, we present concrete attack scenarios in which the inconsistencies are exploited to attack the network functions in a stealthy manner. Finally, we study the feasibility of detecting the inconsistencies through verification.

Our contribution to this work is listed in the following.

- Assisted in designing deployment model to capture the deployment aspects of NFV at different abstraction levels.
- Assisted with other editorial efforts for publication.

This paper is currently submitted to CloudCom19.

## **7.5 ERDC: Ericsson Research Demo Cloud**

We have played a major role in developing ERDC which is a proof of concept platform for research works developed at Audit Ready Cloud (ARC) project [5]. The platform integrates all major auditing approaches at different cloud layers. Figure 27 presents a reference architecture integrating our state-of-art research solutions. Upcoming modules are planned proof-of-concept to be integrated in future, and only platform initialization and deployed modules are part of the contribution.



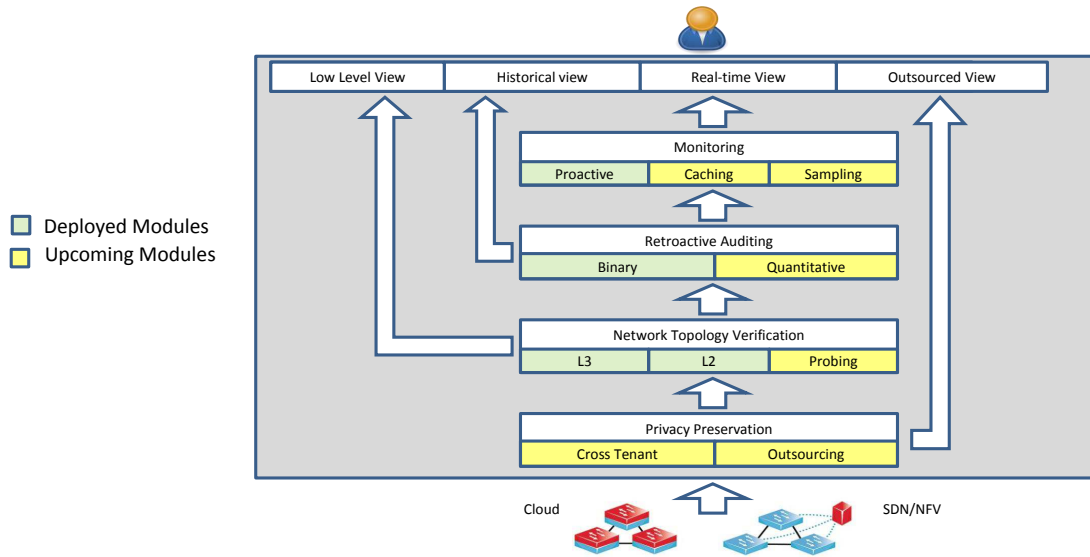


Figure 27: ARC tools integration reference architecture

**The Platform:** The platform runs in a virtual machine deployed within Ericsson research cloud. The machines is a headless server and is virtualized using virtualbox [62] and managed by phpvirtualbox client [6]. We run a controller node and four compute nodes to host infrastructure as a service cloud service deployed using open source cloud platform OpenStack [52]. We use OpenStack version Mitika with network communicating mechanism ML2 with OpenvSwitch, which is most used real cloud configuration [55]. The machines inside are reachable via Apache HTTP proxy where `URL/horizon`<sup>1</sup> serves OpenStack Horizon (cloud management dashboard) [49] and `URL/arc` connects to ARC auditing dashboard. Dashboard provides a detailed view of the cloud environment services and status with alert warning mechanisms. It centralizes the control for all the auditing tools using which an auditor can make on-demand requests and analyze results from each tool. We use HTML, CSS, JS,

<sup>1</sup>The URL to connect demo machine is a public IP address and is disclosed for Ericsson's internal use.

Bootstrap, and Angular for front-end and back-end in PHP. Currently, ERDC comprises of following tools integrated:

- **Layer 2 Verification:** The cloud layer two is being audited with ISOTOP [38], which uses flow rules from virtual switches of compute nodes to detect inter-tenant and intra-tenant breaches.
- **Layer 3 Verification:** The cloud management layer is audited using Tenant-Guard [69], which verifies all pair reachability amongst virtual machines.
- **Proactive Verification:** Existing solutions were after the fact detection where proactive verification precomputes in-advance for the critical event to prevent violation.
- **Multi-tenant Co-residency:** The suite of co-residency matrices Proximet determine proximity between tenants sharing an environment.
- **Log Anonymization:** Sharing of cloud logs is made easier by SegGuard which prevents privacy, preserving the utility of logs for third-party auditing.

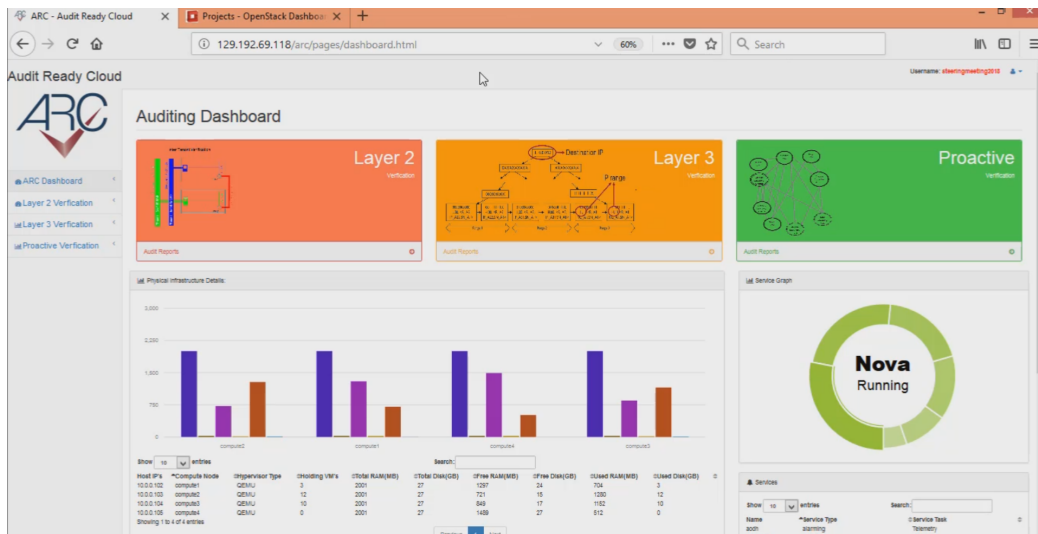
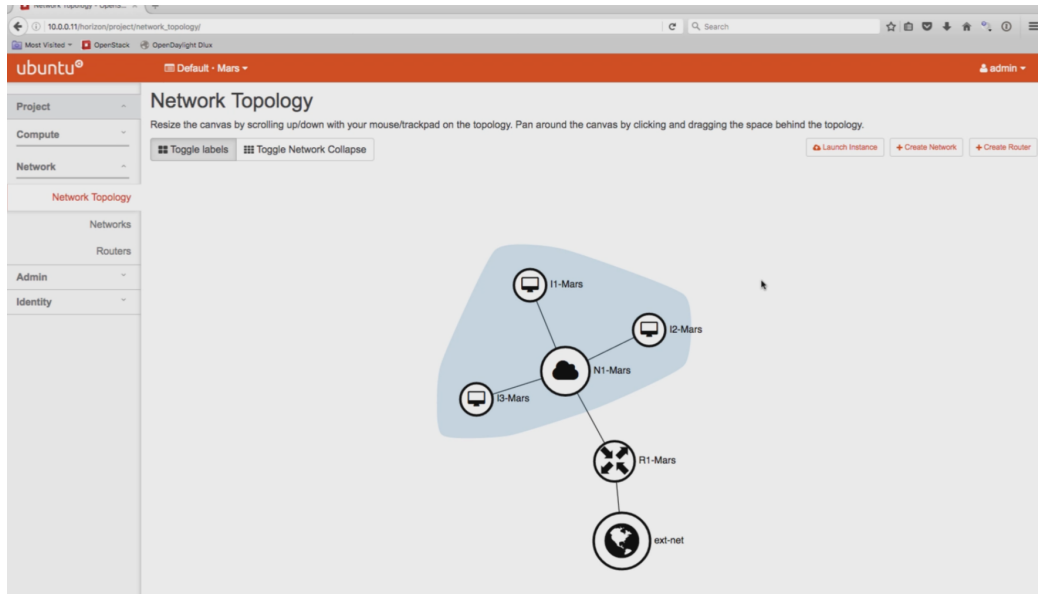


Figure 28: Cloud management using OpenStack Horizon and Audit Ready Cloud dashboard covering complete cloud environment

ARC - Audit Ready Cloud x Projects - OpenStack Dashboard x +

129.192.69.118/arc/pages/AboutCODASPY.html 60%

### Audit Ready Cloud

## About Layer 2 Verification

Public clouds enable cost effectiveness by adopting the Multi-Tenancy model to optimize resources usage. However, multiplexing virtual resources belonging to different tenants on the same physical substrate may lead to critical security concerns such as cross-tenants data leakage and denial of service.

Particularly, virtual networks isolation failures are among the foremost security concerns in the cloud. For instance, virtual machines (VMs) belonging to different corporations and trust levels may share the same set of resources, and their connecting virtual networks are multiplexed over the same substrate, which opens up opportunities for inter-tenant isolation breaches. Consequently, cloud tenants may raise questions like: "How to make sure that all my virtual resources and private networks are properly isolated from other tenants' networks, especially my competitors? Are my vertical Network Segments (e.g., for finance, human resources, etc.) properly segregated from each other?"

More specifically, in large scale OpenStack-based cloud infrastructures, layer 2 virtual networks are implemented on the same server using VLANs, and across the physical network through VXLAN as an overlay technology. On each physical server, disjoint VLAN tags are assigned to ports connecting VMs that are part of different isolated virtual networks. Furthermore, a unique VXLAN identifier is assigned per isolated virtual network in order to extend layer

ARC - Audit Ready Cloud x Projects - OpenStack Dashboard x +

129.192.69.118/arc/pages/record\_details.html?id=1&layer=overlayer 60%

### Audit Ready Cloud

## ARC Audit Report

Details

vmt1	3b6203d5-9583-487b-888d-0a530547f8ec
vmmac1	fa:16:3e:38:83:22
vlan1	2
vnet1	3b207cfe-28c0-4d16-9b23-d04352ee16dd
vmt2	be12856b-14ac-411e-95e3-8d089eda8918
vmmac2	fa:16:3e:95:50:a7
vlan2	1
vnet2	85d7543c-1a66-460d-9b8a-d02ed5d7199f

Figure 29: Layer 2 Auditing using Isotop [38]

ARC - Audit Ready Cloud x Projects - OpenStack Dashbo... x +

129.192.69.118/arc/pages/AboutTENANTGUARD.html 60% Search

Audit Ready Cloud username: admin@openstack.org

## About Layer 3 Verification

Cloud is a multi-tenant environment. Co-Residency of different tenants always exists on cloud environment. Now what is your business machines are placed with your business rivals? Are your machines totally isolated from your rival? Can your rival breach your machines and gain useful information? Sounds scary. Isn't it?

**"Something" went wrong and D is hacked!**

Cloud environment is software virtualized environment created for resource pooling and resource sharing. And the very obvious inherited disadvantage of software development comes with cloud environment too, Software implementation bugs and vulnerabilities.

**[OSSA 2014-008]** Any tenant is able to create a port on another tenant's router!

**[OSSA 2015-021]** Security group rules are not effective on instances immediately!

Does intra tenant's leaks seems to be possible now?

The solution to problem of intra tenant leaks seems to fulfill by isolating tenants network from other tenant networks within cloud environment. This can be verified by checking no possible reachability between tenant and its rival.

ARC - Audit Ready Cloud x Projects - OpenStack Dashbo... x +

129.192.69.118/arc/pages/Output.html 60% Search

Audit Ready Cloud username: admin@openstack.org

## Sample Output for L3 Verification

How does Web server Ingress and Egress looks like?

All reachable devices from Web server [EGRESS]

All reachable devices to Web server [INGRESS]

How does Application server Ingress and Egress looks like?

Figure 30: Layer 3 Auditing using TenantGuard [69]

ARC - Audit Ready Cloud | Projects - OpenStack Dashboard | 129.192.69.118/arc/pages/AboutLEAPS.html

### About Proactive Verification

The existing retroactive methods of auditing catch a security violation after the fact, which leaves the cloud vulnerable for a certain period of time. For instance, our Layer 3 compliance verification method is a retroactive approach to verify network isolation between tenants in multi-tenant cloud environment. However, any existing isolation breach in Layer 3 may be exploited by an adversary before our retroactive tool catches it.

We consider a scenario, where an adversary has exploited the following cloud vulnerabilities to gain access into victim's machine.

VM2 is legitimately reachable to VM1

**[OSSA 2015-021]** Security group rules are not effective on instances immediately!

**[OSSA 2014-006]** Any tenant is able to create a port on another tenant's router!

ARC - Audit Ready Cloud | Projects - OpenStack Dashboard | 129.192.69.118/arc/pages/Monitoring\_Dashboard.html

### Audit Ready Cloud 1001

#### Monitoring Dashboard

##### Summary and Statistics

###### Requests Distribution

###### Blocked Requests per Event Type

##### Dependency Model

##### Recent Alert Notifications

Timestamp	Request Type	Security Property
2018-03-27 19:16:46.313632	attach security group	No downgrade of security group
2018-03-27 14:43:53.792725	attach security group	No downgrade of security group
2018-03-25 15:06:58.295348	attach security group	No downgrade of security group
2018-03-25 15:06:23.046349	attach security group	No downgrade of security group
2018-03-22 23:49:24.753384	attach security group	No downgrade of security group

#### All Requests Details

Timestamp	Tenant id	Request Type	Predicted Compliance	Proactive Action	Property	More...
2018-03-28 18:55:40.356826	a662779a0c414a3e8aef05c0b0314c6	attach security group	Not Violated	Allowed	No downgrade of security group	<a href="#">Go to Details</a>
2018-03-27 19:16:46.313632	a662779a0c414a3e8aef05c0b0314c6	attach security group	Violated	Denied	No downgrade of security group	<a href="#">Go to Details</a>

Figure 31: Proactive Auditing using LeaPS [41]

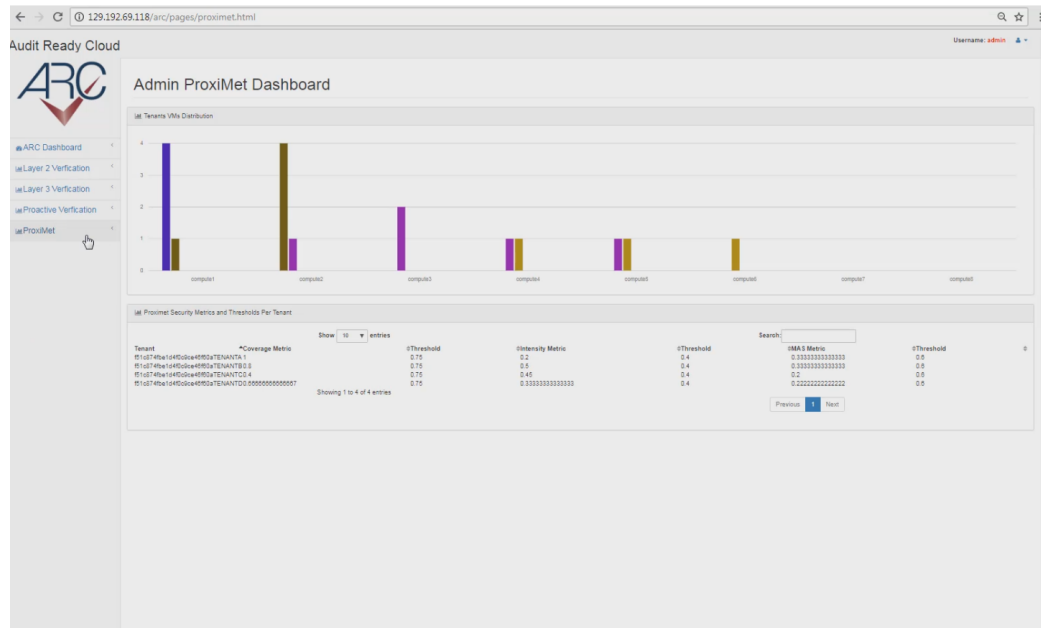
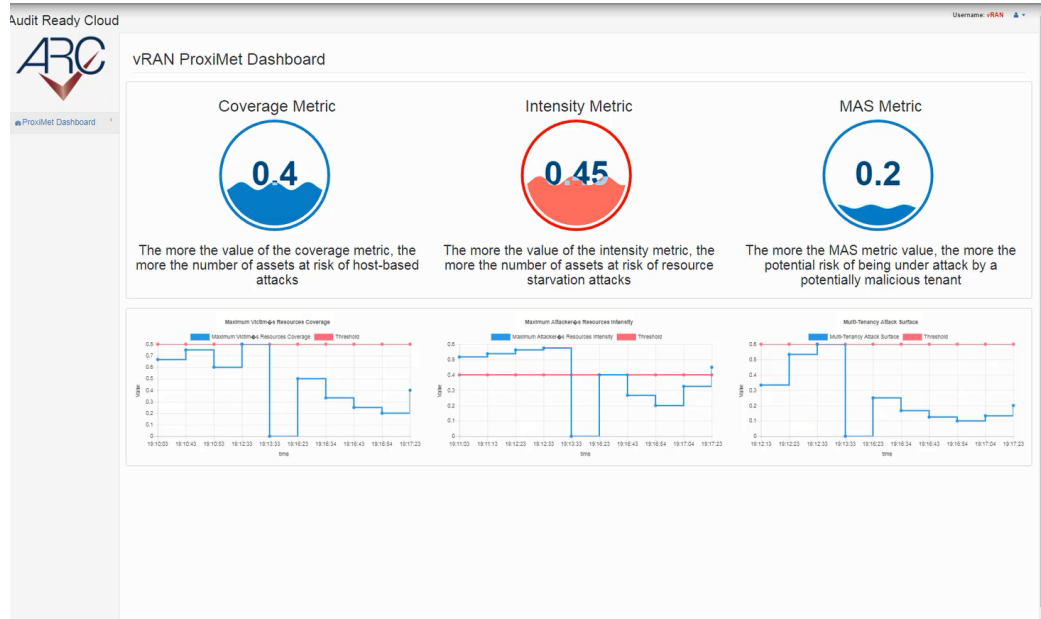


Figure 32: Security Metrics for Co-residency Evaluation using ProxiMet

Audit Ready Cloud

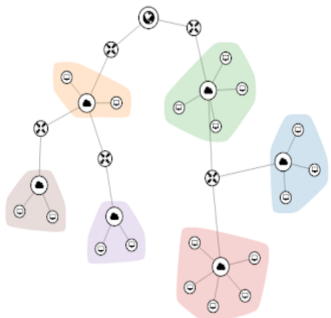
## CSP Anonymization Dashboard

SegGuard Anonymization

Number of Segments:

Number of Preserved Segments:

[Anonymize](#)



Original Network Topology

Audit Ready Cloud

## Auditor Dashboard

Show 10 entries

Search:

ID	Type	Status	Date	Seed Log	Non-Compliant	Compliant	Audit
1	Tokenization	Pending	2018-10-03 14:19:42	Download	✘	✔	
2	SegGuard	Pending	2019-02-27 16:02:02	Download	✘	✔	
3	SegGuard	Pending	2019-02-28 02:25:37	Download	✘	✔	

Showing 1 to 3 of 3 entries

Previous **1** Next

Figure 33: Log Anonymization using SegPriv



# Chapter 8

## Future Work and Conclusion

### 8.1 Future Work

As future work, firstly, we further plan to investigate the feasibility of integrating a formal policy specification language with our solution to enhance its policy support. Secondly, we plan to recommend policies based on type of virtual machine with deployed network function (i.e., firewall, IDS, etc.). Lastly, we see scope of policy optimization by prioritization and minimization of policies applied by VMGuard, which is currently not the case. We also see future in carrying forward our verification methodology to other cloud components (such as storage, identity management, etc.)

### 8.2 Conclusion

In this thesis, we addressed two major issues (e.g., inefficiency and inaccuracy) in the existing virtual network isolation verification approaches, and proposed VMGuard, which is a state-based proactive approach to efficiently verify network isolation policies in a large scale virtual infrastructure. To achieve better efficiency, VMGuard

proactively conducted the verification of future events. On the other hand, to ensure the effectiveness, VMGuard simulated all possible impacts on the current state and verified all those simulated states. Furthermore, we integrated VMGuard with OpenStack, and evaluated its performance and efficiency through extensive experiments using both real and synthetic data. We extended our solution to the network function virtualization (NFV) environment to proactively verify and enforce network isolation.

# Bibliography

- [1] M. Ali, S. U. Khan, and A. V. Vasilakos. Security in cloud computing: Opportunities and challenges. *Inf. Sci.*, 305:357–383, 2015.
- [2] O. Ali, J. Soar, and J. Yong. Challenges and issues that influence cloud computing adoption in local government councils. In *21st IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2017, Wellington, New Zealand, April 26-28, 2017*.
- [3] Amazon Web Services EC2. AWS EC2. Available at: <https://aws.amazon.com/ec2/>.
- [4] Amazon Web Services. Overview of security processes. Available at: [https://d1.awsstatic.com/whitepapers/Security/AWS\\_Security\\_Whitepaper.pdf](https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf).
- [5] ARC. Audit Ready Cloud. Available at: <http://arc.encs.concordia.ca/>.
- [6] ArchLinux. PhpVirtualBox. Available at: <https://wiki.archlinux.org/index.php/PhpVirtualBox>.
- [7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.
- [8] BayesFusion. GeNie and SMILE. Available at: <https://www.bayesfusion.com>.

- [9] S. Bleikertz, C. Vogel, and T. Groß. Cloud radar: near real-time detection of security failures in dynamic virtualized infrastructures. In *Annual Computer Security Applications Conference (ACSAC' 14)*, 2015.
- [10] S. Bleikertz, C. Vogel, T. Groß, and S. Mödersheim. Proactive Security Analysis of Changes in Virtualized Infrastructures. In *Annual Computer Security Applications Conference (ACSAC' 15)*, 2015.
- [11] Y. Cai, F. R. Yu, and S. Bu. Cloud radio access networks (C-RAN) in mobile cloud computing systems. In *2014 Proceedings IEEE INFOCOM Workshops, Toronto, ON, Canada, April 27 - May 2, 2014*.
- [12] CISCO. VNI Mobile Forecast Highlights Tool. Available at: [https://www.cisco.com/c/m/en\\_us/solutions/service-provider/forecast-highlights-mobile.html](https://www.cisco.com/c/m/en_us/solutions/service-provider/forecast-highlights-mobile.html).
- [13] Cloud auditing data federation. PyCADF: A Python-based CADF library, 2015. Available at: <http://docs.openstack.org/developer/keystonemiddleware/audit.html>.
- [14] Cloud Native Computing Foundation. Kubernetes. Available at: <https://kubernetes.io/>.
- [15] Cloud Security Alliance. Cloud control matrix CCM v3.0.1, 2014. Available at: <https://cloudsecurityalliance.org/research/ccm/>.
- [16] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v 4.0. Available at: <https://cloudsecurityalliance.org/working-groups/security-guidance>.

- [17] R. Dua, A. R. Raja, and D. Kakadia. Virtualization vs Containerization to Support PaaS. In *2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, March 11-14, 2014*.
- [18] ETSI. European Telecommunications Standards Institute. Available at: <https://www.etsi.org/>.
- [19] ETSI. Network Functions Virtualisation. Available at: [https://portal.etsi.org/NFV/NFV\\_White\\_Paper.pdf](https://portal.etsi.org/NFV/NFV_White_Paper.pdf).
- [20] ETSI. Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification. Available at: [https://www.etsi.org/deliver/etsi\\_gs/NFV-SOL/001\\_099/001/02.05.01\\_60/gs\\_NFV-SOL001v020501p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/001/02.05.01_60/gs_NFV-SOL001v020501p.pdf).
- [21] ETSI. Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Ve-Vnfm Reference Point. Available at: [https://www.etsi.org/deliver/etsi\\_gs/NFV-SOL/001\\_099/002/02.03.01\\_60/gs\\_NFV-SOL002v020301p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/002/02.03.01_60/gs_NFV-SOL002v020301p.pdf).
- [22] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. D. Millstein, V. Sekar, and G. Varghese. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*.
- [23] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. D. Millstein. A General Approach to Network Configuration Analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*.

- [24] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*.
- [25] Google. Google Cloud Platform. Available at: <https://cloud.google.com/>.
- [26] ISO Std IEC. ISO 27002:2005. Information Technology-Security Techniques, 2005. Available at: <http://www.iso27001security.com/html/27002.html>.
- [27] ISO Std IEC. ISO 27017. Information technology- Security techniques, 2013. Available at: <http://www.iso27001security.com/html/27017.html>.
- [28] J. Corbet. Trees I:Radix trees. Available at: <https://lwn.net/Articles/175432>.
- [29] JGraphT. A Java library of graph theory data structures and algorithms. Available at: <https://jgrapht.org/>.
- [30] Y. Jiang, E. Z. Zhang, K. Tian, F. Mao, M. Gethers, X. Shen, and Y. Gao. Exploiting statistical correlations for proactive prediction of program behaviors. In *Proceedings of the CGO 2010, The 8th International Symposium on Code Generation and Optimization, Toronto, Ontario, Canada, April 24-28, 2010*, pages 248–256, 2010.
- [31] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real Time Network Policy Checking Using Header Space Analysis. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*.

- [32] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*.
- [33] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*.
- [34] S. Li, L. D. Xu, and S. Zhao. The internet of things: a survey. *Information Systems Frontiers*, 2015.
- [35] J. Ligatti and S. Reddy. A theory of runtime enforcement, with results. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, pages 87–100, 2010.
- [36] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *Network and Distributed System Security Symposium (NDSS'15)*, 2015.
- [37] Y. Luo, W. Luo, T. Puyang, Q. Shen, A. Ruan, and Z. Wu. Openstack security modules: A least-invasive access control framework for the cloud. In *9th IEEE International Conference on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2, 2016*, 2016.
- [38] T. Madi, S. Majumdar, Y. Wang, Y. Jarraya, M. Pourzandi, and L. Wang. Auditing Security Compliance of the Virtualized Infrastructure in the Cloud:

- Application to OpenStack. In *Conference on Data and Application Security and Privacy (CODASPY' 16)*, 2015.
- [39] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, B. Godfrey, and S. T. King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*, pages 290–301, 2011.
- [40] S. Majumdar, Y. Jarraya, T. Madi, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi. Proactive Verification of Security Compliance for Clouds Through Pre-computation: Application to OpenStack. In *European Symposium on Research in Computer Security (ESORICS '16)*, 2015.
- [41] S. Majumdar, Y. Jarraya, M. Oqaily, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi. LeaPS: Learning-Based Proactive Security Auditing for Clouds. In *European Symposium on Research in Computer Security (ESORICS '17)*, 2015.
- [42] Microsoft. Microsoft Azure, 2010. Available at: <https://azure.microsoft.com/en-us/>.
- [43] OASIS. Open standards, Open source. Available at: <https://www.oasis-open.org/>.
- [44] OASIS. TOSCA Simple Profile for Network Functions Virtualization (NFV). Available at: <http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html>.



- [45] OASIS. TOSCA Simple Profile for NetworkFunctions Virtualization (NFV) Version 1.0; Committee Specification Draft 04. Available at: <http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd04/tosca-nfv-v1.0-csd04.pdf>.
- [46] OpenStack. Barbican. Available at: <https://wiki.openstack.org/wiki/Barbican>.
- [47] OpenStack. Heat : OpenStack Orchestration. Available at: <https://wiki.openstack.org/wiki/Heat>.
- [48] OpenStack. Heat : OpenStack Orchestration. Available at: [https://docs.openstack.org/heat/latest/template\\_guide/hot\\_guide.html](https://docs.openstack.org/heat/latest/template_guide/hot_guide.html).
- [49] OpenStack. Horizon. Available at: <https://wiki.openstack.org/wiki/Horizon>.
- [50] OpenStack. Mistral Overview. Available at: <https://docs.openstack.org/mistral/latest/overview.html>.
- [51] OpenStack. Nova network security group changes are not applied to running instances, 2015. Available at: <https://security.openstack.org/ossa/OSSA-2015-021.html>.
- [52] Openstack. OpenStack : Cloud Operating System. Available at: <https://www.openstack.org/>.
- [53] OpenStack. OpenStack Congress. Available at: <https://wiki.openstack.org/wiki/Congress>.
- [54] Openstack. OpenStack Identity Keystone. Available at: <https://wiki.openstack.org/wiki/Keystone>.

- [55] Openstack. Openstack User Survey, 2017. Available at: <https://www.openstack.org/assets/survey/April2017SurveyReport.pdf>.
- [56] OpenStack. OSSA-2014-008: Routers can be cross plugged by other tenants. Available at: <https://security.openstack.org/ossa/OSSA-2014-008.html>.
- [57] OpenStack. Service Function Chaining Extension for OpenStack Networking. Available at: <https://docs.openstack.org/networking-sfc/latest/>.
- [58] OpenStack. TOSCA-Parser. Available at: <https://wiki.openstack.org/wiki/TOSCA-Parser>.
- [59] Oracle. Java. Available at: <https://www.oracle.com/java/>.
- [60] Oracle. Java: Processes and Threads. Available at: <https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html>.
- [61] Oracle. Tacker - OpenStack NFV Orchestration. Available at: <https://wiki.openstack.org/wiki/Tacker>.
- [62] Oracle. VirtualBox. Available at: <https://www.virtualbox.org/>.
- [63] V. D. Piccolo, A. Amamou, K. Haddadou, and G. Pujolle. A Survey of Network Isolation Solutions for Multi-Tenant Data Centers. *IEEE Communications Surveys and Tutorials*, 2016.
- [64] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese. Scaling network verification using symmetry and surgery. In *Principles of Programming Languages (POPL' 16)*, 2015.
- [65] T. Probst, E. Alata, M. Kaâniche, and V. Nicomette. An approach for the automated analysis of network access controls in cloud computing infrastructures. In

- Network and System Security - 8th International Conference, NSS 2014, Xi'an, China, October 15-17, 2014, Proceedings*, pages 1–14, 2014.
- [66] Sugar. Sugar: a SAT-based Constraint Solver. Available at: <http://bach.istc.kobe-u.ac.jp/sugar/>.
- [67] vmware. vmware cloud. Available at: <https://cloud.vmware.com/>.
- [68] R. J. W. Definitions and Examples. In *Introduction to Graph Theory, Second Edition*, 1979.
- [69] Y. Wang, T. Madi, S. Majumdar, Y. Jarraya, A. Alimohammadifar, M. Pourzandi, L. Wang, and M. Debbabi. TenantGuard: Scalable Runtime Verification of Cloud-Wide VM-Level Network Isolation. In *Network and Distributed System Security Symposium (NDSS'17)*, 2017.
- [70] Wikipedia. Object Copying. Available at: [https://en.wikipedia.org/wiki/Object\\_copying#Deep\\_copy](https://en.wikipedia.org/wiki/Object_copying#Deep_copy).
- [71] Wikipedia. OSS/BSS. Available at: <https://en.wikipedia.org/wiki/OSS/BSS>.
- [72] H. Yang and S. S. Lam. Real-time verification of network properties using atomic predicates. In *2013 21st IEEE International Conference on Network Protocols, ICNP 2013, Göttingen, Germany, October 7-10, 2013*, pages 1–11, 2013.
- [73] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat. Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*.

- [74] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *J. Internet Services and Applications*, 2010.