Integrity Verification for Virtualized Networks Using Side-Channel

A S M ASADUJJAMAN

A THESIS

IN

THE DEPARTMENT

OF

CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN INFORMATION AND SYSTEMS ENGINEERING AT

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

OCTOBER 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **A S M ASADUJJAMAN**

Entitled: **Integrity Verification for Virtualized Networks Using Side-Channel**

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

DOCTOR OF PHILOSOPHY (INFORMATION AND SYSTEMS ENGINEERING)

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. Mustafa K. Mehmet Ali	Chair
Dr. Elias Bou-Harb	External Examiner
Dr. Anjan Bhowmick	External to Program
Dr. Chadi Assi	Examiner
Dr. Lingyu Wang	Examiner
Dr. Suryadipta Majumdar	Thesis Supervisor

Approved by ______________________

Dr. Jun Yan, Graduate Program Director

2023 ______________________

Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Integrity Verification for Virtualized Networks Using Side-Channel

A S M ASADUJJAMAN

Concordia University, 2023

Virtualization of networks has recently attracted enormous interest as an enabler of high-performance, cost-effective, scalable, and reliable communication services (e.g., 5G). However, these advantages are accompanied by issues such as increased attack surface, software bugs, lack of visibility, and lack of control over in-the-cloud virtualized networks. These issues pose the risk of integrity breaches of virtualized networks preventing them from providing services as intended by their owners (i.e., network service providers). Therefore, to reap the benefits of virtualized networks, appropriate integrity verification mechanisms must be deployed to detect any integrity breaches that may arise due to these issues. On one hand, it is often challenging to find mechanisms to perform such verification under the constraints of limited access and high-scalability requirements of contemporary communication services, while, on the other hand, potential attacks are getting more and more sophisticated (e.g., attack on the underlying infrastructure, zero-day attacks, and runtime attacks). To that end, existing works can be mainly divided into two categories: pre-deployment and runtime. Firstly, existing pre-deployment approaches are applied before the deployment of virtualized networks and therefore, are unable to detect any breach of integrity at runtime. Secondly, existing runtime approaches require access to data that are typically unavailable to owners of virtualized network services. Moreover, even when such data are made available, collecting these data requires intrusive techniques that affect the performance and scalability of network services. In this thesis, we overcome all the above limitations of existing works by looking beyond what is possible with traditional direct observation-based approaches and focusing on the indirect effects of the attacks (a.k.a., side-channels). We propose a side-channel based integrity verification system that offers a practical and scalable approach without requiring access to data that are typically unavailable. For this purpose, we organize our work into three main phases. In the first phase, we propose an approach to verify the integrity of virtualized network function (VNF) chains; where the proposed system verifies a wide range of integrity breaches of VNF chains, such as, VNF bypassing, packet dropping, and packet injection without requiring access to the underlying cloud infrastructure on which the VNFs are deployed. In the second phase, we propose a mechanism to detect functional integrity breaches of the virtualized network functions (VNF) caused by code injection (through the exploitation of vulnerabilities at different levels of the virtualization ecosystem). Thus, the first two phases combined can provide overall integrity verification by guaranteeing that the components (i.e., VNFs) are working properly both collectively (i.e., packets are being forwarded properly through the service chains) and individually (i.e., each VNF is providing exactly the same functionality as intended). Finally, in the third phase, we improve the above solutions to become more efficient and resilient against adaptive attempts to deceive our mechanisms by proposing a continuous verification technique. In summary, this thesis contributes towards enhancing the comprehensiveness, practicality, and security of integrity verification for virtualized networks.

Acknowledgments

This thesis work is the result of help and collaboration from many people, to whom I am extremely grateful and would like to appreciate their support.

First, I would like to thank my PhD supervisor Dr. Suryadipta Majumdar. His continuous guidance and availability helped me the most to complete this thesis work. I am grateful to him for enlightening me with his profound knowledge and precise insights. His constructive criticism greatly helped me to improve my research skills throughout my PhD study.

I would also like to thank Dr. Lingyu Wang for his mentorship during my PhD. Despite his busy schedule, he always manages his time to listen to my various issues and to provide realistic solutions to them. I am grateful to the members of my PhD examination committee, Dr. Anjan Bhowmick, Dr. Jeremy Clark, Dr. Elias Bou-Harb, Dr. Mustafa Mehmet Ali, and Dr. Chadi Assi, for their insightful advice during different phases of this work. I also thank Dr. Mohammad Shah Alam, my master's supervisor, who continued his support to me during my PhD.

My sincere gratitude extends to all members of the Audit Ready Cloud group, especially Dr. Makan Pourzandi, Dr. Yosr Jarraya, Dr. Mohammad Ekramul Kabir, Momen Oqaily, and Hinddeep Purohit, with whom I collaborated throughout my PhD study. I also acknowledge the financial support of NSERC, Ericsson Canada, and Concordia University.

Finally, I would like to acknowledge the unconditional affection and continuous support of my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Network virtualization softwarizes and outsources traditional network functions (NFs) to the cloud in order to bring more scalable, rapidly deployable, flexible, and reliable Network Services (NS). Fueled by these benefits, the global market size of network virtualization is expected to exceed 36.3 billion USD by 2024 [2]. However, these incredible benefits of network virtualization also bring concerns about increased complexity, software bugs, lack of visibility, increased attack surface, and lack of control on the off-premises cloud infrastructure. As virtualized networks combine both traditional networking and cloud computing; the attack surface can be even larger than that of traditional networking and cloud computing combined. Therefore, to leverage the benefits of network virtualization, appropriate security verification mechanisms must be in place to detect any security violation that may arise due to these concerns [3, 4]. Finding such verification mechanisms is made difficult by the facts that: 1) cloud tenants typically do not have access to the underlying infrastructure [5], and 2) modern communication services are sensitive to performance, scalability, and deployability [6].

There exists a body of work on the integrity verification of virtualized networks. First, the *Pre-deployment Verification* approaches [7, 8, 9, 10] perform static verification of various policies and system specifications before actual deployment. These approaches rely on access to the policies and descriptions of both the Network Functions (NFs) and the underlying cloud infrastructure. However, this may not be feasible in many scenarios (e.g., information, such as, how many virtual machines may be running on a physical host is typically not made available to the cloud tenant [11]). Moreover, they cannot perform verification against breaches that might happen at runtime after deployment. Second, the *Runtime Verification* approaches [12, 13, 14, 15] perform verification of performance and correctness of the operations of the NFs at runtime. They require equipment external to the original cloud (e.g., a local replica of NFs [12]) to carry out such verification or require intrusive modification of the NFs [13, 14, 15]. Thus, all these existing approaches hardly meet the requirements of working without access to underlying infrastructure and being unintrusive (in terms of performance, scalability, and deployability).

1.2 Problem Statement

In this thesis, we propose verification mechanisms to guarantee the security of virtualized networks while respecting constraints of access, performance, scalability, and deployability. To that end, we look beyond what is possible with traditional direct observation-based approaches and analyze the indirect effects of the attacks (known as side-channels). In particular, we first propose an approach to verify the integrity of virtualized network function (VNF) chains as a whole. Second, we move closer to the virtualized network functions (VNF) themselves and propose mechanisms to detect breach of integrity of their software function. Third, we propose mechanisms for continuous verification (i.e., minimizing the detection time as much as possible).

In particular, this thesis work mainly addresses the following research questions:

(1) How can we verify that virtualized networks are forwarding traffic as intended by the tenant?

(2) How can we verify that virtualized networks are providing exactly the same functionality as desired by the tenant?

(3) How can we maximize security coverage while remaining unintrusive to preserve the high efficiency and performance of the network?

We discuss the above problems in detail in the following.

1.2.1 Forwarding Integrity Verification for Virtualized Networks Using Side-Channel

During the first phase of our work, we focus on the integrity verification of Network Functions Virtualization (NFV) service chains to ensure traffic forwarding as intended by the tenant. To this end, our goal is to address the limitation of all existing works and to overcome the previously mentioned challenges in the area of integrity verification of virtualized networks. Specifically, we focus on allowing NFV tenants to perform verification without requiring any infrastructure-level data by exploring techniques available in the area of network traffic throughput estimation. Additionally, we focus on detecting a wide range of attacks that are regarded as difficult to detect and avoided by existing works. Chapter 3 details our work on forwarding integrity verification for virtualized networks.

1.2.2 Functional Integrity Verification for Virtualized Networks Using Side-Channel

In the second phase of our work, we focus on ensuring that the virtualized network functions themselves perform as intended by the tenant. To this end, we note that by exploiting the existing vulnerabilities, an attacker, ([16, 17]) can inject code into some virtualized network functions. We call it a breach of functional integrity. Here our goal is to provide a practical solution that works at runtime without access to the underlying host operating system while guaranteeing high-performance requirements. Chapter 4 further describes our idea on verifying functional integrity for virtualized networks.

1.2.3 Continuous Verification of Forwarding Integrity for Virtualized Networks Using Side-Channel

In the third phase of our work, we aim to guarantee continuous verification for our forwarding integrity verification approach to build a practical network service integrity verification system. To this end, we propose a new hybrid approach where we design and integrate active and passive techniques to work side-by-side to confirm inconsistencies. Chapter 5 further describes our idea on achieving continuous verification of forwarding integrity for virtualized networks.

In summary, the three phases of this thesis work provide a comprehensive (i.e., covering both network service chains and individual network functions), practical, and secure mechanism for the integrity verification of virtualized networks by cloud tenants. In the following, we describe the link between these topics and, how they were identified. We start with the goal to provide a solution for the integrity verification of virtualized networks and find that these networks can be divided into two layers of abstraction: 1) network service chains and, 2) network functions. Thus to have a comprehensive security solution, we cover both of these layers in the first two phases of our PhD research. A key challenge faced during the early phases of our work is that attackers may try to evade our verification mechanism by acting only when they cannot be detected. Therefore, in the third phase of our work, we develop a mechanism for continuous verification without leaving any window for attackers to go undetected.

1.3 Contributions

The main contributions of this thesis work are towards enabling cloud tenants to verify their in-the-cloud virtualized network services practically and efficiently. To this end, we propose a side-channel-based system, which supports verification of end-to-end service (i.e., both forwarding and function), overcomes the limitations of lack of access of cloud tenants to the underlying cloud infrastructure, and uses techniques that allow performance of the network to remain unaffected. We elaborate on these contributions in the following.

Firstly, our work focuses on the verification of forwarding integrity of virtualized network services by cloud tenants where several network functions are chained together to create a complete network service. To that end, we propose a side-channel-based approach that does not require any infrastructure-level data. In this work, we first send two rounds of probe packets to create artificial congestion (without actually causing any network overhead) and see their effects in terms of delay between packet pairs (a.k.a., packet-pair dispersion). We then apply a machine learning (clustering) algorithm which enables our integrity verification approach to decide whether there is any breach of forwarding integrity.

Secondly, we propose a mechanism for verification of functional integrity that complements our forwarding integrity mechanism to provide overall network service integrity verification. In this work, we collect performance metrics that are already made available to the tenants for a purpose other than security verification and derive side-channel information from them to achieve verification of functional integrity.

Thirdly, we propose a continuous security verification mechanism that extends our forwarding integrity verification mechanism by enabling it to perform verification continuously. To that end, we design a hybrid approach that leverages a passive observation-based mechanism and does not completely rely on the previous probing-based active approach.

In summary, our main contributions are the following:

- As per our knowledge, we propose the first side-channel-based approach that eliminates the need for infrastructure-level data for verifying the forwarding integrity of virtualized network services.
- We are the first to propose a mechanism for verifying the functional integrity of virtualized network functions without requiring access to the underlying host operating system and keeping network performance unaffected.
- We are also the first to propose a mechanism to perform continuous black-box verification of virtualized network services.
- We implement and integrate our proposed systems into major cloud platforms such as OpenStack [18] and Amazon Elastic Compute Service (ECS) [19]. Evaluation with both synthetic and real data shows the effectiveness and efficiency of our approach.

1.4 Thesis Structure

This thesis is organized into six chapters. Chapter 1 introduces this thesis work. Chapter 2 reviews the related literature. Chapter 3 discusses the result of our blackbox approach to verification of NFV service chain integrity. Chapter 4 presents our research work on functional integrity verification of virtualized network functions. Chapter 5 details our work on continuous integrity verification of virtualized networks using side-channel. Finally, we conclude our thesis in Chapter 6.

Chapter 2

Related Work

In this chapter, we review existing works related to our problem areas. We also present a qualitative comparison between our proposed solutions and existing works in this chapter. Our review of the related works is organized as follows. Firstly, in Section 2.1, we review the literature and show a qualitative comparison between existing works and our proposed solutions in the area of forwarding integrity verification. Secondly, in Section 2.2, we review the literature and show a qualitative comparison between existing works and our proposed solutions in the area of functional integrity verification.

2.1 Forwarding Integrity Verification

Most existing works in forwarding integrity verification have focused on pre-deployment verification of the integrity of VNFs themselves [7, 8, 9, 10] and little attention has been given to verifying the integrity of service chains in runtime. Moreover, the majority of the works that propose a technique to verify service chain integrity in runtime mainly use hop-by-hop verification using per-packet tag and require access to the underlying infrastructure (e.g., reading flow rules [13, 14] or reprogramming firmware [20, 21, 15, 22, 23]).

2.1.1 Pre-deployment Verification

To verify the service chains before actual deployment, several works have been proposed. These works can be broadly classified into four types: (i) static script verification, (ii) sand-box deployment, (iii) resource requirement determination, and (iv) optimal placement. EasyOrchestrator [24] proposes to verify the correct ordering of VNFs in a service chain. Bouten [8] proposes a detection mechanism for affinity and anti-affinity violation of VMs in a service chain. Durante [9] proposes a formal modeling and verification approach for service chains. They demonstrate how their model can detect service chains that are placed in incorrect order. 5GTango [25] proposes a verification and validation framework for network services using a sand-box environment. Touloupou [26] presents an approach for determining resource requirements in NFV by performing correlation between metrics (E2E latency, throughput, CPU utilization, number of users etc.). Marchetto [27] proposes a framework for optimal placement of VNFs onto physical resources (e.g., servers).

2.1.2 Runtime Verification

Several works propose solutions for verifying the correct enforcement of service chain specification at runtime. Shin et al. [13] proposes an approach that is capable of verifying service chain forwarding even when there can be a dynamic change in forwarding behavior (e.g., FW is used only when DPI detects anomaly). It does so by repeatedly translating service chain specification and flow tables into an intermediate representation called pACR and then comparing both translations. ChainGuard [14] generates topology from flow tables and then compares it with the service chain forwarding graph. vHSFC [28], vSFC [21] and FlowCloak [15] use a tagging-based hop-by-hop verification mechanism. In [29, 30] authors propose techniques for detecting the presence of malware (leading to a breach of functional integrity) at runtime but their approach requires interception of system calls which can affect the performance of the NFs (e.g., increased network latency [31, 32]).

Finally, Oqaily et al. [33] proposes an innovative approach combining both formal methods and machine learning that can potentially be adapted to quickly verify the integrity of service chains at runtime given access to the underlying infrastructure.

2.1.3 Traffic Throughput Estimation

In the literature, extensive work has been done in estimating available capacity (which we use to estimate throughput) using Packet Pair Dispersion Technique [34, 35, 36, 37, 38, 39, 40]. However, these techniques are not applicable in our context. Firstly, because, they mainly measure available capacity at the bottleneck link (i.e., link with the lowest capacity) and cannot measure available capacity at a link that may not be the bottleneck link. Secondly, most of the existing tools require cooperation from both ends (i.e., require on-premise deployment) of the path for estimation. It makes their deployment very difficult [41] and cannot provide an in-cloud solution. Finally, existing tools have large relative errors [42] making them unsuitable for integrity verification purpose.

2.1.4 Comparison Among Related Works

Table 2.1 summarizes the comparison among existing works and our proposed solutions for forwarding integrity verification, namely, APPD and $(SC)^3$. The first column lists the works. The second column lists their main techniques. The next four columns indicate different design goals, such as continuous (i.e., without leaving a window for an attacker to go undetected), unintrusive (i.e., without affecting performance and deployability), blackbox (i.e., without requiring access to the underlying cloud infrastructure), and in-cloud (i.e., on-premise hardware/software which would increase the total cost of ownership (TCO) [43] is not needed). The next eight columns list different integrity breaches and threats that are mentioned in Section 5.2.3.

As shown in Table 2.1, most existing works are unable to provide continuous verification while remaining unintrusive. Moreover, most of the works that verify service chain integrity require access to the underlying infrastructure (e.g., reading flow rules [13, 14] or reprogramming firmware [20, 21, 15, 22, 23]). Additionally, some works propose on-premise mechanism [23] (i.e., they are not in-cloud solutions). Not only that the existing works do not provide a blackbox approach but also they cannot verify different types of service chain integrity breaches. Firstly, they cannot detect packet bypass when all VNFs are bypassed or the last VNF in the chain is bypassed [20, 21, 15, 22, 23]. This is because they need to first assign a tag to the packets which can be done only after the packets have arrived at least at one (first) VNF. Also, they need to verify the tags in the last VNF of the chain (at the latest). Additionally, existing works cannot detect both packet dropping and fake packet injection before the first VNF in the chain [20, 21, 15, 22, 23]. This is because existing works need to collect statistics which is available only when the packets arrive at least at the first VNF.

Table 2.1: We propose the first blackbox approach for continuous sanity checking of service chains in NFV while covering all likely integrity breaches

Proposals	Approach	Goals				Capabilities							
						VNF bypass			Packet drop		Packet injection		
		Continuous	**Unintrusive**	**Blackbox**	**In-cloud**	**Full SC**	**Partial (except last VNF)**	**Partial (last VNF)**	**Before first VNF**	**After first VNF**	**Before first VNF**	**After first VNF**	**Adaptive attack resistance**
FlowCloak [15]	Per-packet tag	✓	-	✓	-	-	✓	-	-	-	-	-	✓
vSFC [21]	Per-packet tag and acknowledgement	✓	-	✓	-	-	✓	-	-	✓	-	✓	✓
REV [20]	Per-packet per-switch tag	✓	-	✓	✓	-	✓	-	-	✓	-	✓	✓
Thang et al. [23]	Modification of end hosts	-	-	✓	-	✓	✓	✓	-	✓	-	✓	-
Shin et al. [13]	Forwarding device configuration	-	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	-
ChainGuard [14]	Forwarding device configuration	-	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	-
APPD	Side-channel	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$(SC)^3$	Side-channel	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

2.2 Functional Integrity Verification

In search of a solution to detect functional integrity breaches in virtualized 5G core, we investigated existing works close to our problem area. To that end, we looked at the area of 5G security, malware detection, network virtualization, and performance metrics correlation.

2.2.1 Detection of Functional Integrity Breach in 5G

Existing works in 5G security can be categorized by the related key technologies as follows: 1) Software Defined Networking (SDN), 2) virtualized Network Functions (NFs), 3) Network Slicing, and 4) Mobile Edge Computing (MEC). Among these, works that discuss security issues related to virtualized NFs are the closest to our research. Among the works in this area, authors report the possibility of 1) code injection attack [16], 2) migration of NFs to a less secure host to compromise it subsequently [17], and 3) propagation of malicious code to a co-hosted NF [44]. However, they do not provide any mechanism to perform integrity verification against these attacks.

2.2.2 Malware Detection

In the literature, extensive work has been done on detecting malware in applications [45]. However, existing malware detection works cannot be applied in our case for the following reasons: (i) we classify works that statically extract features from binaries for detection [46, 47, 48, 49, 50] as pre-deployment. However, these solutions might not be applicable against our threat model, especially when an attack will happen after the binaries (i.e., CNF images) are deployed and run, (ii) on the other hand, a few works that perform during run-time by using a signature-based approach cannot detect zero-day attacks [51], and (iii) works [29, 30] that attempt to detect zero-day attacks but use system features (e.g., system calls) usually cannot be applied due to challenges in collecting those features (e.g., lack of access to the underlying host operating system by a tenant) and even when those features can be collected, collecting them would require significant cost in terms of overhead (considering the high throughput requirement of 5G) due to interception and need for instrumentation of each CNF [52].

2.2.3 Performance Metrics Correlation

Correlation between different performance metrics (e.g., CPU utilization and memory utilization) of various elements can provide valuable insight into the behavior of a system. In [53], the correlation between different performance metrics (e.g., CPU utilization, memory utilization, received packets etc.) for a single VNF has been studied. On the other hand, [54] studied the correlation between metrics from different 5G Radio Access Network (RAN) cells. However, none of the above works studied the correlation among metrics of different network functions (multi-network function metric correlation) and therefore, cannot be adapted to serve our purpose.

2.2.4 Comparison Among Related Works

Table 2.2 summarizes the comparison among existing works and our proposed solution on functional integrity verification, namely, 5GFIVer. The first column lists the works. The second column lists their main approach. The next four columns indicate different capabilities, such as unintrusive (i.e., without affecting performance and deployability), Runtime (i.e., able to detect attacks that occur at runtime), blackbox (i.e., without requiring access to the underlying cloud infrastructure), and zero-day (i.e., able to detect previously unknown attacks).

Table 2.2: We propose the first blackbox approach for functional integrity verification

Proposals	Approach	Unintrusive	Runtime	Blackbox	Zero-day
Schultz et al. [46]	Extracting static features from binary executables	✓	-	-	✓
Hahn et al. [47]	Extracting static features from binary executables	✓	-	-	-
Shaukat et al. [48]	Extracting static features from binary executables	✓	-	-	✓
Chaganti et al. [49]	Extracting static features from binary executables	✓	-	-	-
Luo et al. [50]	Extracting static features from binary executables	✓	-	-	-
Wagener et al. [29]	Extracting dynamic features from binary executables	-	✓	-	-
Das et al. [30]	Extracting dynamic features from binary executables	-	✓	-	-
5GFIVer [15]	Side-channel	✓	✓	✓	✓

Chapter 3

Forwarding Integrity Verification of NFV Service Chains Using Artificial Packet Pair Dispersion

3.1 Introduction

Network Functions Virtualization (NFV) is considered one of the cornerstones of the emerging 5G technology due to its various benefits such as cost efficiency and greater flexibility [55]. NFV allows virtual network functions (VNF), such as firewalls, IDS, and DPI to be implemented as service chains over a third-party cloud infrastructure, such that the network service providers (i.e., NFV tenants) can leverage the benefits of NFV without having to deploy and manage their own infrastructures [56, 57, 58]. However, such outsourcing of VNFs might limit the capability of an NFV tenant to know whether their VNFs have been properly deployed in the underlying cloud infrastructure, as the third-party cloud providers would typically not allow the NFV tenant to access the infrastructure-level resources (e.g., SDN switches) and data (e.g., logs and configuration).

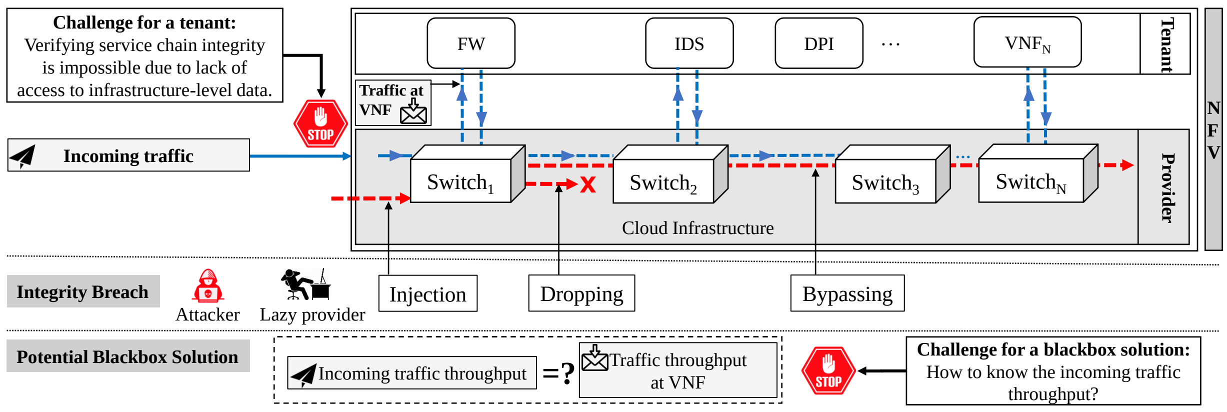
On the other hand, enabling the above-mentioned verification capability for NFV tenants is becoming more important with the growing security concerns in NFV infrastructure, especially those involving various types (e.g., VNF bypassing, packet dropping, fake packet injection) of integrity breach of VNF service chains [20, 21, 15, 22]. Such integrity breaches are mainly caused by misconfigured (e.g., [59]) or compromised (e.g., [60, 61, 62]) components of the underlying infrastructure (e.g., SDN switches), which could lead to severe security consequences, such as circumventing security mechanisms (e.g., virtual firewall or IDS) inside a service chain. Therefore, an interesting research challenge is to *enable the verification of service chain integrity for NFV tenants without requiring access to infrastructure-level resources or data*.

Most existing efforts (e.g., [13, 14, 63, 20, 21, 15, 22]) fall short of fulfilling this need. Specifically, some existing works (e.g., [13, 14, 63]) rely on the infrastructure-level data (e.g., flow rules and flow statistics in SDN switches) to verify service chain integrity. Other existing works (e.g., [20, 21, 15, 22]) aim to reduce the requirement of infrastructure-level data by using a cryptographic tagging mechanism at the VNF level. Nonetheless, those works require modifications (such as reprogramming the firmware) to infrastructure-level devices (e.g., SDN controller), which may not be practical with third-party providers. Moreover, those works are not designed to detect all types of integrity breaches (e.g., bypassing the last VNF, or all VNFs, in the service chain). To the best of our knowledge, there does not exist a *blackbox* approach (where tenant-level data along with the available side-channel data would be sufficient to verify service chain integrity).

In this thesis, we propose a *blackbox* approach, namely, *artificial packet-pair dispersion (APPD)*, to allow NFV tenants to verify service chain integrity without requiring any infrastructure-level data. Our key idea is twofold. First, if we could somehow enable the VNFs to estimate the throughput of incoming traffic to NFV (i.e., traffic flowing into the service chain), then by comparing this throughput to the actual throughput observed by each VNF along the service chain, the VNFs could then identify any integrity breach all by themselves (e.g., an increased throughput may indicate bypassing). Second, to address the key challenge of allowing VNFs to estimate the incoming traffic throughput, we rely on the fact that the inter-packet delay could be increased by (and thus indicate) congestion in a link. Specifically, our proposed approach consists of two major steps. First, APPD estimates incoming traffic throughput to NFV from the inter-packet delay by creating an artificial congestion (as natural congestion is rare in an NFV-like high-bandwidth infrastructure). Second, it detects service chain integrity breaches by comparing the estimated incoming traffic throughputs with the actual traffic throughput using a machine learning (clustering) approach. We will further elaborate on our motivation and idea through an example in Section 5.2.

In summary, our main contributions are the following:

- As per our knowledge, this is the first blackbox approach that eliminates the need for infrastructure-level data for verifying common integrity breaches (e.g., bypassing, fake packet injection, and packet dropping) in NFV service chains.
- We are the first to introduce a novel method of artificial packet-pair dispersion (APPD), which allows to the creation of artificial congestion in a high-bandwidth network like in an NFV infrastructure (where natural congestion is rare, if not impossible) for estimating incoming traffic throughputs to NFV. We believe this novel method for estimating throughput may find other applications beyond service chain integrity verification.
- As a proof of concept, we integrate APPD with OpenStack/Tacker, a popular choice for NFV deployment, and, through extensive experiments in a real network environment, we demonstrate both effectiveness and efficiency (i.e., negligible overhead) of APPD.



Challenge for a tenant:
Verifying service chain integrity is impossible due to lack of access to infrastructure-level data.
STOP
Incoming traffic
FW
IDS
DPI
…
VNF$_N$
Tenant
Traffic at VNF
Switch$_1$
Switch$_2$
Switch$_3$
Switch$_N$
Cloud Infrastructure
Provider
NFV
Integrity Breach
Attacker
Lazy provider
Injection
Dropping
Bypassing
Potential Blackbox Solution
Incoming traffic throughput
=?
Traffic throughput at VNF
STOP
Challenge for a blackbox solution:
How to know the incoming traffic throughput?

Figure 3.1: An example of integrity breaches in NFV (top) and a possible solution and its key challenge (bottom)

3.2 Preliminaries

This section first presents a motivating example. Then it defines our threat model and assumptions.

3.2.1 Motivating Example

The top of Fig. 3.1 depicts a simplified NFV deployment, with different integrity breaches (indicated by the red dashed lines), as well as the main challenge (the red stop sign). The bottom of Fig. 3.1 illustrates a potential solution and its key challenge.

NFV Deployment. The top slice of Fig. 3.1 shows an example of an NFV environment where VNFs are running on a third-party cloud provider's infrastructure. As shown in blue dashed lines, the incoming traffic is planned to pass through the service chain consisting of several VNFs, such as Firewall (FW), Intrusion Detection System (IDS), and Deep Packet Inspection (DPI) as well as their underlying cloud infrastructure ($Switch_1$, ..., $Switch_N$).

Integrity Breach in NFV Service Chains. The middle slice of Fig. 3.1 shows various integrity breaches including *injection* of fake packets, *dropping* legitimate packets, and *bypassing* one or more VNFs due to misconfigurations (e.g., [59]) by a cheap/lazy provider

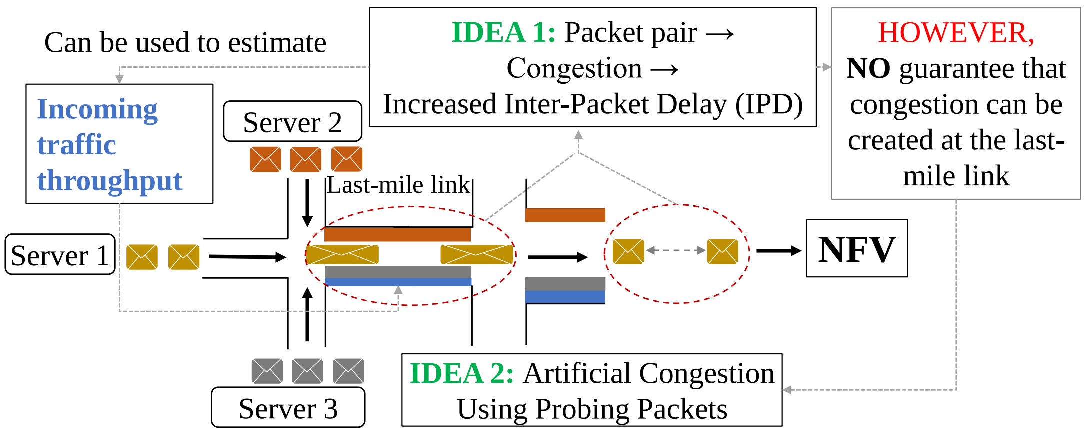

Can be used to estimate
IDEA 1: Packet pair → Congestion → Increased Inter-Packet Delay (IPD)
HOWEVER, NO guarantee that congestion can be created at the last-mile link
Incoming traffic throughput
Server 2
Last-mile link
Server 1
NFV
Server 3
IDEA 2: Artificial Congestion Using Probing Packets

Figure 3.2: The main ideas of APPD

or attacks by exploiting compromised resources (e.g., [60, 61, 62]). As a result, traffic may follow an entirely different path (as shown in the red lines) than planned paths. An NFV tenant cannot easily verify such an integrity breach, due to the limited access to the underlying infrastructure-level data (including the flow rules of $Switch_1$, ..., $Switch_N$).

Potential Blackbox Solution and Its Challenge. The bottom slice of Fig. 3.1 shows a potential *blackbox* solution that could avoid the need for infrastructure-level data. The solution compares the incoming traffic throughput (i.e., the traffic flowing into NFV to be processed by the service chain) and the traffic throughput actually observed at a VNF. However, it is not feasible for the VNFs to measure the incoming traffic throughput due to the fact that the VNFs are not directly connected to the incoming traffic. Therefore, the key challenge to this blackbox solution is: "*How to know the incoming traffic throughput?*".

3.2.2 Main Idea

Fig. 3.2 illustrates our main ideas as follows.

Idea 1: Estimation of Incoming Traffic Throughput. Our first idea is to estimate (instead of directly observing which would require access to the underlying cloud switches) the tenants' network traffic throughput by extending the concept of Packet-Pair Dispersion

(PPD) [34], where a traffic throughput can be estimated from inter-packet delay by causing a momentary congestion in a network. Particularly, the concept of PPD indicates that if two packets (e.g., the two yellow envelopes in Fig. 3.2) are transmitted at a rate that can cause congestion in a link, then this will lead to an increase in the inter-packet delay (IPD) between these two packets. Conversely, from the increase in the IPD, it is possible to estimate the network traffic throughput. However, in our context, it is almost impossible to directly apply natural PPD, as natural congestion in NFV-based networks is rare due to their high-bandwidth nature.

Idea 2: Artificial Packet-Pair Dispersion. To overcome the above-mentioned NFV-specific issue, our second idea is to *artificially* create a PPD using probing packets such that we no longer rely on natural PPD to estimate traffic throughput. Particularly, to generate artificial PPD, we send probing packets from multiple hosts (via different ingress links) for a short period of time (to ensure that there will be no significant overhead on the NFV environment or on any tenant resources, as also validated by our experimental results in Section 3.5). Afterwards, we estimate the tenants' network traffic throughput by leveraging a machine learning (i.e., clustering) based approach, and verify service chain integrity without requiring any infrastructure-level data. Section 3.3 will further elaborate on these ideas.

3.2.3 Threat Model and Assumptions

This work considers integrity breaches of service chains that may be caused when (i) a malicious attacker compromising *any* of the underlying forwarding devices (e.g., SDN switches [60, 61, 62]), or (ii) a cheap-and-lazy cloud provider [59] is misconfiguring (intentionally or by mistake) the underlying forwarding devices.

We consider a stronger threat model in comparison to existing works (e.g., [20, 21, 15, 22]) by including a wide range of attacks and attacker capabilities as follows. (i) *VNF* *bypassing:* Compromised or misconfigured switches may bypass one or more VNFs in the service chain; compromised switches may also collaborate with each other [23] to bypass a combination of VNFs (e.g., the entire service chain). (ii) *Packet dropping:* Compromised or misconfigured switches may drop packets at any switch (e.g., first switch) instead of forwarding as planned. (iii) *Packet injection:* Attackers may inject fabricated packets to overwhelm the VNFs at any position (e.g., before the first VNF). Many of these possibilities are deemed hard to detect and avoided by most existing works [23].

As the NFV tenant has access to the VNFs to deploy any security mechanism (i.e., there is no need for blackbox verification for VNFs), we exclude any attack on VNFs from our threat model. Similar to SDN switches in the cloud infrastructure, we consider that the tenants' gateway router can also be compromised by attackers [64]. However, as the tenant has access to the logs and configuration data of the gateway router, we consider that any changes in the forwarding rules in the gateway router can be verified by the tenant admin. Therefore, we only consider the violation of confidentiality (e.g., communication between the gateway router and NFV may not be trusted due to compromised secret keys) of the gateway router. We assume a hierarchical structure of the Internet bandwidth where links closer to the edge (e.g., NFV tenant) have lower capacity compared to the links closer to the core (e.g., NFV/cloud infrastructure) [65].

3.3 Methodology

This section presents the APPD methodology.

3.3.1 Overview

Fig. 3.3 shows an overview of our methodology, which contains two major stages. Stage 1 performs incoming traffic throughput estimation (detailed in Section 3.3.3), and Stage 2

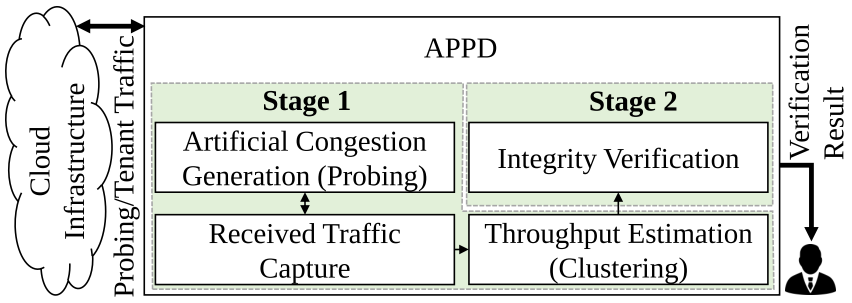
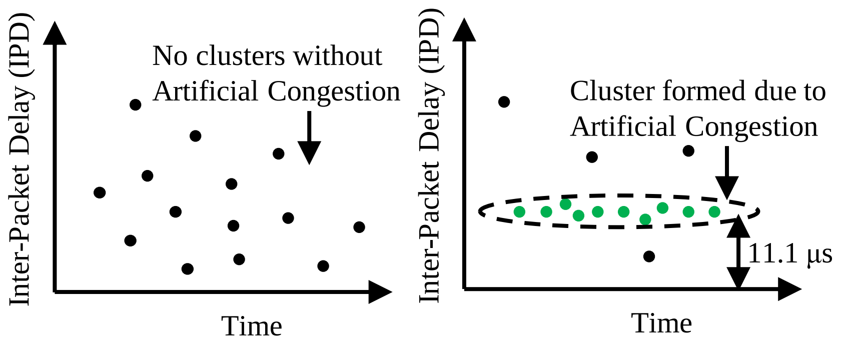


Figure 3.3: A high-level overview of APPD



(a) No artificial congestion (b) Artificial congestion

Figure 3.4: Clusters are formed due to artificially created congestion (the APPD effect)

performs integrity verification (detailed in Section 3.3.4). In Stage 1, APPD first sends probing packets to create artificial congestion in tenants' last-mile link and then relies on the received packets at the VNF (affected by artificial congestion) to estimate the incoming traffic throughput. In Stage 2, APPD performs integrity verification of the service chain by comparing the incoming traffic throughput with the throughput of the received traffic at a VNF (in a service chain).

3.3.2 The APPD Effect

The *APPD effect* refers to the formation of a distinguishable cluster of IPD values due to the artificial congestion created by APPD (as shown in Fig. 3.4). These clusters are distinguishable by their high density in the data space as shown in Fig. 3.4b. Moreover,

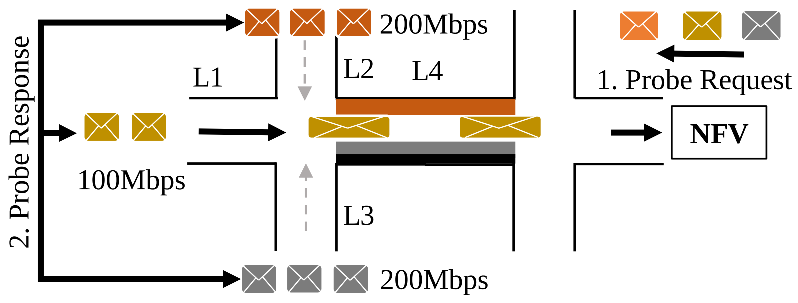


Figure 3.5: An example of probe generation

they have a very specific shape of being spread in the horizontal direction and having a very small height. Although the cluster in Fig. 3.4b is easy to identify visually in this particular case, the clusters may not be easy to identify manually in general. To this end, we exploit the high spatial density of the clusters to identify them using a density-based clustering algorithm, which will be detailed in the next section.

3.3.3 Stage 1: Estimation of Incoming Traffic Throughput

This stage consists of the following three steps: (Step 1.1) artificial congestion generation, (Step 1.2) received traffic capture, and (Step 1.3) throughput estimation using clustering.

Step 1.1: Artificial Congestion Generation. This step is to create an artificial congestion at the tenants' last-mile link for a very small duration (e.g., 50ms). To do so, APPD generates probing packets that will enter the tenants' gateway router through multiple ingress ports. To achieve this, probe request traffic is generated from the VNFs using a request/response protocol (e.g., HTTP get request, ICMP echo request, etc.) to different hosts that are connected through different ingress ports of the tenants' gateway router. Thus, when the reply packets come back to the tenants' gateway router, the accumulated last-mile link traffic will experience artificial congestion.

Example 1 *As shown in the road-junction analogy in Fig. 3.5, the junction (i.e., router)* *connecting roads (i.e., ingress links) L1, L2, L3 and the egress link L4 is tenants' gateway router. Probe traffic, shown in gray/orange/golden envelopes (each color for a different ingress port), are sent to this router as a response to requests sent from NFV (specifically from the first VNF in the service chain) through ingress links L1, L2, and L3. As a result, artificial congestion is created, and the traffic leaves the last-mile link L4 experiencing the APPD effect.*

The probe traffic generation module is designed to ensure there will be artificial congestion to cause the APPD effect on the traffic at the last-mile link. To achieve this, the APPD adjusts the probe throughout based on the actual received traffic throughput at VNF (which should be the same as the last-mile link traffic when there are no integrity breaches) such that the combined throughput of probe traffic and tenants' traffic will be equal to the last-mile link capacity. As a result, the APPD effect will induce distinguishable patterns in terms of the inter-packet delay, as discussed in Section 3.3.2 and evaluated in Fig. 3.8. More formally, for last-mile link capacity C, probe rate T_P, received traffic throughput T_{VNF} and the last-mile link traffic throughput λ, clusters will be found when $T_P + \lambda \geq C$.

Now, considering the last-mile link throughput to be equal to the received throughput (i.e., $\lambda = T_{\text{VNF}}$), the combined traffic will be equal to last-mile link capacity (i.e., $T_P + \lambda = C$) when probe throughput is set to,

$$T_P = C - T_{\text{VNF}} \tag{1}$$

Sending only one round of probing packets with the throughput calculated above may result in false estimation if the last-mile link throughput is more than the received traffic throughput (i.e., in case of fake traffic injection). To avoid this possibility of false estimation, two rounds of probing packets are sent. One at probe throughput T_{P1} and another at probe throughput T_{P2} as given in the following equations,

$$T_{P1} = C - T_{\text{VNF}} - \delta_T \tag{2}$$

$$T_{P2} = C - T_{\text{VNF}} \tag{3}$$

Here, the parameter δ_T is a small number that can be configured by the tenant admin. Computing this parameter automatically by using an efficient binary search approach will be an interesting future work. The number of clusters generated at probe throughput T_{P1} is denoted as N_{C1} and the number of clusters generated at probe throughput T_{P2} is denoted as N_{C2}.

Step 1.2: Received Traffic Capture. In this step, APPD first collects attributes of each packet (e.g., timestamp, size in bytes, etc.) by sniffing packets from the network interface, and then calculates IPD values from the timestamps.

Example 2 *As shown in Fig. 3.5, 1,000 probe response packets are generated and received at the first VNF having timestamps $P_1 \to t_{P1}$, $P_2 \to t_{P2}$, $P_3 \to t_{P3}$, ..., $P_{1000} \to t_{P1000}$. Now, the packet capture step at the first VNF will output the following to the next module: t_{P1}, t_{P2}, t_{P3}, ..., t_{P1000}. IPD values will then be calculated as follows: $D_1 = t_{P2} - t_{P1}$, $D_2 = t_{P3} - t_{P2}$, $D_3 = t_{P4} - t_{P3}$, ..., $D_{999} = t_{P1000 - t_{P999}}$.*

Step 1.3: Throughput Estimation (Clustering). This step is mainly responsible for two operations,

(i) Clustering the inter-packet delay (IPD) values: This step performs clustering on each data window. It uses data points comprising inter-packet delay and timestamp as input. As mentioned in Section 3.3.1, under the APPD effect, inter-packet delay values form clusters. We use the extended DBSCAN algorithm as the clustering algorithm and CityBlock distance metric. The use of the CityBlock distance metric allows us to select

only those clusters that are spread horizontally and have a very small height. For a real example of the clusters, see Fig. 3.8. After clustering, if the number of clusters is non-zero (i.e., at least one cluster is formed), then artificial congestion is confirmed for the current round of probing.

Example 3 *As shown in Fig. 3.4b, clustering algorithm on IPD values: $D_1 = t_{P2} - t_{P1}$, $D_2 = t_{P3} - t_{P2}$, $D_3 = t_{P4} - t_{P3}$, ..., $D_{999} = t_{P1000 - t_{P999}}$ finds zero clusters for the first round (i.e., $N_{C1} = 0$) and two clusters for the second round (i.e., $N_{C2} > 0$). Then artificial congestion is not confirmed for the first round but confirmed for the second round.*

(ii) Throughput estimation: This step estimates traffic throughput based on the clustering result. When artificial congestion is confirmed, received throughput is equal to the last-mile link throughput, that is, $\lambda = T_{\text{NFV}}$. Replacing this in Equation 1 and replacing λ with estimated throughput λ' we have,

$$\lambda' = C - T_P \tag{4}$$

Example 4 *Suppose, the clustering result is "$N_{C1} = 0$ and $N_{C2} > 0$", C = 1Gbps and T_{P2} = 500Mbps. Since artificial congestion is confirmed for the second round of probing, we have $\lambda' = C - T_{P2} = 1Gbps - 500Mbps$ = 500Mbps.*

Once λ' is known, an expected value of incoming traffic throughput is calculated by subtracting reportedly dropped or otherwise rerouted traffic throughput by each previous VNF. The information update process by which a VNF obtains dropped or otherwise rerouted traffic throughput of each previous VNF is described in the following paragraph. Now, for VNF_N, the expected incoming traffic throughput (T_E) is calculated as given in Equation 5.

$$T_E = \lambda' - \sum_{i=1}^{N-1} D_{\text{VNF}i} \tag{5}$$

Every VNF periodically provides updates (e.g., incoming traffic throughput and dropped/rerouted traffic throughput) to the next VNF(s) in the service chain. These information updates can be encrypted and digitally signed for confidentiality and integrity protection. As such updates comprise aggregate information of many packets (i.e., not on a per-packet basis), the use of encryption and digital signatures would not introduce additional communication overhead to tenants.

3.3.4 Stage 2: Verification of Service Chain Integrity

Depending on the position of the VNF in the service chain, integrity verification is done using one of the following two approaches (as shown in Fig. 3.6): (i) cluster-based verification, and (ii) throughput-based verification. The first approach is used only for the first VNF in the service chain whereas the second approach is used for the remaining VNFs in the service chain. These two steps are mainly concerned with three variables: (i) number of inter-packet delay (IPD) clusters for each round of probing, (ii) actual received traffic throughput, and (iii) expected traffic throughput. In the following, we detail these two approaches to integrity verification.

Step 2.1: Cluster-based Verification. For the first VNF in the service chain, verification is performed based on the probe round for which clusters were found in Step 1.1. This is because, firstly there are no preceding VNFs that can legitimately drop/reroute traffic. Therefore, estimating expected throughput (which is a way to take dropped/rerouted traffic into account) is not necessary. Secondly, if artificial congestion is not confirmed (i.e., no inter-packet delay clusters are formed for *Equation 4* to be valid) throughput estimation cannot be performed. In fact, knowing the number of clusters indirectly reveals incoming traffic throughput. The cluster-based integrity verification logic is given below,

$$\text{Integrity} = \begin{cases} \text{Normal}, & \text{if } N_{C1} = 0 \text{ and } N_{C2} > 0 \\ \text{Drop/Bypass}, & \text{if } N_{C1} > 0 \\ \text{Injection}, & \text{if } N_{C2} = 0 \end{cases}$$

Step 2.2: Throughput-based Verification. For the remaining VNFs in the service chain, the expected throughput (T_E) is compared with actual received traffic at the VNF (T_{VNF}) to verify the service chain integrity and classify the result of the verification according to the detection logic, which is shown in the equation below,

$$\text{Integrity} = \begin{cases} \text{Normal}, & \text{if } T_E = T_{VNF} \\ \text{Drop/Bypass}, & \text{if } T_E > T_{VNF} \\ \text{Injection}, & \text{othewise} \end{cases}$$

The rationale behind using expected traffic throughput (T_E) for all other VNFs (except the first) in the service chain is there are preceding VNFs that may legitimately drop/re-route traffic. Since (T_E) is calculated using the dropped/re-routed traffic information, using (T_E) will give accurate verification results even when some traffic is legitimately dropped/re-routed by preceding VNFs as part of those preceding VNF's functionality.

Example 5 *Suppose, $\lambda' = 500Mbps$, at VNF 2, T_{VNF2} = 400Mbps, and VNF 1 reported no packet drop/rerouting. Then, $T_E = \lambda' - D_{VNF1} = 500Mbps$. So, the condition $T_E = T_{VNF}$ is not satisfied implying integrity violation. Since, in this case, $T_E > T_{VNF}$, the detected integrity violation is classified as "Drop/Bypass".*

It is noteworthy that, the estimated incoming traffic throughput may differ from the actual throughput by δ_T. This means the expected throughput may also differ from the

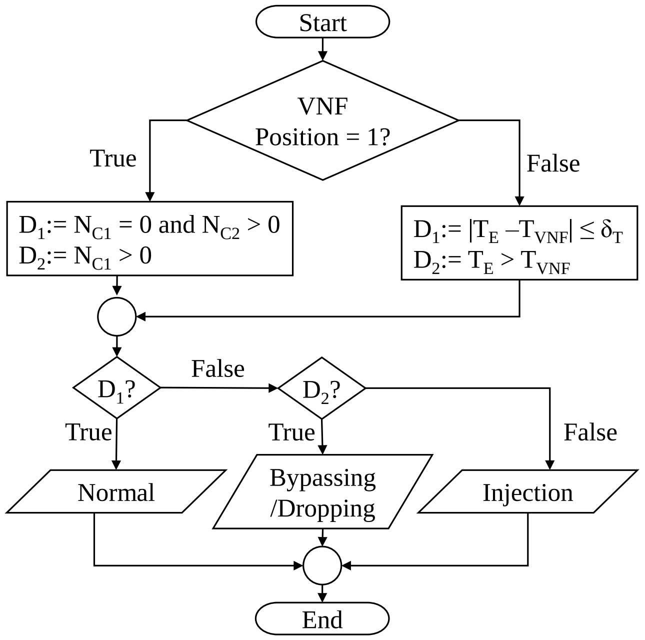


Figure 3.6: Integrity verification logic

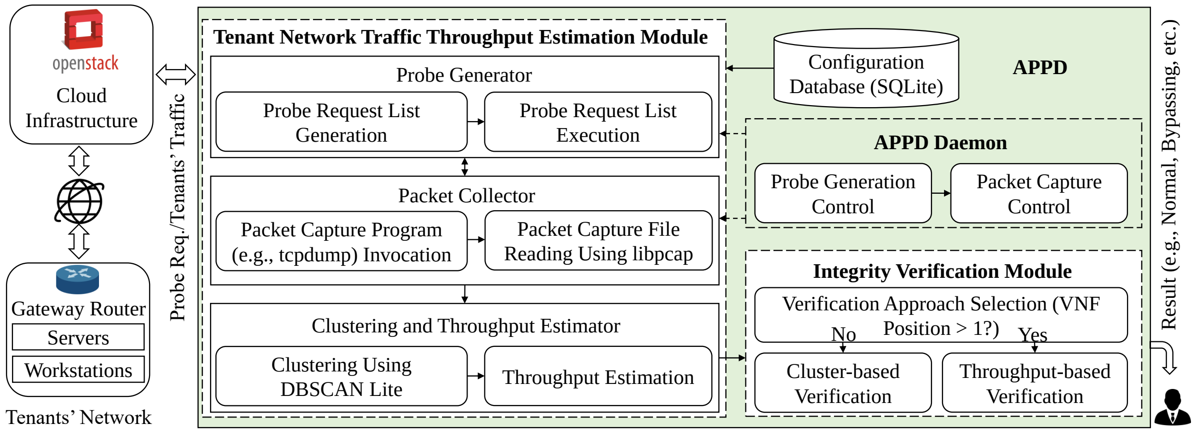

openstack
Cloud Infrastructure
Gateway Router
Servers
Workstations
Tenants' Network
Probe Req./Tenants' Traffic
Tenant Network Traffic Throughput Estimation Module
Probe Generator
Probe Request List Generation
Probe Request List Execution
Packet Collector
Packet Capture Program (e.g., tcpdump) Invocation
Packet Capture File Reading Using libpcap
Clustering and Throughput Estimator
Clustering Using DBSCAN Lite
Throughput Estimation
Configuration Database (SQLite)
APPD
APPD Daemon
Probe Generation Control
Packet Capture Control
Integrity Verification Module
Verification Approach Selection (VNF Position > 1?)
No
Yes
Cluster-based Verification
Throughput-based Verification
Result (e.g., Normal, Bypassing, etc.)

Figure 3.7: The architecture of APPD

actual throughput by δ_T.

3.4 Implementation

This section presents the implementation of APPD.

APPD Architecture. There are four major components of APPD (Fig. 3.7): (i) the APPD daemon for orchestrating the other modules, (ii) the incoming traffic throughput estimation module for estimating incoming traffic throughput, (iii) the integrity verification module for conducting service chain integrity verification, and (iv) the configuration database for storing parameters (e.g., δ_T) for different modules of APPD. The incoming traffic throughput estimation module is further divided into three components. (i) the probe generator is periodically started by the APPD daemon to send probing request packets, (ii) the packet collector is periodically started by the APPD daemon to capture received traffic at the VNF, and (iii) the clustering and throughput estimator performs clustering on inter-packet delay (IPD) data and estimates incoming traffic throughput.

Implementation Details. We implement APPD as a Linux service using systemd [66]. We chose Linux because it is the most popular operating system in the cloud [67]. However, APPD can also be deployed in any other operating system following a similar architecture as described in this section. APPD is deployed on each VNF, started as soon as the VNF operating system (OS) is booted, and continues to run as long as the VNF OS is running. All of the modules of APPD are developed using C programming language. The packet collector invokes a packet capture program (i.e., the tcpdump command-line packet analyzer [68]), prepares input for the clustering and throughput estimator by reading the capture file (.pcap) generated by the packet capture program, starts the clustering and throughput estimator, and receives update messages from previous VNF. An in-memory storage is used to pass captured packets from tcpdump to the packet collector. Clustering is done using DBSCAN Lite (our extended version of DBSCAN algorithm, which is detailed below). We use SQLite [69], a fast database engine, to implement the configuration database.

Extending DBSCAN. We extended the DBSCAN algorithm to guarantee fast execution time. We call the resulting algorithm DBSCAN Lite. To that end, our main extensions are: (i) reducing search space by sorting and axis trimming, (ii) reducing search space by dividing data into blocks, and (iii) reducing the number of searches by using a convex hull.

3.5 Experiments

This section presents our experimental results.

3.5.1 Overview of Experiments

As APPD is the first blackbox auditing solution to verify the forwarding integrity of service chains, a quantitative comparison between our work and other existing works is infeasible. Therefore, we evaluate APPD in terms of its ability to correctly verify experimental scenarios, the effectiveness of its clustering algorithm, and the overhead caused by probing.

3.5.2 Experimental Settings

To conduct our experiments, we build our NFV testbed using Tacker [70] and OpenStack [18], where OpenStack is a very popular infrastructure-as-a-service (IaaS) software and Tacker is an official OpenStack [27] project that provides a VNF Manager (VNFM) and an NFV Orchestrator (NFVO) that can be used to deploy and manage VNFs. Our testbed includes one controller node and up to 80 compute nodes, each with 8 CPUs and 12 GB RAM running Ubuntu 20.04 server. We have used Mininet-2.3.0 [71] to set up the tenant network and Internet links (between tenant network and NFV) with virtual hosts, virtual links, and Open vSwitch (OVS) [72] virtual switches on a dedicated server. To connect the tenant network to the service chains, the server where the tenant network is set up is then connected to the NFV testbed using a 10Gbps local area network (LAN). Also, similar to real ISP, we set up a traffic shaper to limit the bandwidth (to 1Gbps) from the tenant network to NFV using the Linux traffic control module NetEm [73]. We also set up 10 virtual hosts inside the tenant network and 10 additional virtual hosts connected to the Internet switches. The virtual hosts either act as video servers (using ffserver [74]) or video

Table 3.1: Applying APPD in real network setting shows that it could correctly verify all the experimental scenarios

Exp No.	**Experimental Integrity Scenario**	**VNF Position**	**Actual Received Throughput (T_{VNF})**[1]	**Estimated Tenant Throughput (λ')**[1]	**Probe Throughput (T_{P1})**[1]	**Probe Throughput (T_{P2})**[1]	**No. of IPD Clusters (N_{C1})**	**No. of IPD Clusters (N_{C2})**	**Dropped/Rerouted Throughput (T_D)**[1]	**Expected Throughput ($T_E = \lambda' - \sum T_D$)**[1]	**APPD Result**
1	*Normal* (first VNF)	= 1	500	-	440	500	0	2	-	-	$N_{C1} = 0$ and $N_{C2} > 0$: *Normal*
2	*Bypass/Drop* (first VNF)	= 1	400	-	540	600	2	2	-	-	$N_{C1} > 0$: *Bypass/Drop*
3	*Injection* (first VNF)	= 1	600	-	340	400	0	0	-	-	$N_{C2} = 0$: *Injection*
4	*Normal* (second VNF)[2]	>1	400	500	-	-	-	-	100	400	$\|T_E - T_{\text{VNF}}\| \leq \delta_T$: *Normal*
5	*Bypass/Drop* (second VNF)[2]	>1	300	500	-	-	-	-	100	400	$T_E > T_{\text{VNF}} + \delta_T$: *Bypass/Drop*
6	*Injection* (second VNF)[2]	>1	500	500	-	-	-	-	100	400	$T_E < T_{\text{VNF}} - \delta_T$: *Injection*

[1] In Mbps.
[2] Second VNF is representative of any VNF in the service chain except the first VNF (i.e., VNF Position >1).

clients (using MPlayer [75]). On one hand, to generate tenant network traffic, hosts inside the tenant network act as video clients to stream video from video servers in the Internet. On the other hand, to generate cross-traffic [76], hosts outside tenant network act as video clients to stream video from video servers on the Internet.

3.5.3 Experimental Results

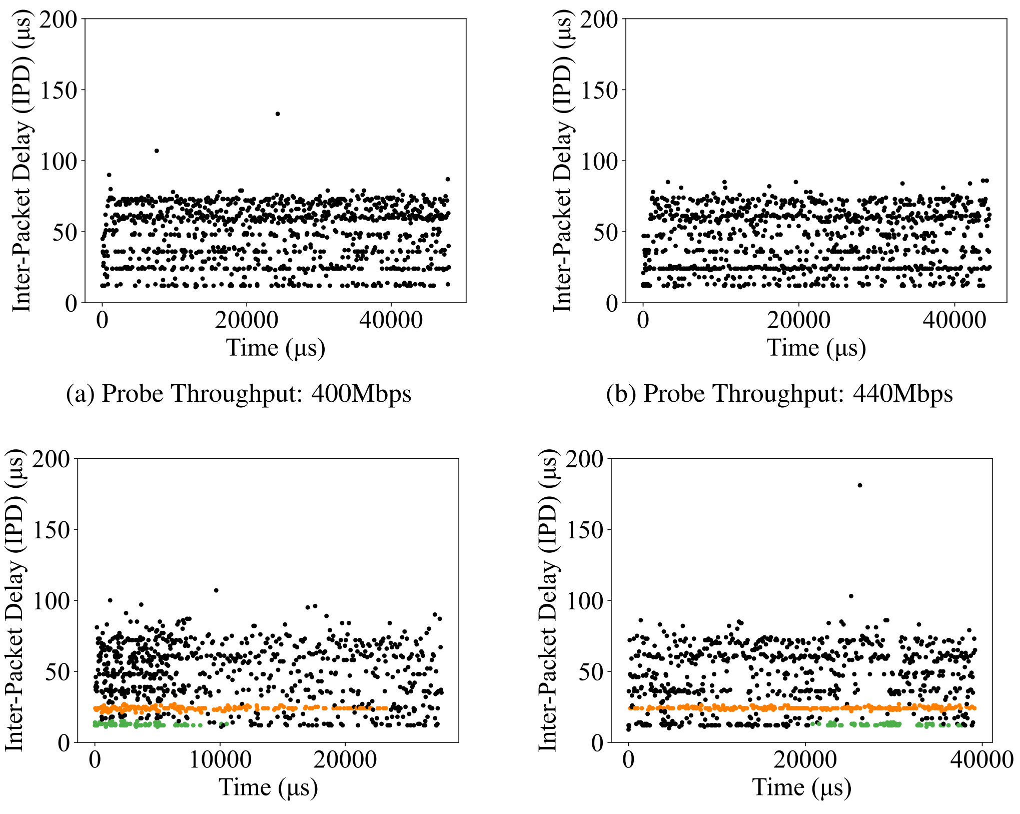
We present our experimental results to evaluate the effectiveness and overhead of APPD as follows.

Effectiveness in Verifying Service Chain Integrity. Table 3.1 demonstrates the effectiveness of APPD through six different scenarios (including different attacks such as bypass, drop, injection, as well as normal behavior at different VNFs) where APPD could correctly detect all existing breaches. We emulate the attacks by modifying the flow rules of the SDN switches. This table reports the results corresponding to the first two VNFs. The first VNF is shown to demonstrate cluster-based verification, whereas the second VNF illustrates throughput-based verification. Any remaining VNFs in the service chain can perform integrity verification in the same way (i.e., throughput-based) as the second VNF; however, we do not report their result for the sake of brevity. In the following, we explain the scenarios listed in Table 3.1. In these scenarios, the tenant is sending traffic at a throughput $\lambda = 500$Mbps at the time of verification, capacity $C = 1$Gbps, and $\delta_T = 60$Mbps. APPD is running on each VNF of the service chain `FW-IDS-...-VNF N` that is receiving the tenants' traffic. The first three scenarios are for the first VNF (i.e., `FW`) whereas the last three scenarios are for the second VNF (i.e., `IDS`).

- *First scenario:* The actual throughput received at `FW` is $T_{VNF} = 500$Mbps. Two rounds of probe requests are sent from `FW`, the first having response throughput $T_{P1} = C - T_{VNF} - \delta_T = 440$Mbps and the second having response throughput $T_{P2} = C - T_{VNF} = 500$Mbps. Now, since $\lambda = 500$Mbps, at $T_{P1} = 440$Mbps the clustering algorithm doesn't find any clusters in the IPD values, as shown in Fig. 3.8b resulting in $N_{C1} = 0$. However, at $T_{P2} = 500$Mbps, two clusters are found as shown in Fig. 3.8d, giving $N_{C2} = 2$. Since $N_{C1} = 0$ and $N_{C2} > 0$, at the verification phase (using cluster-based verification) it is confirmed that the scenario is *Normal* and the estimated incoming throughput (λ') is calculated to be $C - T_{VNF} = 500$Mbps.

- *Second scenario:* Traffic is bypassing/dropping `FW` and the actual throughput received at `FW` is $T_{VNF} = 400$Mbps. Then, first probe requests are sent having response throughput $T_{P1} = C - T_{VNF} - \delta_T = 540$Mbps and the clustering algorithm finds two clusters. Since $N_{C1} > 0$, *Bypass/Drop* is detected.

- *Third scenario:* Similar to above.



(a) Probe Throughput: 400Mbps (b) Probe Throughput: 440Mbps

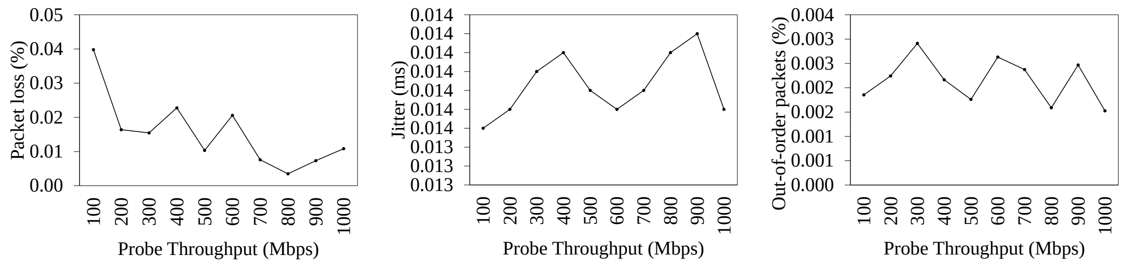
(c) Probe Throughput: 450Mbps (d) Probe Throughput: 500Mbps

Figure 3.8: Applying DBSCAN clustering algorithm on inter-packet delay (IPD); Noise (●), Cluster 1 (🟢), Cluster 2 (🟠)

- *Fourth scenario:* Received throughput is $T_{VNF} = 400$Mbps. `IDS` is updated by `FW` that $\lambda' = 500$Mbps and dropped traffic by `FW` throughput is $T_D = 100$Mbps. So, the IDS calculates its expected traffic throughput $E = 400$Mbps. Since $T_E - T_{VNF} = 0 \leq 60$, the scenario is detected as Normal.

- *Fifth and sixth scenarios:* Similar to fourth scenario.

Effectiveness of Probing and IPD Clustering. In Fig. 3.8, we demonstrate the IPD clustering results for different probe throughputs in a *Normal* scenario (i.e., no integrity breaches). Here the tenants' last-mile link capacity and throughput are 1Gbps and 500Mbps, respectively. For lower probing throughputs: 400Mbps (Fig. 3.8a) and 440Mbps (Fig. 3.8b)



(a) Packet loss (b) Jitter (c) Out-of-order packets

Figure 3.9: Measuring APPD overhead in terms of network performance metric (packet loss, jitter, out-of-order packets)

no cluster is formed, and for higher probing throughputs: 450Mbps (Fig. 3.8c) and 500Mbps (Fig. 3.8d) two clusters (as indicated in orange and green) are formed. Here the transition from no clusters to two clusters happens between probing throughput 440Mbps and probing throughput 450Mbps. Therefore, APPD expects no clusters in a *Normal* scenario for its first round of probing ($T_{P1} = 440$Mbps), as calculated from Equation 2 in Section 3.3. Similarly, APPD expects one or more clusters for its second round of probing ($T_{P2} = 500$Mbps), as calculated from Equation 2 in Section 3.3.

Overhead. We evaluate the overhead of APPD in terms of impact on different network performance metrics (e.g., packet loss, jitter, and packet reordering). To do so, we measure these metrics while performing tenant network throughput estimation at different possible probe rates. To measure these metrics, we capture packets (at both video clients and video servers) and perform calculations on these packets by identifying the same packets using Transmission Control Protocol (TCP) sequence numbers. The results of these experiments are shown in Fig. 3.9 where we can see that there is no correlation between these performance metrics and the APPD probe throughput (e.g., packet loss does not show an upward trend as probe throughput increases). Thus, it is evident that APPD has a negligible impact on network performance.

Effect of Network Sensitivity. We evaluate the effect of dynamic changes in the network

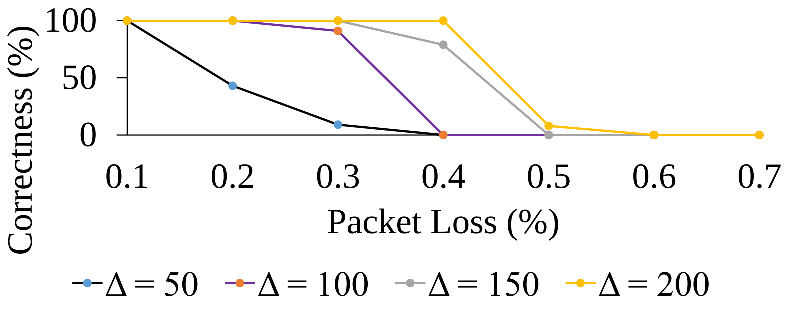


Figure 3.10: Sensitivity of APPD to packet loss in the network

(e.g., packet loss, delay, and packet reordering) on the ability of APPD to correctly verify integrity. For this purpose, we utilize the experimental integrity scenario that we created for verification of effectiveness with varying levels of dynamic changes in the network. We then observe how many (out of the total six scenarios) are correctly verified by APPD and calculate a percentage score. We also vary the parameter Δ to see if it can improve the percentage of correctly verified scenarios. As we can see in Fig. 3.10, although APPD can correctly verify integrity under a low packet loss, an increase in packet loss quickly deteriorates its ability to perform verification correctly. However, by increasing the value of Δ, APPD can still perform verification correctly under a higher packet loss rate. We do not show the effect of delay and jitter here as we did not notice any significant effect of those metrics on the ability of APPD to perform verification correctly.

3.6 Conclusion

This chapter proposed a blackbox approach, namely, APPD, to verify service chain integrity in NFV without requiring any access to the infrastructure-level data or resources. Additionally, APPD can verify integrity breaches resulting from a wider range of attacks in comparison to the existing works. To that end, APPD first created an artificial packet-pair dispersion among the incoming traffic to NFV using probing packets. Second, APPD estimated tenant network traffic throughput from inter-packet delay that is caused by the artificial packet-pair dispersion. Finally, APPD verified different types of integrity breaches by comparing the estimated throughput with the actual traffic throughput observed in any VNF in a service chain. Experimental results in a real network environment showed that our approach can effectively verify service chain integrity for a wide range of integrity breaches and have negligible impact on network performance. As future work, we plan to automate setting the parameters of the clustering algorithm and further optimize other parameters of APPD. Furthermore, we plan to perform extensive security analysis and more experimental evaluations of APPD in future.

Chapter 4

Functional Integrity Verification for 5G Cloud-Native Network Functions

4.1 Introduction

The cloudification of network functions (NFs) in 5G networks is recently empowered by cloud-native technologies (e.g., containers) to ensure high-speed, cost-efficient, and large-scale connectivity [77]. However, running these software as Cloud-Native Network Functions (CNFs), where the NFs share resources (e.g., operating system kernel) with potentially malicious collocated tenants, increase the risk of many threats including code injection attack that might lead to malicious modifications of the in-memory instructions of the NFs (a.k.a. breach of functional integrity) at runtime [16],[17]. For instance, such vulnerabilities have been reported in open-source implementations of 5G core network software (e.g., Open5GS [78]). Additionally, some reused libraries and software components have been shown to be vulnerable to code injection (e.g., Log4j [79] and CVE-2022-28391 [80]). For example, our scanning of the Open5GS [78] project using Trivy [81], an open-source security scanner, reveals numerous bugs (1,309 vulnerabilities, where 41 of those vulnerabilities are critical and 317 vulnerabilities are highly severe). Considering the vital role of 5G in various cyber critical infrastructure and their security, tackling malicious code injection and verifying the functional integrity of NFs are very important for the security of 5G networks.

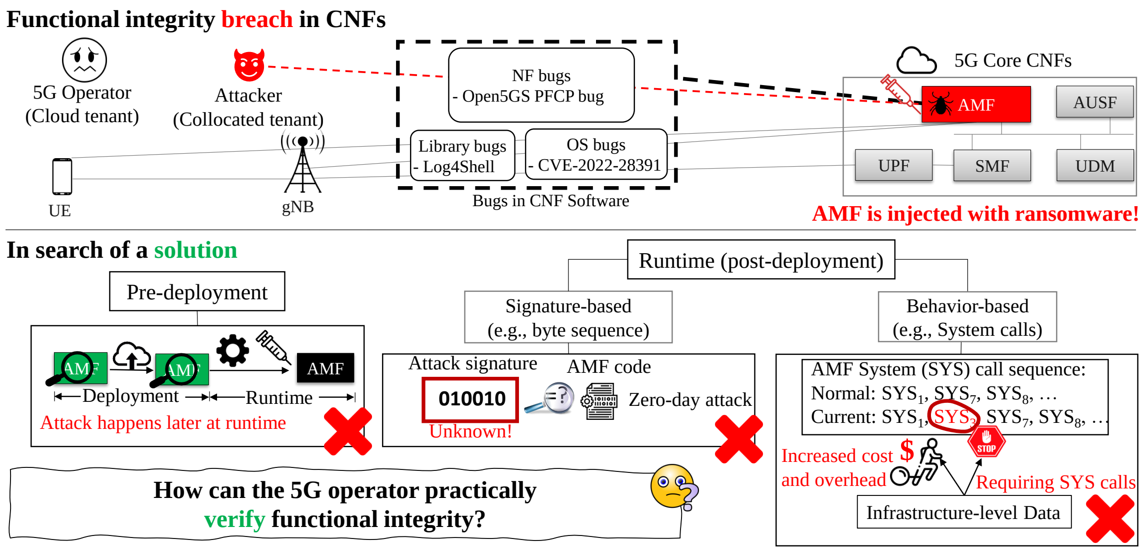
However, two major approaches (i.e., pre-deployment and post-deployment) to verifying functional integrity are usually insufficient in the context of CNFs in 5G mainly due to the following reasons: i) *Pre-deployment:* The verification approaches (e.g., [46, 47]) that are conducted before the deployment of the network functions in the cloud cannot detect the integrity breaches that are caused by the malicious code injected at runtime (i.e., after the deployment). (ii) *Post-deployment:* The verification approaches (e.g., [29, 30]) that are conducted after the deployment of the network functions require infrastructure-level data (which is usually inaccessible to 5G operator [82]). On the other hand, relying on an infrastructure-level runtime solution (e.g., using system-call interception) to verify such functional integrity may add significant latency [31, 32] to the performance sensitivity of 5G applications. We further illustrate these limitations and our main ideas through a motivating example as follows.

4.1.1 Motivating Example

The top of Fig. 4.1 illustrates a practical scenario of functional integrity breaches in a possible 5G core implementation, and the bottom depicts the limitations of existing verification solutions.

Functional Integrity Breach. By exploiting the existing vulnerabilities, e.g., Open5GS PFCP[78], an attacker, e.g., collocated tenant ([16, 17]) can inject code into some NFs. Particularly, as shown in the figure, the attacker injects malicious code into the in-memory NF, *Access and Mobility Management Function* (`AMF`), to modify its functionality.

Limitations of Existing Solutions. The existing verification approaches could be divided into two major categories: 1) the bottom left illustrates the pre-deployment approaches and



Functional integrity breach in CNFs
5G Operator (Cloud tenant)
Attacker (Collocated tenant)
UE
gNB
NF bugs - Open5GS PFCP bug
Library bugs - Log4Shell
OS bugs - CVE-2022-28391
Bugs in CNF Software
5G Core CNFs
AMF
AUSF
UPF
SMF
UDM
AMF is injected with ransomware!
In search of a solution
Pre-deployment
Runtime (post-deployment)
Signature-based (e.g., byte sequence)
Behavior-based (e.g., System calls)
AMF
Deployment
Runtime
Attack happens later at runtime
Attack signature
AMF code
010010
Unknown!
Zero-day attack
AMF System (SYS) call sequence:
Normal: SYS1, SYS7, SYS8, …
Current: SYS1, SYS3, SYS7, SYS8, …
Increased cost and overhead
Requiring SYS calls
Infrastructure-level Data
How can the 5G operator practically verify functional integrity?

Figure 4.1: An example of functional integrity breach in a possible CNF implementation of 5G core (top) and limitations of existing solutions (bottom)

2) the bottom right depicts the post-deployment approaches. A pre-deployment approach verifies the integrity of AMF before deployment either at the operator level or at the cloud provider level. Therefore, such an approach cannot detect this integrity breach where AMF is modified at the runtime. On the other hand, a post-deployment approach performs either signature-based or behavior-based verification. Signature-based verification (e.g., [83]) relies on the attack signature of the malicious code snippet and checks the binary code of AMF to find a match. On the other hand, behavior-based verification matches the current system call sequence [45] against the normal sequence of system calls to identify that $\texttt{SYS}_3$ is a mismatch and causing the integrity breach in AMF. However, this verification method requires access to the infrastructure-level data which makes the approach costly (due to the instrumentation) and inefficient (due to the frequent system-level interception). Moreover, the adoption of CNF limits the capacity of 5G operators in accessing provider-level data [82], and hence these post-deployment approaches also become infeasible for the 5G operator.

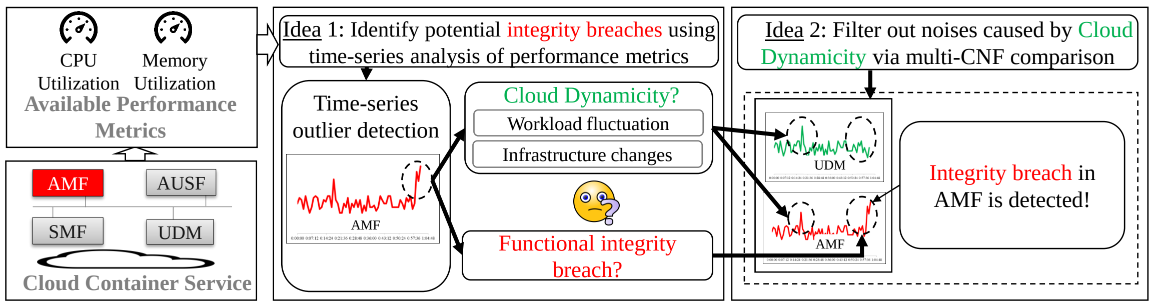

CPU Utilization
Memory Utilization
Available Performance Metrics
AMF
AUSF
SMF
UDM
Cloud Container Service
Idea 1: Identify potential integrity breaches using time-series analysis of performance metrics
Time-series outlier detection
AMF
Cloud Dynamicity?
Workload fluctuation
Infrastructure changes
Functional integrity breach?
Idea 2: Filter out noises caused by Cloud Dynamicity via multi-CNF comparison
UDM
AMF
Integrity breach in AMF is detected!

Figure 4.2: The main ideas of 5GFIVer

4.1.2 Main Ideas and Contributions

To overcome those limitations in the existing solutions, our main ideas (Fig. 4.2) are as follows.

Idea 1: Performance Metric as a Side-channel. As depicted in the middle of Fig. 4.2, our first idea is inspired by the fact that any change in software functionality (i.e., caused by injected code) may affect the resource (e.g., CPU, memory, power, etc.) consumption [84] in a distinctive way. Particularly, our idea is to perform a time-series analysis of the available performance metrics (e.g., CPU and memory utilization of each container) to identify outliers. The key advantages of this idea are: 1) our approach makes the decision based on the available performance metrics (e.g., CPU and memory utilization of AMF, UDM, AUSF, SMF as shown in Fig. 4.2), and hence, does not need any access to the infrastructure level data, 2) these performance metrics are made available by public cloud providers (e.g., Amazon AWS, Microsoft Azure, IBM cloud, etc.) for billing purposes [85], and hence, our proposed approach does not require any instrumentation (i.e., no modification of 5G core NF) to collect data, and 3) the implementation of time-series analysis in detecting outliers provides us the opportunity to detect breaches at runtime. A key challenge here is that, apart from the attacks mentioned earlier, the dynamic behavior of the cloud infrastructure (e.g., workload variation, infrastructural changes, etc.) may also affect CPU and memory consumption. Therefore, relying only on *Idea 1* would result in false-positive decisions;

which motivates us to propose our second idea to address such concerns.

Idea 2: Multi-CNF Correlation. Our second idea is to correlate the findings of time-series analysis of the containers whose resource consumptions are correlated with each other (detailed in Section 4.2) in order to verify the decisions made by *Idea 1*. The containers of the 5G core NFs are considered correlated with each other when they tend to respond in a similar way to any legitimate changes in their performance metrics (e.g., due to underlying infrastructure change or workload fluctuation). Therefore, the first step of *Idea 2* is to identify the correlated containers and then to implement a time-series analysis for each container to identify respective outliers. If an outlier is identified for a particular container, we correlate this finding with its correlated containers which are expected to show similar outliers, while for the attack, only the infected container may show an outlier. Consequently, *Idea 2* can potentially help to reduce false positives while keeping all advantages of *Idea 1*. We will elaborate on these ideas in Section 4.3.

Main Contributions. The main contributions of this work are:

- We propose an operator-oriented black-box approach to verify functional integrity in 5G core implementation without relying on the provider-level data.
- To improve the accuracy and minimize the false positives, we implement time-series analysis on the available performance metrics (e.g., memory and CPU consumption by each container) to detect outliers and verify whether those are due to attacks or not, through the correlation between containers.
- Our proposed approach does not require any instrumentation to collect required data, i.e., no change is required to the 5G core during the data collection process. This makes our solution more deployable and practical.
- We implement our solution and integrate it with Open5GS [86], a popular open-source 5G core implementation, under our testbed, and our experimental evaluation

shows the robustness of our solution.

The rest of this chapter is organized as follows. Section 4.2 provides the background on 5G NFs and defines our threat model. Section 4.3 describes the proposed solution. Section 4.4 presents our implementation details along with the experimental results. Finally, Section 4.6 concludes the chapter.

4.2 Preliminaries

This section discusses 5G core cloudification, describes the correlation among the containers, and then defines our threat model and assumptions.

Cloudification of 5G Core. 5G core is an essential component of 5G networks that orchestrates various services such as authentication, authorization, session management, routing and switching, data and policy management, and maintaining connectivity with the client. These functionalities are completely virtualized on the cloud deployment and are called CNFs. As shown in Fig. 4.3, the 5G core is composed of a plethora of CNFs including but not limited to Access and Mobility Management Function (AMF), User Plane Function (UPF), Session Management Function (SMF), and Unified Data Management (UDM). All of these functions serve diverse roles, say, AMF is responsible for letting user equipment (UE), e.g., mobile phones, connect to the core network. 5G core can function only when these CNFs interact. As a result of this interaction, as shown in Fig. 4.4a, a new task in one CNF (e.g., processing *Initial UE Message* in AMF) also leads to a new task for another

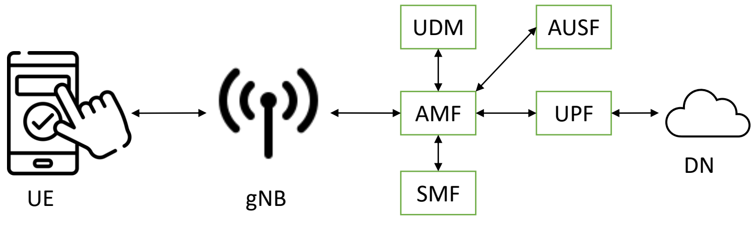
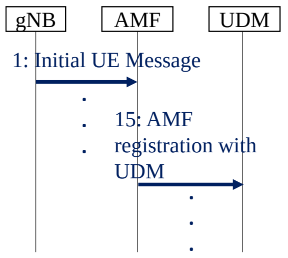
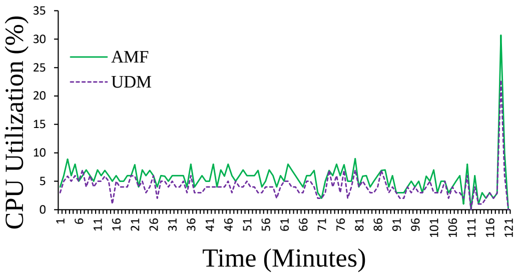


Figure 4.3: An excerpt of 5G topology [1]



(a) An example of AMF call flow in 5G [87]



(b) CPU utilization of AMF and UDM

Figure 4.4: An example of a metric (CPU utilization), that shows to be highly correlated for AMF and UDM.

CNF (e.g., *AMF registration* for UDM). Since certain CNFs share the related responsibility, it is likely that their performance metrics may also be correlated (discussed below).

The Correlation Among 5G CNFs. Our experiments indicate that performance metrics (e.g., CPU and memory utilization) of certain CNFs (e.g., AMF) show a strong correlation with other specific CNFs (e.g., UDM). For example, as shown in Fig. 4.4b, CPU utilization of AMF (shown in solid green) shows a similar pattern as CPU utilization of UDM (shown in dashed purple).

Another way to interpret the above phenomenon is that any "unusual" behavior shown in the performance metrics due to the cloud dynamicity (e.g., change in workload or underlying infrastructure) is likely to affect multiple CNFs instead of only one (i.e., it will affect all the CNFs that are correlated). We can leverage this observation to search for evidence of cloud dynamicity and thus filter them out to avoid false positives. Although the correlation in Fig. 4.4b is easy to visually identify in this particular case, it may not be the case in the presence of many CNFs and a large volume of data. To this end, we use correlation analysis to identify highly correlated CNFs, which will be detailed in Section 4.3.

Threat Model and Assumptions. This work considers integrity breaches of containerized network functions (CNFs) caused by a code injected into the CNF by exploiting vulnerabilities [88, 89] in: (i) CNF software (e.g., Open5GS PFCP bug [78]), (ii) libraries used by the CNF (e.g., Log4j [79]), or (iii) underlying host operating systems(e.g., CVE-2022-28391 [80]). This code injection happens at runtime in-memory after the CNF is loaded from its image and does not modify the latter. Once compromised, the CNF cannot be trusted any longer (e.g., for collecting logs via SSH) because it may be under the control of the attacker. In addition, the code injection may be either (i) *transient:* when the code modification is not permanent and only lasts for the duration of the processing of the current packet/request, or (ii) *permanent:* when the code modification is permanent and lasts even after the packet/request is completed/processed. We assume that as the vulnerabilities in each CNF are unique, it will be difficult for the attacker to compromise all the CNFs that are correlated with each other. We do not assume that the attack signature is known (i.e., possibly for a zero-day attack). Also, we do not assume that the tenant has access to the underlying infrastructure to collect information (e.g., logs) apart from what is commonly available to users of cloud container services. On the other hand, the out-of-scope threats for this chapter include verifying integrity breaches due to those attacks that: i) compromise all CNFs, ii) similarly affect the correlated CNFs, and iii) do not have much impact on CPU/memory consumption, which will be considered in our future work.

4.3 Functional Integrity Verification for 5G

This section presents the methodology of 5GFIVer in detail.

4.3.1 Approach Overview

Fig. 4.5 shows an overview of 5GFIVer, which contains two stages. First, in *Stage 1*, 5GFIVer performs a time series analysis of the performance metrics (e.g., CPU, memory, etc.) to detect the outliers as an indication of potential attacks. Specifically, in this stage, 5GFIVer first collects and processes the available performance metrics for each container.

Then, it deploys an unsupervised learning technique (e.g., level shift detection [90]) to perform a time series analysis of the collected data to detect outliers for each container (e.g., AMF). In *Stage 2*, 5GFIVer filters out the outliers caused by the cloud dynamicity to identify integrity breaches. Specifically, in this stage, 5GFIVer eliminates false-positive decisions made in *Stage 1* by using the correlated behavior among multiple CNFs (e.g., UDM is correlated with AMF) and then identifies integrity breaches accordingly. We detail each stage as follows.

4.3.2 Stage 1: Outliers Detection

This first stage of 5GFIVer performs: (Step 1.1) performance metrics collection, and (Step 1.2) time series analysis.

Step 1.1: Performance Metrics Collection. This step is to collect available performance metrics for the CNFs to be analyzed for detecting potential integrity breaches. According to our investigation, different performance metrics (e.g., CPU or memory utilization) from different public cloud container services are available to the 5G operator [85]. Our experiments with those available performance metrics show that these metrics are affected by the

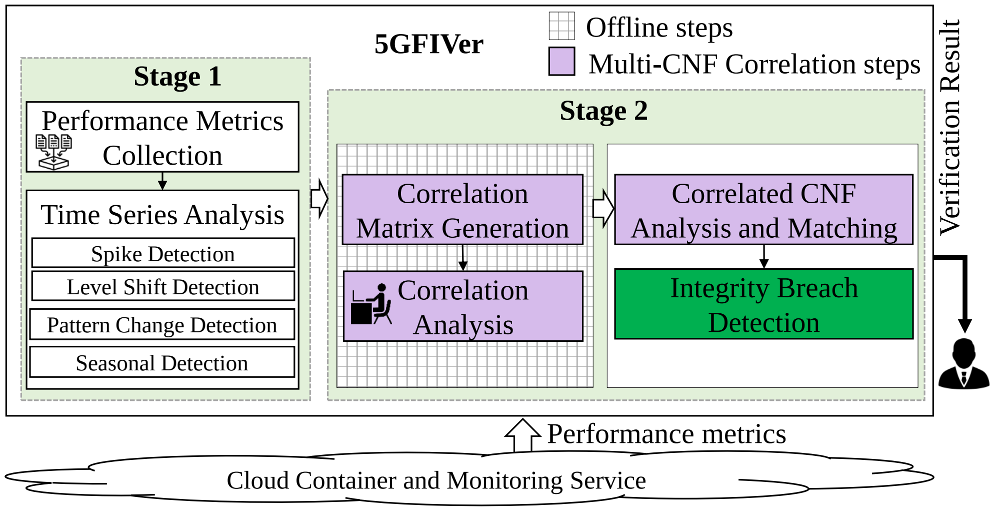
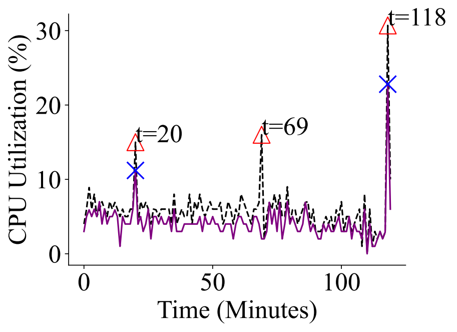
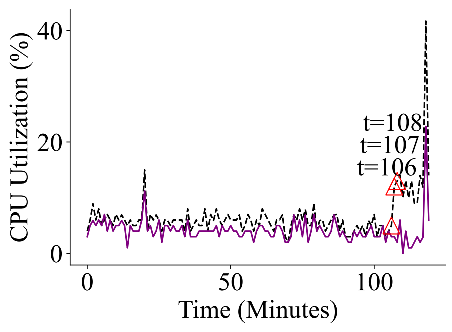


Figure 4.5: A high-level overview of 5GFIVer



(a) Transient code injection



(b) Permanent code injection

Figure 4.6: 5GFIVer time series analysis; AMF (---), UDM (—), analysis of AMF (△), analysis of UDM (×)

code injection attack, and hence can be utilized in our solution to detect such breaches. For instance, we collect performance metrics (e.g., CPU utilization) periodically (e.g., at every minute) during the operation of 5GFIVer as shown in Fig. 4.6. Hence, the outcome of this step can be represented as a stream of timestamp-metric pairs for time t_1 to t_n as follows: t_1→M$_{t_1}$, t_2→M$_{t_2}$, t_3→M$_{t_3}$, ..., t_n→M$_{t_n}$.

Step 1.2: Time Series Analysis. This step is to identify outliers in the stream of timestamp-metric pairs collected in *Step 1.1*. Fig. 4.5 lists a number of possible types of time series outlier detection algorithms. In this chapter, we demonstrate two of them to detect two types of outliers: 1) detecting spikes caused by the transient code injection and 2) detecting level shift caused by the permanent code injection.

The spike (a.k.a. additive outlier [91]) is a sudden change in the performance metrics for a short span of time. This can be detected by considering three consecutive disjoint sliding windows and tracking the difference in the median value between the short central window and the other two outer windows [91]. For instance, Fig. 4.6 (a) depicts the spikes as outliers in CPU consumption of AMF at $t = 20$, $t = 69$, and $t = 118$. Hence, the outcome of this spike detector for each CNF is a set of timestamps: T_S(CNF) $= \{t_{S_1}, t_{S_2}, t_{S_3}, ..., t_{S_n}\}$, where t_{S_n} indicates the time for a spike S_n.

On the other hand, the level shift [90] indicates a shift in the level of the performance metrics that might occur due to a permanent code injection. To determine this level shift, we leverage the LevelShiftAD detector from the anomaly detection toolkit by Arundo [92], which detects a shift in the metrics value level by considering the difference in median value between two adjacent sliding windows. It needs to be mentioned that LevelShiftAD is not spike sensitive [92], and hence any spike generated by transient injection is not detected by this detector. Fig. 4.6 (b) shows that a level shift occurs at $t = 106$ in the CPU consumption of AMF, while no shift is found for UDM. The outcome of the level shift detector is another set of timestamps: $T_L(\text{CNF}) = \{t_{L_1}, t_{L_2}, t_{L_3}, ..., t_{L_n}\}$, where t_{L_n} indicates the time for a level shift L_n.

However, the key challenge in outlier detection is that these outliers (i.e., spike or level shift) can occur either due to code injection or caused by legitimate changes in the cloud infrastructure. Hence, we verify the detected outliers in *Stage 2* to eliminate false positive decisions.

4.3.3 Stage 2: Integrity Breach Detection

The second stage performs: (Step 2.1) multi-CNF correlation, and (Step 2.2) integrity breach detection.

Step 2.1: Multi-CNF Correlation. This step (marked in the light purple background in Fig. 4.5) is to identify outliers (both spikes and level shifts) caused by cloud dynamics but detected by *Stage 1*. The inputs to this step are a CNF under verification (e.g., CNF1), its correlated CNF (e.g., CNF2), and the sets of timestamps for each of these CNFs from *Stage 1*. This step consists of (i) conducting an offline process to find which CNF metrics are correlated with each other, and (ii) conducting an online process to identify outliers caused by the cloud dynamicity.

The offline process generates a correlation matrix, i.e., the matrix containing the similarity in resource consumption patterns of different containers (e.g., AMF, UDM, AUSF, etc.). For instance, Fig. 4.6 shows that the CPU consumption, respectively, of AMF and UDM have similarities (evaluated in Fig. 4.7a) and hence, these two CNFs are correlated with each other. Hence, the outcome of this offline process is a matrix containing the information regarding the correlated CNFs (e.g., AMF is correlated with UDM or SMF is with UPF) which will be available to the online process. This outcome enables the next steps, for any CNF under verification (e.g., *CNF1*), to know its correlated CNF (e.g., *CNF2*).

The online process performs correlation to identify the common ones among the detected outliers (in *Stage 1*) of the correlated CNFs (i.e., *CNF1* and *CNF2*). The outcome of this step is the number of outliers that are in common in *CNF2* and in *CNF1*, for each type. As these CNFs have correlated metrics, the outliers in common can be related to cloud dynamicity issues as discussed in Section 4.2. Thus, the number outliers in common of type spike, denoted by N_{C_S}, and the number outliers in common of type level shift, denoted by N_{C_L}, can be computed as follows:

$$N_{C_S} = n(\{T_S(\text{CNF1}) \cap T_S(\text{CNF2})\}) \tag{6}$$

$$N_{C_L} = n(\{T_L(\text{CNF1}) \cap T_L(\text{CNF2})\}) \tag{7}$$

where $n(T)$ is the number of elements in a set T. For instance, in Fig. 4.6a (it is noteworthy that 4.6a and 4.6b are from different experiments and we do not show the corresponding spike detection for 4.6b due to lack of space), three spikes are detected in CPU consumption of AMF (i.e., *CNF1*) at $t = 20$, $t = 69$, and $t = 118$, while two spikes are detected for UDM (i.e., *CNF2*) at matched timestamps (i.e., at $t = 20$, and $t = 118$). The online process computes that $N_{C_S} = 2$. Similarly, as no level shift is detected in UDM in Fig. 4.6b, it computes $N_{C_L} = 0$.

Step 2.2: Integrity Breach Detection. Using the findings from the previous step, this step identifies integrity breaches and classifies them (i.e., transient or permanent).

The input to this step is the same pair of CNFs as in Step 2.1, the number of their respective identified outliers, and the number of outliers common between them. Let N_{CNF1} be the total number of identified outliers (in *Stage 1*) for *CNF1* computed using the following equation,

$$N_{CNF1} = N_{CNF1_S} + N_{CNF1_L} \tag{8}$$

Here, N_{CNF1_S} is the number of outliers detected by the spike detector, and N_{CNF1_L} is the number of outliers detected by the level shift detector in CPU utilization of CNF1 (e.g., AMF). Now a positive value of N_{CNF1} indicates a potential integrity breach, whereas, if N_{CNF1} is zero, it indicates that there are no integrity breaches. If $N_{CNF1} > 0$, we compute the total number of correlated outliers N_C as follows,

$$N_C = N_{C_S} + N_{C_L} \tag{9}$$

Then we define a parameter Δ as follows.

$$\Delta = N_{CNF1} - N_C \tag{10}$$

It should be noted that, according to our assumptions in Section 4.2, $N_{CNF1} \geq N_C$ and $N_C \geq 0$. At this point, a positive value of Δ indicates that there are integrity breaches in *CNF1*, and a zero value of Δ indicates that there is no integrity breach.

To further classify the type of code injection (i.e., transient or permanent), the value of spike or level shift can be consulted. Specifically, if $\Delta > 0$, the code is injected in *CNF1*. Now to classify the injected code, N_{CNF1_S} and N_{CNF1_L} can be utilized. Since transient and permanent code injections are exclusive in our threat model, a positive value of N_{CNF1_S} indicates transient code injection. On the other hand, if N_{CNF1_L} is positive, it indicates permanent code injection.

$$\text{Code Injection} = \begin{cases} \text{Transient}, & \text{if } \mathrm{N}_{CNF1_S} > 0 \\ \text{Permanent}, & \text{if } \mathrm{N}_{CNF1_L} > 0 \end{cases}$$

For instance, from Fig. 4.6b, we can see that the total number (both spikes and level shifts) of outliers in CNF1 (i.e., AMF) is $N_{CNF1} = 3$, while for the correlated CNF2 (i.e., UDM), $N_C = 0$. Hence, we have a value of $\Delta = 3$ and this indicates that there is an integrity breach in AMF. On the other hand, since $N_{CNF1_L} = 3 > 0$, the breach is caused by a permanent injection in AMF.

4.4 Implementation

This section presents the implementation details and experimental evaluation of 5GFIVer. We implement 5GFIVer as a Linux service using systemd [66] running on a virtual machine (Ubuntu 20.04 server). We use a VM-based deployment as it is easier to port to any server. However, 5GFIVer can also be deployed on any other platform (e.g., bare-metal or containers) following a similar architecture as described in this section. 5GFIVer should continuously run where it is deployed (e.g., container, VM, bare-metal) to perform its verification. It uses different open-source libraries (e.g., Python library ADTK [93] is used to implement the Level Shift Detector) as well as scripts developed by us in Python. The performance metrics collector invokes a script to collect performance metrics from the underlying cloud service which is specific to the monitoring service (e.g., Amazon CloudWatch, Google Cloud Metrics, Azure Monitor, or Prometheus) and prepares input for the time-series outlier detector in it's required format. Communication between different components and modules is done using a database. We use MongoDB [94], an open-source NoSQL database engine, to implement the database.

We extended the Anomaly Detection Toolkit (ADTK) [93] to develop our spike detector, called SpikeAD, which detects spikes by tracking the difference between median values at the central and the other two outer windows of three sliding time windows next to each other. Thus, SpikeAD can detect spikes while ignoring level shifts.

4.5 Experiments

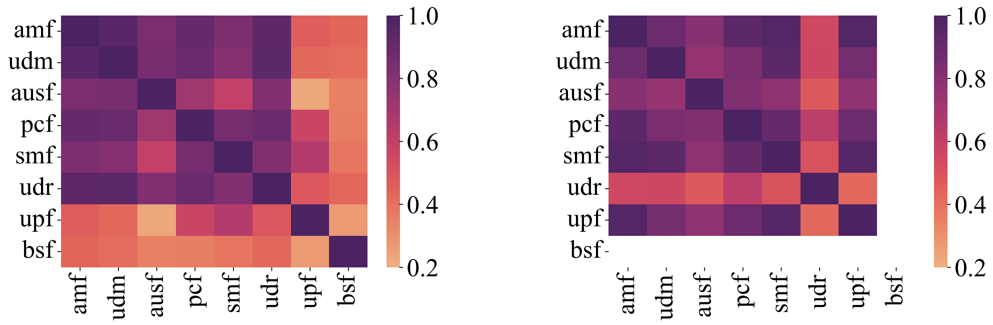
This section presents our experimental results.

4.5.1 Overview of Experiments

As 5GFIVer is the first side-channel based solution to verify the functional integrity of CNFs, we could not perform a quantitative comparison between our solution and other existing approaches (as there exist none). It is also infeasible to perform a quantitative comparison between 5GFIVer and our previous solution, APPD, because they solve two orthogonal problems (i.e., functional and forwarding integrity verification) to achieve the common goal of securing virtualized networks. Instead, we evaluate 5GFIVer in terms of its effectiveness (e.g., to find correlated CNFs and appropriate features), the accuracy of its time series algorithms, and the effect of cloud dynamicity on its overall accuracy.

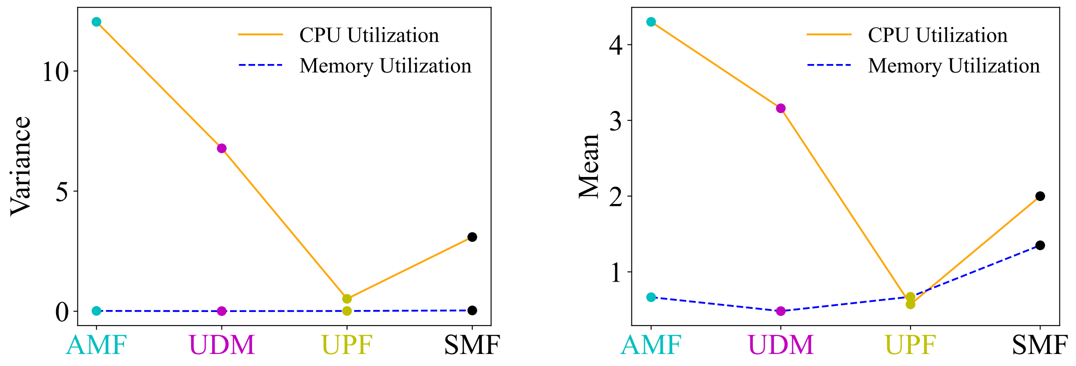
4.5.2 Experimental Settings

We adopt Open5GS-2.4.8 [86], which is a popular open-source implementation of the 5G core network, to create images of the CNFs for the 5G core. We used UERANSIM [95] to emulate the Radio Access Network (RAN) and user equipment (UE). We emulate up to 10,000 UEs in our experiments. We limit the allowed CPU cores (between 1 and 8) and memory (512 MB) for each CNF. To investigate the behavior of CNFs in terms of consuming the resources from the cloud, we deployed containers on Amazon Elastic Container



(a) Correlation (CPU utilization)　　(b) Correlaton (Memory utilization)

Figure 4.7: Heatmaps showing correlation in CPU and memory utilization of different CNFs (blank cells mean no correlation)



(a) Variance　　(b) Mean

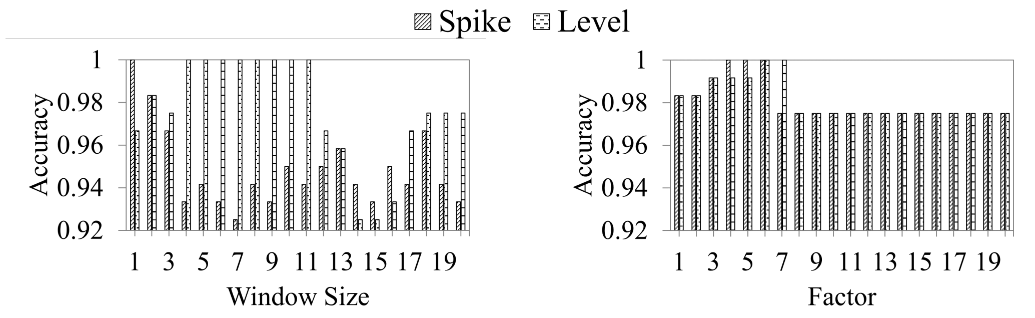
Figure 4.8: The variance and mean of CPU and memory utilization in different CNFs

Services (ECS) [19]. We then collected performance metrics for our deployed containers from Amazon CloudWatch [96] metric monitoring service.

4.5.3 Experimental Results

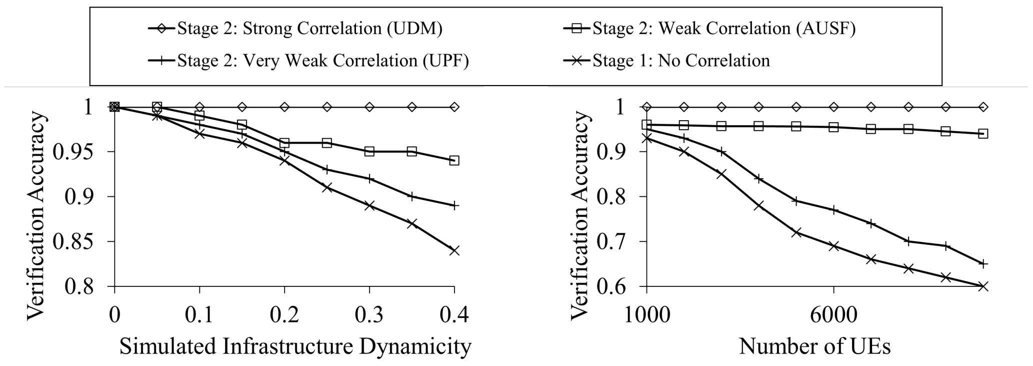
This section presents the experimental results to evaluate the effectiveness of 5GFIVer.

Metrics Selection and Multi-CNF Correlation. These sets of experiments identify the correlated behavior among different CNFs and the effectiveness of performance metrics in detecting code injection attacks. The heatmaps in Fig. 4.7 demonstrate the correlation



(a) Impact of window sizes on accuracy (b) Impact of factors on accuracy

Figure 4.9: Accuracy of time series algorithms for different hyperparameter value settings



(a) Impact of infrastructure dynamicity (b) Impact of workload variation

Figure 4.10: Impact of cloud dynamicity on the accuracy of verification on AMF for different correlated CNFs; Stage 1 (without multi-CNF correlation), Stage 2 (with multi-CNF correlation)

among multiple CNFs in terms of consuming CPU and memory. Fig. 4.7a) depicts that CPU utilization of some CNFs is highly correlated (the darker the shade, the higher the co-relation) with that of some other CNFs (e.g., AMF and UDM are correlated with each other), while, Fig. 4.7b) shows the correlation for Memory utilization. Hence, to verify the outliers (i.e., due to attack or cloud dynamicity) found in AMF, we can utilize its correlation with UDM.

On the other hand, Fig. 4.8 shows the variance and mean of CPU and Memory utilization for different CNFs. Although the variance (Fig. 4.8a) of the CPU utilization is quite high for the different CNFs, the variance of memory utilization is very low (indicating a lack of useful information [97]). Hence, to attain a higher entropy, we select CPU utilization throughout the rest of our experiments. However, memory utilization of other performance metrics can also be used for this purpose.

Accuracy of Detectors. The detection accuracy is highly dependent on the adopted detection algorithms and their parameter selection. Fig. 4.9 shows the effectiveness of detecting outliers using time series analysis algorithms (as mentioned in Section 4.3.2) for different hyper-parameters (i.e., *window size* and *factor* value) [92]. As shown in Fig. 4.9a, the level shift detector (i.e., LevelShiftAD) is most effective when *window size* is between 4 and 11, as indicated by the accuracy of 1.0. Similarly, Fig. 4.9a, and Fig. 4.9b show the effectiveness of detection when the hyper-parameters of the detection algorithms are configured correctly. Hence, by tuning these parameters, we can achieve higher accuracy in outlier detection.

Effect of CPU Sensitivity on the Accuracy of Verification. We investigate the impact of cloud dynamicity, i.e., the sensitivity of our solution to changes in the infrastructure (Fig. 4.10a) and workload variation (Fig. 4.10b) in our verification accuracy and highlight the importance of the *Stage 2* verification. To that end, we simulate an increasing level of cloud dynamicity by incrementing the number (by 0% to 40%) of occurrences of fluctuations in the performance metrics of the CNFs caused by cloud dynamicity. On the other hand, we vary the workload dynamicity by increasing the number of UEs (from 1K to 10K). Finally, we intend to detect the breaches in AMF for four different scenarios: 1) implementing *Stage 1* only (i.e., no verification from *Stage 2*), 2) considering UDM (i.e., strongly correlated with AMF) as the correlated CNF in *Stage 2*, 3) considering AUSF (i.e., weakly correlated with AMF) as the correlated CNF in *Stage 2*, and 4) considering UPF (i.e., very weakly correlated with AMF) as the correlated CNF in *Stage 2*.

Fig. 4.10a illustrates that the accuracy of *Stage 1* (i.e., without verifying by *Stage 2*)

Table 4.1: Performance profile of 5GFIVer on a lightweight Amazon EC2 virtual machine (VM) of type t2.medium (i.e., two vCPUs and 4 GB memory)

Performance measure	**Average**	**Min**	**Max**
Execution time for each iteration	1.2s	0.7s	2s
CPU utilization during each iteration	31%	11.0%	52%
Memory utilization during each iteration	1.1%	0.7%	4.0%

decreases with an increased value of dynamicity, while this decreasing trend is almost linear with the increased workload. Then to improve the performance (i.e., eliminate false positive decisions due to cloud dynamicity/workload), we implement *Stage 2* for three correlated CNFs and find that the accuracy can be regained high when the considered CNF (i.e., UDM) has a strong correlation with the AMF. On the other hand, considering UPF as the correlated CNF cannot make any significant improvement due to its very weak correlation with AMF.

Overhead Evaluation. We examine the added overhead by 5GFIVer to evaluate its efficiency. To that end, we deploy 5GFIVer on a lightweight Amazon EC2 virtual machine (VM) of type *t2.medium* and observe its performance profile. We list the average value along with the observed maximum and minimum values of various parameters (e.g., required time, CPU, and memory consumption) of the performance profile in Table 4.1 which shows that 5GFIVer is very fast to detect the breaches (e.g., it takes only 1.2s in average to complete an iteration) while adding a negligible amount of CPU and memory overhead (e.g., CPU utilization of 31% and memory utilization of 1.1% on average).

4.6 Conclusion

This chapter proposed an operator-oriented, lightweight side-channel-based black-box approach, namely, 5GFIVer, to verify the functional integrity of CNFs without relying on any underlying cloud infrastructure data. To that end, 5GFIVer first analyzed performance metrics available to the 5G operators to detect outliers that may contain many false positives due to the dynamic behavior of clouds or noises. To filter out the false positives, 5GFIVer then verified service chain integrity by correlating the outliers with the outliers found in other correlated CNFs. We integrated 5GFIVer with a popular open-source 5G core implementation (Open5GS [86]), under our testbed and the experimental results showed that our approach can effectively verify functional integrity by adding a negligible overhead. In the future, we plan to extend our approach for performing verification when the integrity of multiple CNFs can be breached, ensemble the findings from multiple detection algorithms to attain more accuracy, and further optimize other parameters of 5GFIVer. Furthermore, we plan to perform extensive security analysis and more experimental evaluations of 5GFIVer to show its effectiveness in other 5G core implementations in the future.

Chapter 5

Continuous Forwarding Integrity Verification of Virtualized Service Chains Using Side-Channel

5.1 Introduction

Virtualized service chains allow automated rerouting of data traffic through one or more virtual network functions (VNF), such as firewall, Intrusion Detection System (IDS), and Deep Packet Inspection (DPI) over a third-party cloud infrastructure. This allows network service providers (i.e., NFV tenants) to leverage the benefits of NFV (e.g., greater flexibility and cost efficiency) without having to deploy and manage their own infrastructures [56, 57, 58]. However, this virtualization also comes along with the threat of various possible forwarding integrity breaches of network services (e.g., VNF bypassing, packet dropping, fake packet injection) [20, 21, 15, 22]. Such integrity breaches are mainly reported to be caused by misconfigured (e.g., [59]) or compromised (e.g., [60, 61, 62]) components (e.g., SDN switches) of the underlying third-party cloud infrastructure. Due to the critical nature of network services, leaving even a short window for the attackers to go undetected may lead to various security and privacy issues [15]. Therefore, it is extremely important for the service providers to be able to **continuously** ensure that services are forwarding traffic exactly as intended without suffering from the aforementioned security concerns.

On the other hand, network service providers typically do not have access to the infrastructure owned by third-party providers. Which makes it difficult to directly observe such infrastructure to ensure the integrity of network services. Therefore, an interesting research challenge is to *enable the continuous verification of service chain integrity for NFV tenants without requiring access to infrastructure-level resources or data*.

Most existing efforts (e.g., [13, 14, 63, 20, 21, 15, 22]) fail to fulfill this need. Specifically, some existing works (e.g., [13, 14, 63]) rely on third-party infrastructure-level data (e.g., flow rules and flow statistics in SDN switches) to verify virtualized service chain integrity. Other existing works (e.g., [20, 21, 15, 22]) can avoid the need for third-party infrastructure-level data by using a cryptographic tagging mechanism at the VNF level. Nonetheless, those works require either (1) modifications (such as reprogramming the firmware) to infrastructure-level devices (e.g., SDN controller), which may not be practical with third-party providers, or, (2) inefficient detour of traffic to and from NFV tenants' premise. Moreover, those works are not designed to detect all types of integrity breaches (e.g., bypassing the last VNF, or all VNFs, in the service chain). To the best of our knowledge, there does not exist a *blackbox* approach (where tenant-level data along with the available side-channel data would be sufficient to continuously verify service chain integrity).

In this chapter, we bridge this gap and propose a *blackbox* approach, to allow NFV tenants to continuously verify virtualized service chain integrity without requiring any third-party infrastructure-level data. Our key ideas are twofold, **firstly**, as packets travel through the network the inter-packet gap (IPG) will be naturally affected by the switches and thus will generate additional information. So, we can possibly leverage this side-channel information of "how" the packets have been affected by the switches. Using the aforementioned intuition, we first develop a technique to detect inconsistency in packet processing by only passively observing IPGs. However, due to the noisy nature of the network traffic, relying on the passive observation-based technique alone may lead to false positives. To that end, **secondly**, we use an active probing-based technique that can confirm the occurrences of inconsistencies. It is to be noted that, the active probing technique is capable of detecting inconsistencies alone, however, it can do so only at certain intervals, leaving a window for attacks to go unnoticed. Therefore, we use it only as a second stage after the passive verification approach, which can perform continuous verification (i.e., without leaving any significant interval for attacks to go unnoticed). In the following, we describe the above two stages more specifically.

For the passive approach, we look for events (hereafter referred to as *packet swarm*) when several packets arrive at the underlying switch at a rate that the switch is unable to process. This results in packets leaving the switch at a nearly constant gap. If any packet in a swarm is dropped by the switch, it can be identified by a larger gap between the packets received at the VNF.

On the other hand, in the active probing approach, we first send two rounds of probe packets to carefully attempt to create artificial congestion. We then observe the inter-packet gap (IPG) of the response packets and use an unsupervised machine learning algorithm (i.e., clustering) to detect the actual occurrence of artificial congestion. Finally, our integrity verification algorithm can confirm a breach of integrity (e.g., VNF bypassing) based on the combination of the number of clusters detected in both rounds of probing.

We will further elaborate on our motivation and idea through an example in Section 5.2.

In summary, our main contributions are the following:

- As per our knowledge, this is the first blackbox approach that can continuously verify common forwarding integrity breaches (e.g., bypassing, fake packet injection, and packet dropping) in virtualized service chains without requiring any third-party infrastructure-level data.
- We are the first to introduce a novel hybrid method combining a passive observation and an active probing technique to detect common forwarding integrity breaches.
- As a proof of concept, we integrate the solution with OpenStack/Tacker, a popular choice for virtualized network service deployment, and, through extensive experiments in a real network environment, we demonstrate both effectiveness and efficiency (i.e., negligible overhead) of our solution.

The remainder of this chapter is organized as follows. Section 5.2 provides preliminaries. Section 5.3 presents the overview of our solution which we call Side-Channel-based Sanity Checking of Service Chains, or in short $(SC)^3$. Section 5.4 provides the detailed methodology of the passive observation-based technique. Section 5.5 discusses the active probing-based approach. Section 5.6 provides the theoretical security and performance analysis of $(SC)^3$. Section 5.7 describes the implementation details of $(SC)^3$. Section 5.8 presents our obtained experimental results. Finally, Section 5.9 concludes the chapter.

5.2 Preliminaries

In the following, we first present a motivating example. Then we define our threat model and assumptions.

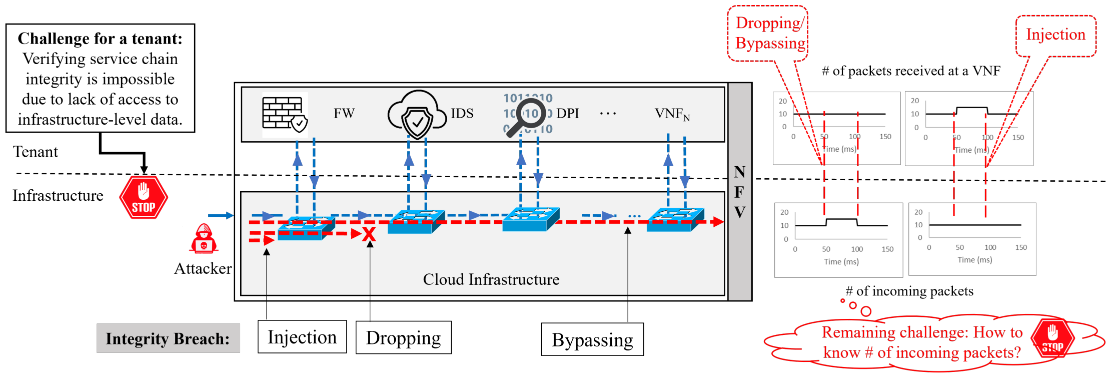

Challenge for a tenant: Verifying service chain integrity is impossible due to lack of access to infrastructure-level data.
Tenant
Infrastructure
STOP
Attacker
FW
IDS
DPI
VNF$_N$
NFV
Cloud Infrastructure
Integrity Breach:
Injection
Dropping
Bypassing
Dropping/ Bypassing
Injection
# of packets received at a VNF
Time (ms)
# of incoming packets
Remaining challenge: How to know # of incoming packets?

Figure 5.1: An example of integrity breaches in NFV

5.2.1 Motivating Example

The left of Fig. 5.1 depicts a simplified NFV deployment, with different integrity breaches (indicated by the red dashed lines). The right of Fig. 5.1 illustrates the necessity of continuous verification and the challenge for a black-box solution.

NFV Deployment. The left part of Fig. 5.1 shows an example of an NFV environment where VNFs are running on a third-party cloud provider's infrastructure. As shown in blue dashed lines, the incoming traffic is planned to pass through the service chain consisting of several VNFs, such as Firewall (FW), Intrusion Detection System (IDS), and Deep Packet Inspection (DPI) as well as their underlying cloud infrastructure (i.e., the switches).

Continuous Verification of Integrity Breaches in NFV Service Chains. The bottom-left part of Fig. 5.1 lists various integrity breaches including *injection* of fake packets, *dropping* legitimate packets, and *bypassing* one or more VNFs due to misconfigurations (e.g., [59]) by a cheap/lazy provider or attacks by exploiting compromised resources (e.g., [60, 61, 62]). As a result, traffic may follow an entirely different path (as shown in the red lines) than planned paths. An NFV tenant cannot easily verify such an integrity breach, due to the limited access to the underlying infrastructure-level data (including the flow rules of the switches).

The right part of Fig. 5.1 shows the potentially transient nature of these integrity breaches

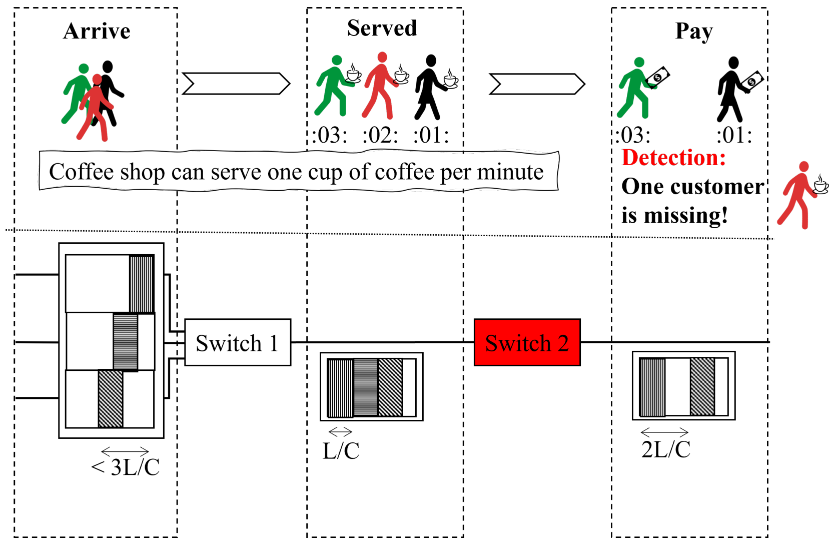


Figure 5.2: The main idea of the passive detection

where an attack can last for only a short duration. For example, in this part of the figure, we can see that for a duration of 50ms (between 50ms and 100ms in the timeline) there is (1) less (for Dropping/Bypassing attack) or (2) more (for Injection attack) number of packets flowing through the VNF layer compared to the number of packets actually passing the third-party infrastructure layer (i.e., the switches); while for the rest of the time, the number of packets passing both layers remains same. Thus, performing verification only occasionally would result in false negatives (e.g., if verification is performed between 0ms and 50ms, no integrity breach will be detected). So, it is desirable that a mechanism verifying the forwarding integrity is able to detect such a transient breach of integrity (i.e., the mechanism must be able to perform continuous verification).

The bottom-right part of Fig. 5.1 highlights the key challenge, *How to know if there is an inconsistency between the number of packets flowing through the different layers without having access to the third-party infrastructure layer?*

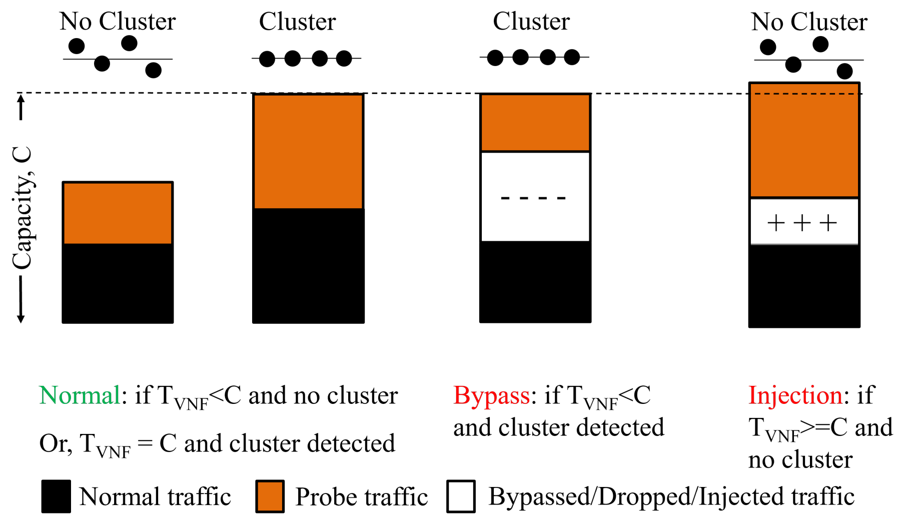

No Cluster
Cluster
Cluster
No Cluster
Capacity, C
- - - -
+ + +
Normal: if T_{VNF}<C and no cluster
Or, T_{VNF} = C and cluster detected
Bypass: if T_{VNF}<C and cluster detected
Injection: if T_{VNF}>=C and no cluster
Normal traffic
Probe traffic
Bypassed/Dropped/Injected traffic

Figure 5.3: The main ideas of the active probing

5.2.2 Main Idea

Fig. 5.2 and Fig. 5.3 illustrate our main ideas as follows. Our main idea is to leverage information carried by IPGs (also known as Inter-Packet Delay or IPD) as a side-channel to detect inconsistencies. However, a single IPG may not always reliably carry information about underlying forwarding inconsistency. To this end, our first idea is to use a notion that we call “packet swarm” and works by only passive observation of the packets. Although the packet swarm-based technique guarantees that it will always detect an inconsistency (i.e., no false negative), it cannot guarantee that the detected event is indeed an inconsistency (i.e., may contain false positive). To this end, our second idea is to use an active probing approach to confirm that the detected event by the passive technique is indeed an inconsistency (i.e., to eliminate false positives). We discuss these ideas in more detail in the following.

Idea 1: Packet Gap Sandwich. We propose a novel system, namely packet gap sandwich (PGS), that encapsulates several packets between a pair of packets. Each of these packets may come from different ingress ports of the router whose egress port is connected to the

NFV upstream. When these packets reach the destination, the final gap between the outermost pair of packets will depend on the number of packets inside the sandwich. Middlebox Bypass, packet drop, or fake generated packets can then be detected by comparing the estimated number of packets inside the sandwich with the actual number of packets received inside it.

In the following, we draw an analogy between our approach and a coffee shop to make it easy to understand. As shown at the top of Fig. 5.2, a coffee vendor can serve one coffee per minute. If any customer comes while the previous customer is still being served, then the new customer will have to wait. In order to make it easy to track which customer came first, the coffee shop employs a system where each customer is assigned a token as they arrive. After collecting tokens, the customers will wait in a queue. The coffee vendor will then serve each customer in their order of arrival. Now, suppose, three customers, $Customer_1$, $Customer_2$ and $Customer_3$ (we avoided labeling these in the figure for the sake of simplicity) arrived within the same minute and assigned tokens $Token_1$, $Token_2$ and $Token_3$. Each token has a timestamp that records the arrival time of the respective customer. Now, clearly, the timestamps as given in the tokens will fall within 1 minute. In other words, the difference between the arrival time of $Customer_1$ and the arrival time of $Customer_3$ will be less than 1 minute. That is,

$$\text{Timestamp}(\text{Customer}_3) - \text{Timestamp}(\text{Customer}_1) < 1\text{min} \tag{11}$$

As these three customers are served and leave the coffee shop their exit time, $Exittime_1$, $Exittime_2$, $Exittime_3$ is monitored. Clearly, the difference between exit time between Customer_1 and Customer_3 will be 3 min, because it takes 1 minute to serve each customer. The fact that Equation 11 is satisfied, makes these three customers a valid sample for our approach and the fact that there is one customer between $Customer_1$ and $Customer_3$ can be estimated by the $Exittime_3$-$Exittime_1$. Now, if at the exit, we see only $Customer_1$ and $Customer_3$, we can still estimate that there was one customer between $Customer_1$ and $Customer_3$ who is missing.

In reality, we do not even have access to the arrival timestamps and therefore we further modify our above idea to eliminate any need for knowing the input timestamp. This will be detailed in our methodology in Section 5.3.

Idea 2: Lightweight Active Probing to Fathom Actual Traffic Level. Our second idea is to fathom the actual traffic level at the underlying third-party cloud infrastructure by a lightweight and efficient active probing technique. To that end, we extend the concept of Packet-Pair Dispersion (PPD) [34], where the inter-packet delay between packets can create a specific pattern indicating momentary congestion. Particularly, the concept of PPD indicates that if two packets are transmitted at a rate that can cause congestion in a link, then this will lead to a specific inter-packet delay (IPD) between these two packets irrespective of their original input latency. Conversely, from observing the IPD, it is possible to infer congestion in the network.

By leveraging the above phenomenon, as shown in Fig. 5.3, our second idea is to design different probe throughputs from observed traffic throughput at the virtualized network functions to *artificially* create momentary congestion. By observing the effect on IPD while the probes are flowing through the network we can fathom whether the amount of traffic received at the virtualized network functions is the actual amount of traffic at the underlying third-party infrastructure.

Section 5.3 will further elaborate on these ideas.

5.2.3 Threat Model and Assumptions

This chapter considers integrity breaches of virtualized network service chains that may be caused when (i) *any* of the underlying forwarding devices (e.g., SDN switches [60, 61, 62]) are compromised by a malicious attacker, or (ii) the underlying forwarding devices are misconfigured (intentionally or by mistake) by a cheap-and-lazy cloud provider [59].

We consider a more challenging threat model in comparison to existing works (e.g., [20, 21, 15, 22]) by including a wide range of attack scenarios as follows. (i) *Transient attack:* Inconsistencies may happen for only a short period of time (e.g., an attacker may quickly add and then remove a flow entry, or an erroneous flow rule may be hit by a flow of short duration). Security mechanisms must detect the attack within such a short period of time. (ii) *VNF bypassing:* Compromised or misconfigured switches may bypass one or more [23] VNFs (including the possibility of bypassing all the VNFs) in the service chain. (iii) *Packet dropping:* Compromised or misconfigured switches may drop packets at any switch (e.g., first switch) rather than rerouting through the service chain as planned. (iv) *Packet injection:* Attackers may inject fake packets to exhaust resources of the VNFs at any position (e.g., before the first VNF). (v) *Packet replay:* Compromised switches may be programmed to replay packets along the service chain. In particular, attackers can evade detection by existing tagging-based or statistics-based mechanisms by replaying packets through the entire service chain. (vi) *Adaptive attack:* Attackers may try to evade detection by launching attacks only when they cannot be detected (a.k.a., coward attacks [98]). Many of these possibilities are not addressed by existing works as they are deemed hard [23].

We exclude any attack on VNFs from our threat model in this chapter. Such attacks are explored in Chapter 4. We also exclude any action by the attacker that is not supported by SDN switches (e.g., delaying packets) from our threat model.

5.3 Overview and Sanity Check

This section presents our methodology.

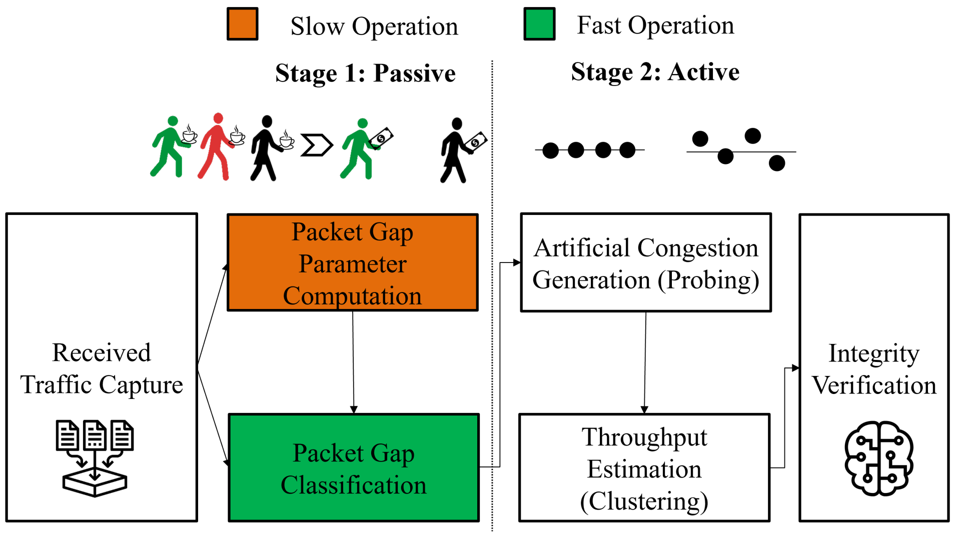

Slow Operation
Fast Operation
Stage 1: Passive
Stage 2: Active
Received Traffic Capture
Packet Gap Parameter Computation
Packet Gap Classification
Artificial Congestion Generation (Probing)
Throughput Estimation (Clustering)
Integrity Verification

Figure 5.4: A high-level overview of (SC)^3

5.3.1 Overview

Fig. 5.4 shows an overview of our methodology which contains two major stages. Stage 1 performs passive observation-based detection (detailed in Section 5.4), and Stage 2 performs active probing-based verification (detailed in Section 5.5). In Stage 1, $(SC)^3$ first passively observes side-channel information (i.e., inter-packet delays) to tentatively detect forwarding integrity breaches. In Stage 2, $(SC)^3$ further confirms the detected forwarding integrity breach (if detected by Stage 1) by using an active probing technique.

We detail these stages in Section 5.4 and Section 5.5.

5.3.2 Sanity Check

Before proceeding with building on our above ideas, we check the feasibility that (i) passive observation of inter-packet gap can be used to detect VNF bypassing, packet dropping, and injection, and (ii) active probing can be used to estimate incoming traffic throughput.

Saturation of Inter-Packet Gap (IPG). As shown in Fig. 5.5, the output inter-packet

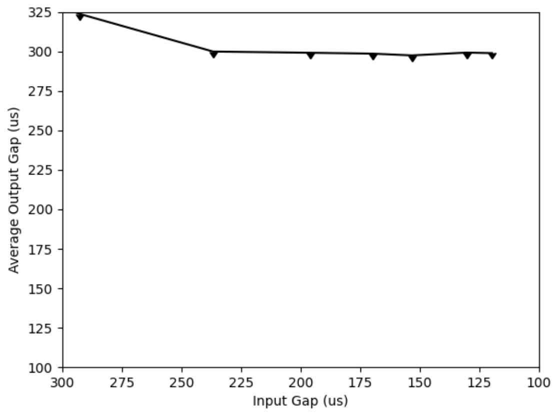
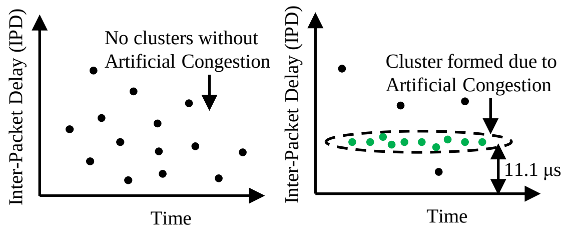


Figure 5.5: Low input IPG leads to a constant output IPG (packet swarm)


Inter-Packet Delay (IPD)
No clusters without Artificial Congestion
Time
Inter-Packet Delay (IPD)
Cluster formed due to Artificial Congestion
11.1 µs
Time

(a) No artificial congestion (b) Artificial congestion

Figure 5.6: Horizontal Inter-Packet Gap (IPG) clusters are formed due to artificially created congestion

gap (IPG) between packets becomes constant when the input IPG is sufficiently low (i.e., output IPG doesn't vary as input IPG varies). This shows the feasibility that the constant IPG can create a pattern to be detected and used as an indicator of a group of packets having sufficiently low input IPG (i.e., packet swarm). Subsequently, the packet swarm can be used to detect if any packet has been dropped, bypassed, or injected by inferring the original number of packets from the IPGs.

Clustering of IPG Under Artificial Congestion. As shown in Fig. 5.6, if we plot IPGs against time, it may or may not show specific clusters of horizontal shape depending on whether there is any congestion created in the underlying forwarding links. These clusters are distinguishable by their high density in the plot as shown in Fig. 5.6b. Although the cluster in Fig. 5.6b is easy to notice visually in this particular case, the clusters may not be easily detectable manually in general, especially from the large volume of data in real network traffic. To this end, we leverage the high spatial density of the clusters to find them even when a large amount of data is involved.

5.4 Passive Observation-based Technique

In this section, we describe our passive observation-based technique. In order for the technique to be scalable, explainable, and easy to be adjusted by a human analyst, we divide our approach into two parts: 1) parameter computation and 2) classification using computed parameters.

5.4.1 Packet Gap Parameter Computation

This step is to compute parameters that will be used by the Packet Gap Classification. The parameters are as follows: (i) saturation region inter-packet gap, (ii) minimum saturated gaps, and (iii) maximum number of virtually inserted packets. This step runs in parallel with the Packet Gap Classification step, providing updated parameter values to the latter to facilitate classification in a dynamic environment.

Saturation Region Inter-packet Gap. The output inter-packet gap (g_{out}) of a switch depends on mainly two values, the input inter-packet gap (g_{in}) and the capacity of the switch C. As g_{in} decreases, g_{out} continues to decrease until a minimum value of g_{in} beyond which the former cannot decrease any more. This region is called the saturation region which occurs when packets arrive at the switch faster than they can process (as mentioned in our main ideas). Knowing the value of g_{out} at the saturated region (g^s_{out}) will allow detection of the moments when a number of packets had arrived at the switch at a rate faster than it could process. This indirect way of detecting such moments is required because there is no way to know the actual arrival time of the packets at the switch. To this end, a naïve approach would be to compute the value of (g_{out}^{s}) by continuously monitoring g_{out} and tracking its minimum value. However, such a simple technique may fail to adapt to the dynamic cloud environment. Instead, we track the standard deviation σ_g of g_{out} in a sliding time window and update g_{out}^{s} as the median of the time window when δ_g is within a certain threshold value.

$$\sigma_g = \sqrt{\frac{1}{N}\sum_{i=1}^{N} g_i - \mu^2} \tag{12}$$

$$g_{out}^{s} = \begin{cases} \tilde{g}, & \text{if } \sigma_g \leq \sigma_g^{\text{th}} \\ \text{unchanged}, & \text{otherwise} \end{cases} \tag{13}$$

Where μ and $\tilde{g}$ are the mean and median respectively for N consecutive samples of g_{out} that form the time window.

Minimum Saturated Gaps. To reliably detect the moments of packets arriving at a rate faster than the switch can process, an important parameter is the minimum number of packets that arrive at such a rate. A value that is too high for this parameter may lead to too many false negatives while a too low value would lead to too many false positives. To address these issues, we continue to monitor consecutive packets arriving at an inter-packet gap of g_{out}^{s} and track the minimum value of their count N_{packet}^{min} that minimizes the number of false positives. This is done through a feedback process from Stage 2 to Stage 1. Here, the false positive produced by Stage 1 is calculated using the outcome of Stage 2 as the ground truth.

Example 6 *Suppose, $g_{out}^{s} = 1ms$, $N_{packet}^{min} = 8$ and we observe seven packets with 1ms gap. Then we will update N_{packet}^{min} to 7. On the other hand, if Stage 2 reports a higher false positive, N_{packet}^{min} will be increased to 8 again.*

Maximum Virtual Packets. In the event of a packet drop or bypass, detecting the moments of packets arriving at a rate faster than the switch can process may fail. This is because the dropping or bypassing will increase the inter-packet gap perceived by the VNF and the condition of $g_{out} = g^s_{out}$ will fail. To overcome this issue, a number of virtual packets are inserted to compensate for such dropped or bypassed packets. However, inserting too many virtual packets may lead to false detection. So, this parameter specifies the maximum number of such virtually inserted packets.

5.4.2 Packet Gap Classification

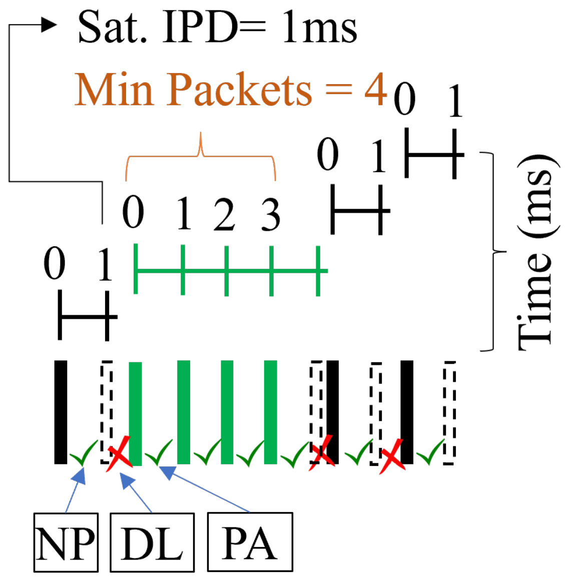
This step is to detect the moments when a number of packets had arrived at the switch at a rate faster than it could process using the parameters g^s_{out} and N^{min}_{packet} computed in the previously discussed step. To do so, it continuously monitors inter-packet gaps in incoming packets and tries to find a consecutive number of packets that satisfy the criteria $g_{out} = g^s_{out}$ and $N_{packet} \geq N^{min}_{packet}$. However, a simple incremental search will fail to find such consecutive packets if one packet is dropped and the remaining packets do not satisfy the criterion of $N_{packet} \geq N^{min}_{packet}$. To overcome this issue, when $g_{out} = g^s_{out}$ fails before satisfying $N_{packet} \geq N^{min}_{packet}$ it inserts a virtual packet at a gap of $g_{out} = g^s_{out}$.

In case of multiple packet dropping, insertion of a single virtual packet is not enough to fill the gap created by such dropping. In such case, this step will insert multiple packets (up to a maximum number specified by N^{vmax}_{packet}) to fill the gap as follows,

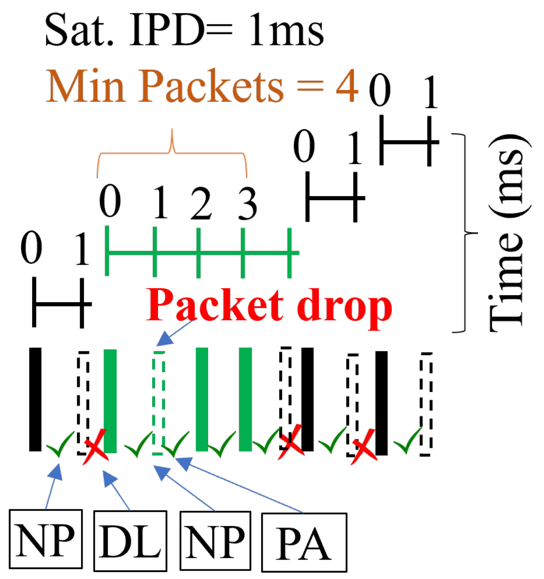
$$N_{vpacket} = \lfloor \frac{g_{out}}{g^s_{out}} \rfloor \tag{14}$$

Where $N_{vpacket}$ is the number of virtually inserted packets. In the following, we show an example of how packet gap classification can detect packet dropping.

Example 7 *As shown in Fig. 5.7, $g^s_{out} = 1ms$ and $N^{min}_{packet} = 4$. Now, we observe three packets (colored green and at the bottom part of the figure) with the following gaps 2ms*



(a) Normal



(b) Packet Drop

Figure 5.7: An example of packet gap classification to detect packet dropping; NP: **N**o **P**ackets at saturation gap (insert a virtual packet), DL: **D**elay is **L**ess than saturation gap (end of pattern), PA: **P**acket **A**t Saturation gap (continue adding packets to pattern)

and 1ms. This will not satisfy the criteria of detection. However, if we consider a virtual packet (as shown with a green dotted border in the figure) after the first packet at a 1ms gap then we see that it satisfies the criterion of $N_{packet} \geq 4$. *Here, the virtually inserted*

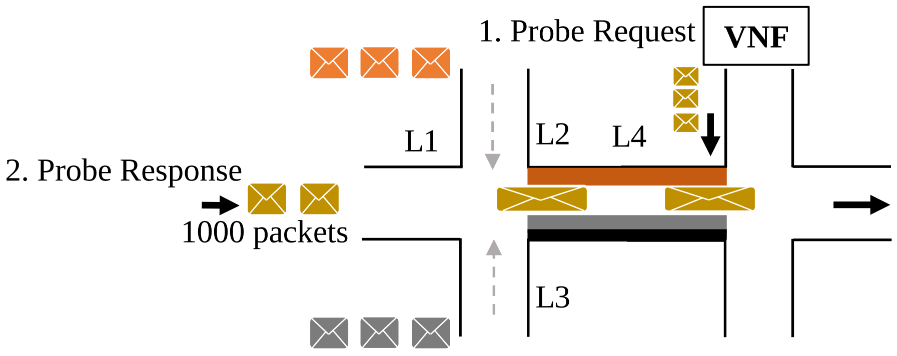


Figure 5.8: An example of probe generation

second packet is possibly a packet dropped or bypassed at the underlying switch.

Grid Search for Optimum Parameter Values. To know the optimum values of the parameters N_{packet}^{min} and N_{packet}^{vmax}, we perform a grid search over possible values of these parameters. To do so, firstly we begin with the first element of the grid (i.e., [0,0]) and observe detection performance (e.g., false positives and detection rate). Next, we continue to advance in the grid and note the detection performance for all the elements. Finally, the grid element yielding the best detection performance is used in classification. This search continues throughout the lifetime of the verification.

5.5 Active Probing-based Technique

In this section, we describe our active probing-based technique. This stage consists of the following three steps: (Step 1.1) Dual probing request, (Step 1.2) probing response capture, and (Step 1.3) clustering.

5.5.1 Dual Probing Request

This step is to create artificial congestion at the incoming link to the switch at the cloud infrastructure for a very short duration (e.g., 50ms). To do so, $(SC)^3$ generates probing packets. To achieve this, probe request traffic is generated from the VNFs using a request/response protocol (e.g., HTTP get request, ICMP echo request, etc.) to different hosts. Thus, when the reply packets reach the ingress link, they will experience artificial congestion for a very short duration.

Example 8 *As shown in Fig. 5.8, the junction (i.e., router) connecting roads (i.e., ingress links) L1, L2, L3, and the egress link L4 is a router in the cloud infrastructure. Probe traffic, shown in golden envelopes, is sent to this router as a response to requests sent from the VNF (specifically from the first VNF in the service chain) through ingress links L1. As a result of this probe traffic combined with existing traffic all links (e.g., L1, L2, and L3) artificial congestion is created, and the traffic leaves the link L4 experiencing a clustering effect in their inter-packet gaps (IPGs).*

The probe traffic generation module is designed to ensure there will be artificial congestion to cause the clustering effect on the traffic at the ingress link. To achieve this, we adjust the probe throughout based on the actual received traffic throughput at the VNF (which should be the same as the traffic at the ingress link, link L4 in Fig. 5.8, when there are no integrity breaches) such that the combined throughput of probe traffic and actual traffic will be equal to the ingress link capacity. As a result, the clustering effect will induce distinguishable patterns in terms of the inter-packet delay, as discussed in Section 5.3.2 and evaluated in Fig. 5.10. More formally, for ingress link capacity C, probe rate T_P, received traffic throughput T_{VNF} and the ingress link traffic throughput λ, clusters will be found when $T_P + \lambda \geq C$.

Now, considering the ingress link throughput to be equal to the received throughput (i.e., $\lambda = T_{\text{VNF}}$), the combined traffic will be equal to ingress link capacity (i.e., $T_P + \lambda = C$) when probe throughput is set to,

$$T_P = C - T_{\text{VNF}} \tag{15}$$

Sending only one round of probing packets with the throughput calculated above may result in a false estimation if the ingress link throughput is more than the received traffic throughput (i.e., in case of fake traffic injection). To avoid this possibility of false estimation, two rounds of probing packets are sent. One at probe throughput T_{P1} and another at probe throughput T_{P2} as given in the following equations,

$$T_{P1} = C - T_{\text{VNF}} - \delta_T \tag{16}$$

$$T_{P2} = C - T_{\text{VNF}} \tag{17}$$

Here, the parameter δ_T is a small number that can be configured by the tenant admin. Computing this parameter automatically by using an efficient binary search approach will be an interesting future work. The number of clusters generated at probe throughput T_{P1} is denoted as N_{C1} and the number of clusters generated at probe throughput T_{P2} is denoted as N_{C2}.

5.5.2 Probing Response Capture

In this step, $(SC)^3$ first collects attributes of each packet (e.g., timestamp, size in bytes, etc.) by sniffing packets from the network interface, and then calculates IPD values from the timestamps.

Example 9 *As shown in Fig. 5.8, 1,000 probe response packets are generated and received at the first VNF having timestamps $P_1 \rightarrow t_{P1}$, $P_2 \rightarrow t_{P2}$, $P_3 \rightarrow t_{P3}$, ..., $P_{1000} \rightarrow t_{P1000}$. Now, the packet capture step at the first VNF will output the following to the next module: t_{P1}, t_{P2}, t_{P3}, ..., t_{P1000}. IPD values will then be calculated as follows: $D_1 = t_{P2} - t_{P1}$, $D_2 = t_{P3} - t_{P2}$, $D_3 = t_{P4} - t_{P3}$, ..., $D_{999} = t_{P1000 - t_{P999}}$.*

5.5.3 Clustering

This step is mainly responsible for clustering the data points formed by inter-packet delays paired with corresponding timestamps on each time window. As mentioned in Section 5.3.2, inter-packet delay values form clusters of special shape (i.e., horizontal) when congestion is present in the underlying forwarding links. We use our extended version DB-SCAN algorithm (for improved performance) as the clustering algorithm and CityBlock distance metric. The use of the CityBlock distance metric lets us select only those clusters that are spread horizontally and have a very small height. For a real example of the clusters, see Fig. 5.10 from our experimental results. After clustering, if at least one cluster is found, then the formation of artificial congestion is ensured for the current round of probing.

Example 10 *As shown in Fig. 5.6b, clustering algorithm on IPD values: $D_1 = t_{P2} - t_{P1}$, $D_2 = t_{P3} - t_{P2}$, $D_3 = t_{P4} - t_{P3}$, ..., $D_{999} = t_{P1000 - t_{P999}}$ finds zero clusters for the first round (i.e., $N_{C1} = 0$) and two clusters for the second round (i.e., $N_{C2} > 0$). Then artificial congestion is not confirmed for the first round but confirmed for the second round.*

Finally, the following logic is applied to confirm and classify the integrity breach,

$$\text{Integrity} = \begin{cases} \text{Normal}, & \text{if } N_{C1} = 0 \text{ and } N_{C2} > 0 \\ \text{Drop/Bypass}, & \text{if } N_{C1} > 0 \\ \text{Injection}, & \text{if } N_{C2} = 0 \end{cases}$$

5.6 Security and Performance Analysis

In this section, we analyze the security guarantee and performance optimizations of $(SC)^3$.

5.6.1 Security Analysis

In the following paragraphs, we analyze the security guarantee of the $(SC)^3$ methodology as described in Sections 5.3 - 5.5; where we describe how $(SC)^3$ verifies service chain integrity. To facilitate our analysis, we consider a service chain consisting of N number of VNFs whose intended forwarding order is: VNF$_1$, VNF$_2$, ... VNF$_N$; and a packet swarm, S_t, consisting of packets: P_1, P_2, ..., P_N being forwarded via the service chain at time t. We also consider the possibility of an adaptive attack [98], where an attacker tries to evade the detection by $(SC)^3$.

VNF Bypassing/Packet Dropping. If packets are bypassing a VNF (e.g., VNF$_1$) or being dropped, one or more of the packets in S_t will be missing at VNF$_1$. Those missing packets will be detected by Stage 1. Subsequently, as VNF$_1$ will be receiving less amount of traffic than the incoming traffic volume, Stage 2 will detect clusters in both rounds of probing. Therefore, the integrity breach will be detected by $(SC)^3$.

On the other hand, if packets are bypassing or being dropped from any other VNF (e.g., VNF$_2$), its expected value of incoming traffic throughput T_E will be greater than the traffic received by VNF$_2$. Thus, VNF$_2$ will be able to detect if it is being bypassed.

Packet Injection. If packets are being injected into a VNF (e.g., VNF$_1$), there must be one or more pairs of packets received at VNF$_1$, having an inter-packet gap $g < g^s_{\text{out}}$. These events will be detected by Stage 1. Subsequently, as VNF$_1$ will be receiving more amount of traffic than the incoming traffic volume, Stage 2 will not detect clusters in any of the two rounds of probing. Therefore, the integrity breach will be detected by $(SC)^3$.

On the other hand, if packets are being injected into any other VNF (e.g., VNF$_2$), its expected value of incoming traffic throughput T_E will be less than the traffic received by VNF$_2$. Thus, VNF$_2$ will be able to detect if packets are being injected into itself. Similarly, the remaining VNFs will also be able to perform packet injection.

Adaptive Attack. We consider the following possibilities that would allow an attacker to evade our detection:

(1) there are no missing packets in any packet swarm when packets are being bypassed or dropped,

(2) there are no inter-packet gaps of $g < g^s_{\text{out}}$ when packets are being injected,

(3) there is no cluster detected when clusters should be present, and

(4) there are present clusters when there should be no clusters.

However, the attacker is capable of only modifying flow rules or injecting traffic. Given these capabilities, it would be impractical for the attacker to modify inter-packet delays between millions of packets. Thus, the above possibilities are infeasible. Therefore, an attacker is unlikely to be able to evade our detection.

Network Issues. Due to different network issues packets may be lost, delayed, or out-of-order. While jitter or out-of-order packets will not affect our probing mechanism as our clustering algorithm is inherently immune to noise [99], packet loss can affect our detection. However, in contemporary high-performance networks, packet loss is typically low and can be accommodated by adjusting the parameter Δ as follows.

$$\Delta = \delta + L_{\text{max}} * T_{\mathbf{VNF}} \tag{18}$$

In the above, δ is the value of Δ without considering packet loss, L_{max} is the maximum packet loss rate, and $T_{\mathbf{VNF}}$ is the received traffic volume at the VNF.

5.6.2 Performance Optimization

$(SC)^3$ uses clustering to detect congestion. However, finding all the clusters is unnecessary for the purpose of detecting congestion. Therefore, we modified our algorithm to terminate as soon as the first point satisfying the criterion of a core point (i.e., ϵ and minPts)

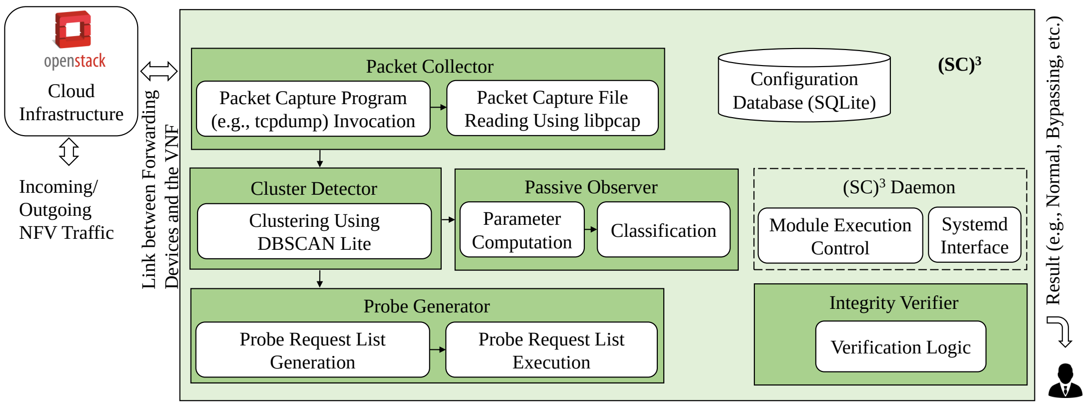

openstack
Cloud Infrastructure
Incoming/ Outgoing NFV Traffic
Link between Forwarding Devices and the VNF
Packet Collector
Packet Capture Program (e.g., tcpdump) Invocation
Packet Capture File Reading Using libpcap
Configuration Database (SQLite)
(SC)³
Cluster Detector
Clustering Using DBSCAN Lite
Passive Observer
Parameter Computation
Classification
(SC)³ Daemon
Module Execution Control
Systemd Interface
Probe Generator
Probe Request List Generation
Probe Request List Execution
Integrity Verifier
Verification Logic
Result (e.g., Normal, Bypassing, etc.)

Figure 5.9: The architecture of $(SC)^3$

is found. Therefore, although the best-case time complexity of DBSCAN would be O(n log n), the best case for our modified algorithm is O(log n). This is because even the first point in the dataset can be a core point. Thus, the clustering algorithm of $(SC)^3$ performs faster than the original DBSCAN algorithm.

5.7 Implementation

This section presents the implementation of $(SC)^3$.

Architecture. There are seven major components of $(SC)^3$ (Fig. 5.9): (i) the $(SC)^3$ daemon for orchestrating the other modules, (ii) the packet collector for extracting metadata from incoming packets, (iii) the cluster detector for performing clustering on metadata (i.e., inter-packet gap), (iv) the passive observer for performing passive observation-based detection (given in Algorithm 1), (v) the probe generator for active probing, (vi) the integrity verifier for consolidating outcomes of other modules into concrete results, and (vii) the configuration database for storing parameters (e.g., maximum swarm size) for different modules of $(SC)^3$.

The cluster detector helps both passive observation-based detection and active probing-based detection. It does so by continuously performing clustering on collected packet metadata (e.g., inter-packet gap) irrespective of whether there is an active probing happening or not. Then it is controlled by the $(SC)^3$ daemon whether the outcome of the cluster detector is processed by the passive observer or directly by the integrity verifier (i.e., interpreted as part of active probing).

Implementation Details. $(SC)^3$ has been implemented as a Linux service using systemd [66]. Linux was chosen because it is the most popular operating system in the cloud [67]. However, $(SC)^3$ can also be deployed in VNFs based on other operating systems following a similar architecture as described in this section. Being a service, $(SC)^3$ is deployed on each VNF, started as soon as the VNF operating system (OS) is booted, and continues to run as long as the VNF OS is running. Some of the modules that need to work at the line rate are developed using C programming language while other modules are developed in Python. We describe the implementation of our verification in Algorithm 1.

Efficient Parameter Learning. As the parameters learned by one VNF for the passive detection remain effective for other VNFs as well, in our implementation, these parameters are learned by only one VNF and then shared with other VNFs. To this end, we define three types of VNFs: 1) repository, 2) designated, and 3) ordinary. The designated VNF is responsible for performing learning of the parameters and updating the repository. On the other hand, ordinary VNFs retrieve these parameters from the repository and reuse them.

5.8 Experiments

This section presents our experimental results.

5.8.1 Overview of Experiments

As $(SC)^3$ is the first side-channel-based solution to perform continuous forwarding integrity verification of virtualized service chains, a quantitative comparison between our solution and other existing approaches is infeasible. Instead, we evaluate $(SC)^3$ in terms

Algorithm 1: $(SC)^3$ Verification

```
1  PassiveDetection ()
       Input: g_out, g_out_sat, $\epsilon$
       Output: Triggering Active Detection if necessary
       Data: Packet metadata $packet$
2      while nextPacketReceived do
3          if $g\_out\_sat - g\_out < \epsilon$ then
4              Increment $n\_packet$ by one
5          else
6              $n\_vpacket = g\_out\_sat/g\_out$
7              if $n\_vpacket < n\_vpacket\_max$ then
8                  Increment $n\_packet$ by $n\_vpacket$ Set $has\_vpacket$ to true
9              else
10                 Reset $n\_packet$ to zero Reset $has\_vpacket$ to false
11         if $n\_packet >= n\_packet\_min$ and $has\_vpacket$ then
12             $ActiveDetection()$
13 ActiveDetection ()
       Input: g_out, g_out_sat, $\epsilon$
       Output: Confirm inconsistency
       Data:
14     Calculate probe throughputs
15     Send probe packets
16     Perform verification
```

of its ability to correctly verify experimental continuous verification scenarios and its overhead. Furthermore, we compare the hybrid technique (by combining passive observation and active probing techniques) used in $(SC)^3$ with our previous active-probing-only technique.

5.8.2 Experimental Settings

To conduct our experiments, we build our NFV testbed using OpenStack [18] which is a very popular infrastructure-as-a-service (IaaS) software, and Tacker [70] which is an official OpenStack [27] project providing a VNF Manager (VNFM) and an NFV Orchestrator (NFVO) for deploying and managing VNFs. Our testbed includes one controller node and up to 80 compute nodes, each with 4 CPUs and 8 GB RAM running Ubuntu 20.04 server. We have used Mininet-2.3.0 [71] to set up the user-side network and Internet links (between the user-side network and NFV) with virtual hosts, virtual links, and Open vSwitch (OVS) [72] virtual switches on a dedicated server. To connect the user-side network to the service chains, the server where the user-side network is set up is then connected to the NFV testbed using a 10Gbps local area network (LAN). Also, similar to real ISP, we set up a traffic shaper to limit the bandwidth (to 1Gbps) from the user-side network to NFV using the Linux traffic control module NetEm [73]. We also set up 10 virtual hosts inside the user-side network and 10 additional virtual hosts connected to the Internet switches. The virtual hosts either act as video servers (using ffserver [74]) or video clients (using MPlayer [75]). On one hand, to generate the user-side network traffic, hosts inside the user-side network act as video clients to stream video from video servers in the Internet. On the other hand, to generate cross-traffic [76], hosts outside the user-side network act as video clients to stream video from video servers on the Internet.

Table 5.1: Applying $(SC)^3$ in real network setting shows that it could correctly verify all the experimental scenarios

Exp No.	**Experimental Integrity Scenario**	**Swarm Size (N_{packet})[1]**	**Number of Virtual Packets (N_{vpacket})**	**Probe Throughput (T_{P1})**	**Probe Throughput (T_{P2})**	**No. of IPD Clusters (N_{C1})**	**No. of IPD Clusters (N_{C2})**	$(SC)^3$ **Result[2]**
1	*Normal*	5	0	0	-	-	-	$N_{\text{packet}} < N_{\text{packet}}^{\max}$ and $N_{\text{vpacket}} = 0$: *Normal*
2	*Bypass/Drop*	5	1	540	600	2	2	$N_{\text{vpacket}} > 0$ $\rightarrow$ $N_{C1} > 0$: *Bypass/Drop*
3	*Injection*	8	0	340	400	0	0	$N_{\text{packet}} > \text{N}_{\text{packet}}^{\max}$ $\rightarrow$ $\text{N}_{C2} = 0$: *Injection*

[1] Max Swarm Size = 6.
[2] Detection logic for passive observation and active probing shown on the left and right side of the arrow ($\rightarrow$) respectively.

5.8.3 Experimental Results

We present our experimental results to evaluate the effectiveness and overhead of $(SC)^3$ as follows.

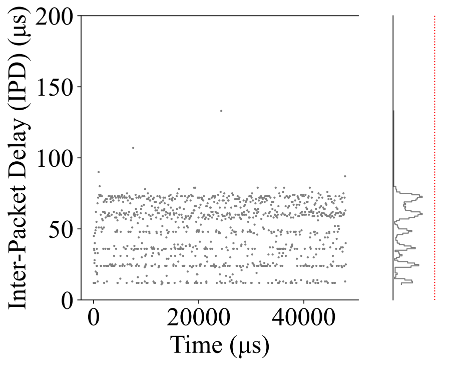
Effectiveness in Continuously Verifying Service Chain Integrity. Table 5.1 demonstrates the effectiveness of $(SC)^3$ through three different scenarios (including different attacks such as bypass, drop, injection, as well as normal behavior) where $(SC)^3$ could correctly detect all common breaches. We emulate the attacks by modifying the flow rules of the SDN switches in our testbed. This table shows the hybrid integrity verification steps without going into detail of the parameter learning and classification steps of the passive detection approach. In the following, we explain the three scenarios listed in Table 5.1. In these scenarios, the VNF is receiving traffic at a throughput $_{\text{VNF}} = 500$Mbps at the time of verification, capacity $C = 1$Gbps and $\delta_T = 60$Mbps.

- *First scenario:* The passive observation algorithm is seeing an average swarm size of five and has not found any virtual packets. As the average swarm size is less than the maximum threshold it is not suspected packet injection. On the other hand, as there is no virtual packet inserted, packet drop/bypassing is not detected either. Therefore, $(SC)^3$ concludes that the scenario is normal and there is no need for the active probing stage.

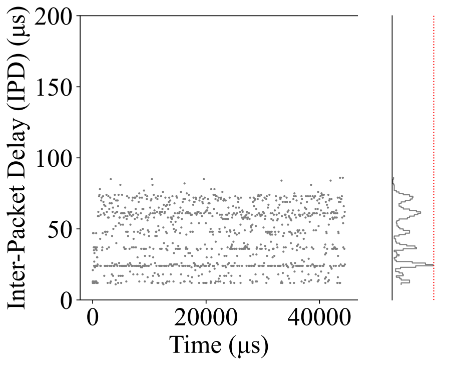
- *Second scenario:* Traffic is bypassing/dropping and therefore $(SC)^3$ has detected it in the passive observation stage by one virtual packet. It then further ensures that there is indeed bypassing/dropping happening by active probing as follows. As the actual throughput received at the VNF is $T_{VNF} = 400$Mbps the active probing stage calculates the throughput for first probe requests $T_{P1} = C - T_{VNF} - \delta_T = 540$Mbps and the clustering algorithm finds two clusters. Since $N_{C1} > 0$, ***Bypass/Drop*** is confirmed.

- *Third scenario:* Traffic is being injected and therefore $(SC)^3$ has detected it in the passive observation stage by an increased swarm size of eight which is higher than the maximum swarm size (i.e., six). It then further ensures that there is indeed packet injection happening by active probing as follows. As the actual throughput received at the VNF is $T_{VNF} = 600$Mbps the active probing stage calculates the throughput for first probe requests $T_{P1} = C - T_{VNF} - \delta_T = 340$Mbps and for second probe requests $T_{P2} = C - T_{VNF} = 400$Mbps the clustering algorithm finds no clusters. Since $N_{C2} = 0$, ***Injection*** is confirmed.

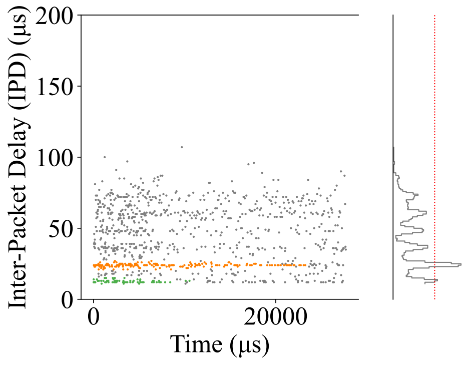
Effectiveness of Probing and IPD Clustering. In Fig. 5.10, we demonstrate the IPD clustering results for different probe throughputs in a *Normal* scenario (i.e., no integrity breaches). As the clusters are difficult to see due to the high density of the links, we additionally plot the histogram of the number of points having the same IPD at the right of



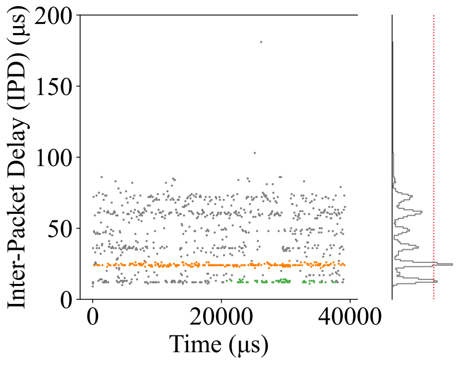
(a) Probe Throughput: 400Mbps



(b) Probe Throughput: 440Mbps



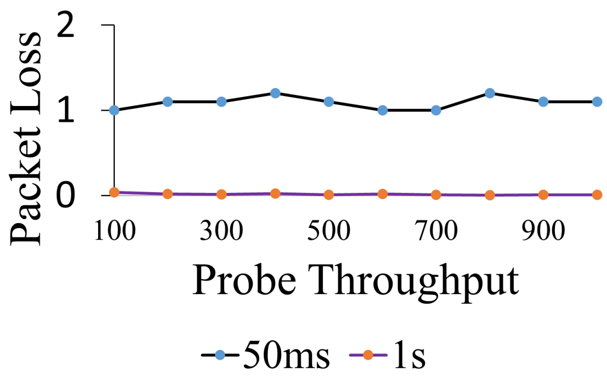
(c) Probe Throughput: 450Mbps



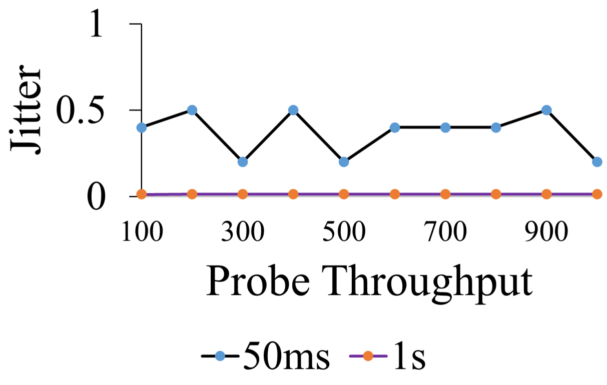
(d) Probe Throughput: 500Mbps

Figure 5.10: Active probing approach is applying DBSCAN clustering algorithm on inter-packet delay (IPD); Noise (●), Cluster 1 (🟢), Cluster 2 (🟠)

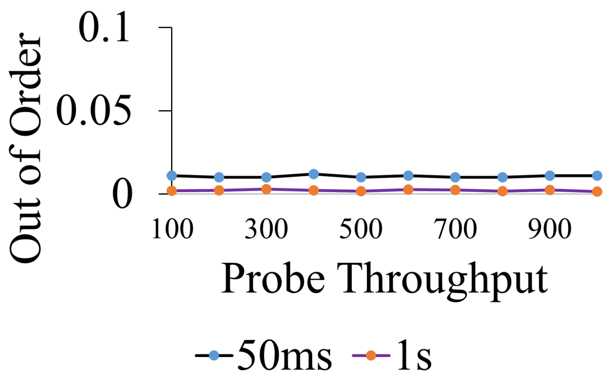
each graph. From these histograms, it is clearly visible that where clusters are formed, the number of points having the same IPD is relatively high and crosses the red vertical line. Now, in this scenario, the tenants' last-mile link capacity and throughput are 1Gbps and 500Mbps, respectively. For lower probing throughputs: 400Mbps (Fig. 5.10a) and 440Mbps (Fig. 5.10b) no cluster is formed, and for higher probing throughputs: 450Mbps (Fig. 5.10c) and 500Mbps (Fig. 5.10d) two clusters (as indicated in orange and green) are formed. Here the transition from no clusters to two clusters happens between probing throughput 440Mbps and probing throughput 450Mbps. Therefore, $(SC)^3$ expects no clusters in a *Normal* scenario for its first round of probing ($T_{P1} = 440$Mbps), as calculated from Equation 16 in Section 5.5. Similarly, $(SC)^3$ expects one or more clusters for its second round of probing ($T_{P2} = 500$Mbps), as calculated from Equation 16 in Section 5.5.



(a) Packet loss



(b) Jitter

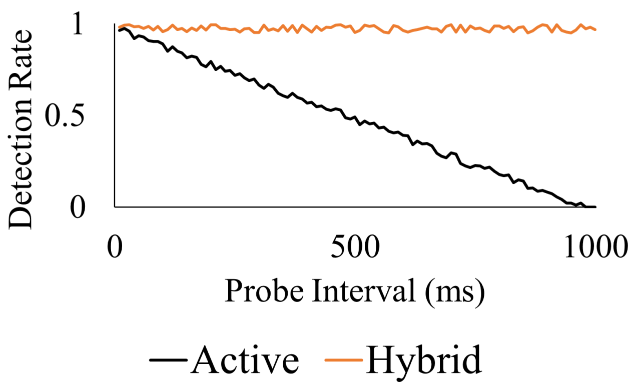


(c) Out-of-order packets

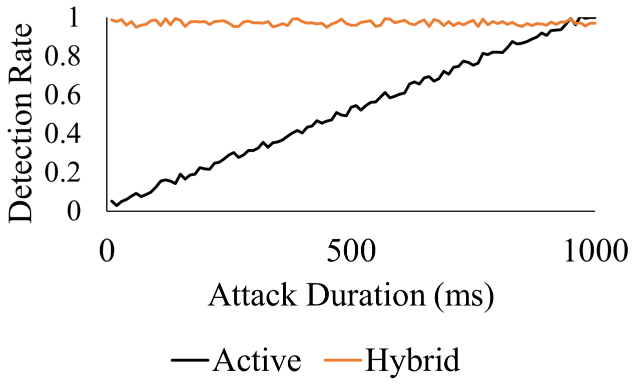
Figure 5.11: Comparison of the effect of probing interval on overhead in terms of network performance metric (packet loss, jitter, out-of-order packets) for active approach

Overhead. We evaluate the overhead of $(SC)^3$ in terms of impact on different network performance metrics (e.g., packet loss, jitter, and packet reordering). To do so, we measure these metrics while performing tenant network throughput estimation at different possible probe rates. To measure these metrics, we capture packets (at both video clients and video servers) and perform calculations on these packets by identifying the same packets using Transmission Control Protocol (TCP) sequence numbers. The results of these experiments are shown in Fig. 5.11 (Fig. 5.11a illustrates the packet loss, Fig. 5.11b shows jitter and Fig. 5.11c depicts out-of-order packets) where we can see that these metrics may or may not be significantly affected depending on the probing interval. When the probing interval is high (e.g., 1s), there is little or no effect on the performance metrics. However, a lower probing interval (e.g., 50ms) may significantly affect these metrics. As $(SC)^3$ uses active probing only when an integrity breach is detected by the passive observation-based approach, we do not require a low probing interval.

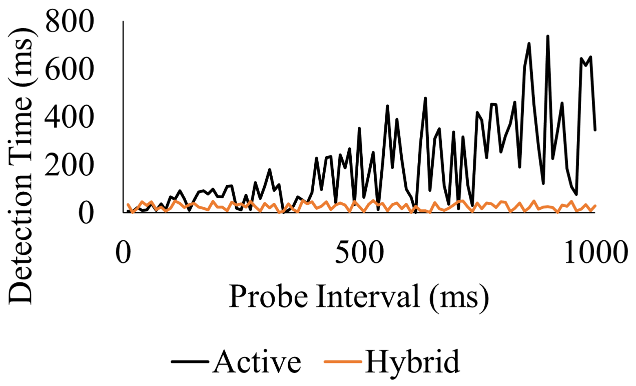
Comparison of Detection Performance With Active Probing. We compare the performance (in terms of detection rate and detection time) hybrid approach in $(SC)^3$ to our previous active-probing-only approach as shown in Fig. 5.12. Fig. 5.12a shows the detection rate for the two approaches while the verification interval is increased. Here, the detection rate increases linearly with the decrease of verification interval for both approaches. Fig. 5.12b illustrates the detection rate for both the active-probing-based approach and the hybrid approach while increasing attack duration. Here, the detection rate increases with the increase in attack duration. However, the detection rate for the hybrid approach increases very quickly to peak while the active probing approach shows a linear improvement. Fig. 5.12c illustrates the detection time for both active and hybrid approaches while the verification interval is increased. Here, the detection time increases for both approaches with the increase in verification interval.



(a) Effect of probe interval on detection rate



(b) Effect of attack duration on detection rate



(c) Effect of probe interval on detection time

Figure 5.12: Comparison of detection performance between the hybrid and active approaches

5.9 Conclusion

This chapter proposed a side-channel-based approach, namely, $(SC)^3$, to verify service chain integrity in virtualized networks without requiring any access to third-party infrastructure-level data or resources. Additionally, $(SC)^3$ can verify integrity breaches continuously without leaving a large window for attackers to escape verification. To that end, $(SC)^3$ employs a hybrid approach that combines a preliminary passive mechanism that works continuously and upon preliminary detection, it further confirms integrity breaches using an active probing mechanism. Experimental results in a real network environment showed that our approach can effectively verify service chain integrity for a wide range of integrity breaches while maintaining a negligible impact on network performance. Additionally, we have performed an extensive security analysis of $(SC)^3$. In future work, we plan to automate setting the parameters of the clustering algorithm and further optimize other parameters of $(SC)^3$.

Chapter 6

Conclusion

Network virtualization is a rapidly growing technology that has received a lot of interest from the industry and academia due to its potential benefits. However, this softwarization opens the door to many security vulnerabilities that must be carefully considered before harvesting these benefits. To this end, there exist two types of solutions: pre-deployment verification and runtime verification. However, existing works under these categories fail to provide solutions for virtualized networks under the constraints of lack of visibility and performance sensitivity of modern communication services. In this thesis, we proposed mechanisms to overcome the above-mentioned challenges by using side channel information (at the tenant-side) as the indirect effects of the attacks. To this end, we first proposed an approach to verify the forwarding integrity of virtualized network function (VNF) chains; which covered a wide range of integrity verification scenarios (e.g., entire service chain bypassing, packet dropping, and packet injection). Second, we proposed mechanisms to detect breaches of integrity of the individual network functions (NF); which can detect zero-day attacks without affecting the performance of the NFs. Third, we proposed mechanisms for continuous verification of the forwarding integrity of the service chains (i.e., minimizing the detection time as much as possible).

However, our work has a few limitations, which will be addressed in future works.

- First, our approach is based on side channels and therefore not as precise as solutions that directly observe cloud configurations. In the future, we intend to improve the precision of our techniques using multiple side channels and more advanced machine learning techniques. Also, we will cover a wider range of integrity verification scenarios to validate the effectiveness of our side-channel based solution.

- Second, our solution to verify functional integrity may not work properly when all the network functions are compromised by a coordinated attack. We plan to overcome this limitation in the future using a self-learning technique to provide a certain degree of resilience in each network function.

- Third, our functional integrity verification approach cannot work when an attack does not affect performance metrics. In the future, we hope to discover more side channels beyond the performance metric to overcome this issue.

- Finally, although we have tested our solutions on various platforms (e.g., OpenStack, Hyperscale Cloud Providers, Open5GS, and free5GC), we have not deployed our solutions in a live environment. In the future, we will explore the possibility to test and optimize our solutions in an environment with active users (e.g., in a cyber range). In addition, we plan to automate the parameter settings of our machine-learning algorithms.

In summary, this work significantly contributed towards providing security assurance to cloud tenants who operate virtualized network services. We believe that our work can facilitate future research as follows. First, our network-based side channels can solve problems in other network security areas (e.g., early detection of DDoS attacks without requiring access to routers on the Internet). Second, our idea of using performance metrics as a side channel and using correlation between multiple functions to filter out false positives may be applied to solve other security problems (e.g., efficiently detecting compromised switches in a large network).

Bibliography

[1] *Evolve your core network for 5G*. [Online]. Available: https://www.ericsson.com/en/core-network/5g-core

[2] *Network Function Virtualization (NFV) Market*. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/network-function-virtualization-market-93929190.html

[3] *NFV Security*. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv-sec/001_099/003/01.01.01_60/gs_nfv-sec003v010101p.pdf

[4] M. Pattaranantakul, R. He, Q. Song, Z. Zhang, and A. Meddahi, "NFV security survey: From use case driven threat analysis to state-of-the-art countermeasures," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3330–3368, 2018.

[5] *What is cloud computing?* [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing#:~:text=Simply%20put%2C%20cloud%20computing%20is,resources%2C%20and%20economies%20of%20scale.

[6] *5G service requirements*. [Online]. Available: https://www.3gpp.org/news-events/3gpp-news/sa1-5g

[7] Y. Yue and B. Cheng, "EasyOrchestrator: A NFV-based network service creation platform for end-users," in *IEEE IPCCC*, 2018.

[8] N. Bouten, R. Mijumbi, J. Serrat, J. Famaey, S. Latré, and F. De Turck, "Semantically enhanced mapping algorithm for affinity-constrained service function chain requests," *IEEE TNSM*, vol. 14, no. 2, 2017.

[9] L. Durante, L. Seno, F. Valenza, and A. Valenzano, "A model for the analysis of security policies in service function chains," in *IEEE NetSoft*, 2017, pp. 1–6.

[10] M. Bonfim, F. Freitas, and S. Fernandes, "A semantic-based policy analysis solution for the deployment of NFV services," *TNSM*, vol. 16, no. 3, pp. 1005–1018, 2019.

[11] *Amazon EC2 cloud is made up of almost half-a-million Linux servers*. [Online]. Available: https://www.zdnet.com/article/amazon-ec2-cloud-is-made-up-of-almost-half-a-million-linux-servers/

[12] X. Zhang, H. Duan, C. Wang, Q. Li, and J. Wu, "Towards verifiable performance measurement over in-the-cloud middleboxes," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1162–1170.

[13] M. K. Shin, Y. Choi, H. H. Kwak, S. Pack, M. Kang, and J. Y. Choi, "Verification for NFV-enabled network services," in *IEEE ICTC*, 2015.

[14] M. Flittner, J. M. Scheuermann, and R. Bauer, "ChainGuard: Controller-independent verification of service function chaining in cloud computing," in *IEEE NFV-SDN*, 2017, pp. 1–7.

[15] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "FlowCloak: Defeating middlebox-bypass attacks in software-defined networking," in *IEEE INFOCOM*, 2018.

[16] A. Aljuhani and T. Alharbi, "Virtualized network functions security attacks and vulnerabilities," in *IEEE CCWC*, 2017, pp. 1–4.

[17] S. Lal, T. Taleb, and A. Dutta, "NFV: Security threats and best practices," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 211–217, 2017.

[18] *OpenStack*. [Online]. Available: https://docs.openstack.org

[19] *Amazon ECS*. [Online]. Available: https://aws.amazon.com/ecs/

[20] P. Zhang, "Towards rule enforcement verification for software defined networks," in *IEEE INFOCOM*, 2017.

[21] X. Zhang, Q. Li, J. Wu, and J. Yang, "vSFC: Generic and agile verification of service function chains in the cloud," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[22] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "Securing middlebox policy enforcement in SDN," *Computer Networks*, vol. 193, 2021.

[23] N. C. Thang and M. Park, "Detecting compromised switches and middlebox-bypass attacks in service function chaining," in *IEEE ITNAC*, 2019.

[24] Y. Yue and B. Cheng, "EasyOrchestrator: An end-user oriented network service creation platform with verification mechanism," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2019, pp. 1–6.

[25] P. Twamley, M. Müller, P.-B. Bök, G. K. Xilouris, C. Sakkas, M. A. Kourtis, M. Peuster, S. Schneider, P. Stavrianos, and D. Kyriazis, "5GTANGO: An approach for testing nfv deployments," in *2018 European Conference on Networks and Communications (EuCNC)*. IEEE, 2018, pp. 1–218.

[26] M. Touloupou, E. Kapassa, A. Mavrogiorgou, and D. Kyriazis, "Towards optimized verification and validation of 5G services," in *2019 Sixth International Conference on Software Defined Systems (SDS)*. IEEE, 2019, pp. 5–10.

[27] G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "A framework for verification-oriented user-friendly network function modeling," *IEEE Access*, vol. 7, pp. 99 349–99 359, 2019.

[28] S. Chen, J. Li, B. Chen, D. Guo, and K. Li, "vHSFC: Generic and agile verification of service function chain with parallel VNFs," in *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 2023, pp. 498–503.

[29] G. Wagener, A. Dulaunoy *et al.*, "Malware behaviour analysis," *Journal in computer virology*, vol. 4, no. 4, pp. 279–287, 2008.

[30] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE TIFS*, vol. 11, no. 2, pp. 289–302, 2015.

[31] M. Gebai and M. R. Dagenais, "Survey and analysis of kernel and userspace tracers on Linux: Design, implementation, and overhead," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–33, 2018.

[32] A. Darki, A. Duff, Z. Qian, G. Naik, S. Mancoridis, and M. Faloutsos, "Don't trust your router: Detecting compromised router," in *IEEE CoNEXT*, vol. 16, 2016.

[33] A. Oqaily, Y. Jarraya, L. Wang, M. Pourzandi, and S. Majumdar, "Mlfm: Machine learning meets formal method for faster identification of security breaches in network functions virtualization (nfv)," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 466–489.

[34] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?" in *IEEE INFOCOM*, 2001.

[35] X. Liu, K. Ravindran, and D. Loguinov, “What signals do packet-pair dispersions carry?” in *IEEE INFOCOM*, 2005.

[36] P. L. Dordal, *Linux Traffic Control (tc)*. [Online]. Available: http://intronetworks.cs.luc.edu/current/uhtml/mininet.html

[37] S. K. Khangura, “Neural network-based available bandwidth estimation from TCP sender-side measurements,” in *IEEE PEMWN*, 2019.

[38] F. Ciaccia, I. Romero, O. Arcas-Abella, D. Montero, R. Serral-Gracià, and M. Nemirovsky, “SABES: Statistical available bandwidth estimation from passive tcp measurements,” in *IEEE IFIP Networking*, 2020.

[39] S. K. Khangura and M. Fidler, “Available bandwidth estimation from passive TCP measurements using the probe gap model,” in *IEEE IFIP Networking*, 2017.

[40] V. Kirova, E. Siemens, D. Kachan, O. Vasylenko, and K. Karpov, “Optimization of probe train size for available bandwidth estimation in high-speed networks,” in *MATEC Web of Conferences*, vol. 208. EDP Sciences, 2018, p. 02001.

[41] N. Hu, L. Li, Z. M. Mao, P. Steenkiste, and J. Wang, “Locating internet bottlenecks: Algorithms, measurements, and implications,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 41–54, 2004.

[42] S. Bauer, J. Janelidze, B. Jaeger, P. Sattler, P. Brzoza, and G. Carle, “On the accuracy of active capacity estimation in the internet,” in *IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–7.

[43] S. Bibi, D. Katsaros, and P. Bozanis, “Business application acquisition: On-premise or saas-based solutions?” *IEEE software*, vol. 29, no. 3, pp. 86–93, 2012.

[44] A. H. Anwar, G. Atia, and M. Guirguis, “It’s time to migrate! a game-theoretic framework for protecting a multi-tenant cloud against collocation attacks,” in *IEEE CLOUD*, 2018, pp. 725–731.

[45] Ö. A. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[46] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, “Data mining methods for detection of new malicious executables,” in *IEEE S&P*, 2000.

[47] K. Hahn and I. Register, “Robust static analysis of portable executable malware,” *HTWK Leipzig*, vol. 134, 2014.

[48] K. Shaukat, S. Luo, and V. Varadharajan, “A novel deep learning-based approach for malware detection,” *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106030, 2023.

[49] R. Chaganti, V. Ravi, and T. D. Pham, “Deep learning based cross architecture internet of things malware detection and classification,” *Computers & Security*, vol. 120, p. 102779, 2022.

[50] J.-S. Luo and D. C.-T. Lo, “Binary malware image classification using machine learning with local binary pattern,” in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 4664–4667.

[51] R. Tahir, “A study on malware and malware detection techniques,” *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.

[52] Y. Ji, Q. Li, Y. He, and D. Guo, “Overhead analysis and evaluation of approaches to host-based bot detection,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 5, p. 524627, 2015.

[53] A. Zafeiropoulos, E. Fotopoulou, M. Peuster, S. Schneider, P. Gouvas, D. Behnke, M. Müller, P.-B. Bök, P. Trakadas, P. Karkazis *et al.*, "Benchmarking and profiling 5G verticals' applications: An industrial iot use case," in *IEEE NetSoft*, 2020, pp. 310–318.

[54] P. Munoz, I. De La Bandera, E. J. Khatib, A. Gómez-Andrades, I. Serrano, and R. Barco, "Root cause analysis based on temporal analysis of metrics toward self-organizing 5G networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 3, pp. 2811–2824, 2016.

[55] *NFV deployment–important considerations for operators*. [Online]. Available: https://www.ericsson.com/en/blog/2018/6/nfv-deploymentimportant-consideration s-for-operators

[56] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[57] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *USENIX NSDI*, 2016.

[58] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "Safebricks: Shielding network functions in the cloud," in *USENIX NSDI*, 2018, pp. 201–216.

[59] K. D. Bowers, M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "How to tell if your cloud files are vulnerable to drive crashes," in *ACM CCS*, 2011.

[60] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an SDN with a compromised openflow switch," in *Nordic Conference on Secure IT Systems*. Springer, 2014, pp. 229–244.

[61] R. M. Hinden, “Why take over the hosts when you can take over the network,” in *RSA Conference*, 2014, pp. 1–41.

[62] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, “Software-defined network (SDN) data plane security: issues, solutions, and future directions,” *Handbook of Computer Networks and Cyber Security*, pp. 341–387, 2020.

[63] P. T. Dinh and M. Park, “ECSD: Enhanced compromised switch detection in an SDN-based cloud through multivariate time-series analysis,” *IEEE Access*, vol. 8, pp. 119 346–119 360, 2020.

[64] *Router Bugs Flaws Hacks and Vulnerabilities*. [Online]. Available: https://routersecurity.org/bugs.php

[65] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, “On the treeness of internet latency and bandwidth,” in *ACM SIGMETRICS*, 2009.

[66] *Systemd*. [Online]. Available: https://www.freedesktop.org/wiki/Software/systemd/

[67] *Ubuntu Linux is the Most Popular Operating System in Cloud*. [Online]. Available: https://fossbytes.com/ubuntu-linux-is-the-most-popular-operating-system-in-cloud/#:~:text=simple%20answer%20is.-,Ubuntu%20Linux.,popular%20operating%20system%20in%20cloud.

[68] *Tcpdump*. [Online]. Available: https://www.tcpdump.org/

[69] *SQLite*. [Online]. Available: http://mininet.org/

[70] *Tacker*. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[71] *Mininet*. [Online]. Available: http://mininet.org/

[72] *Open vSwitch*. [Online]. Available: https://www.openvswitch.org/

[73] *NetEm*. [Online]. Available: https://www.linux.org/docs/man8/tc-netem.html

[74] *Ffserver*. [Online]. Available: https://trac.ffmpeg.org/wiki/ffserver

[75] *MPlayer*. [Online]. Available: http://www.mplayerhq.hu/design7/news.html

[76] C. Blake, D. Katabi, S. Katti *et al.*, "Cross-traffic: noise or data?" in *ISMA Bandwidth Estimation Workshop*. San Diego, 2003.

[77] *Cloud native is transforming the telecom industry*. [Online]. Available: https://www.ericsson.com/en/cloud-native

[78] "Open5GS PFCP bug." [Online]. Available: https://research.nccgroup.com/2021/10/06/technical-advisory-open5gs-stack-buffer-overflow-during-pfcp-session-establishment-on-upf-cve-2021-41794/

[79] *Exploiting, Mitigating, and Detecting CVE-2021-44228: Log4j Remote Code Execution (RCE)*. [Online]. Available: https://sysdig.com/blog/exploit-detect-mitigate-log4j-cve/

[80] *CVE-2022-28391 Detail*. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2022-28391

[81] *Trivy*. [Online]. Available: https://aquasecurity.github.io/trivy/dev/

[82] A. Asadujjaman, M. Oqaily, Y. Jarraya, S. Majumdar, M. Pourzandi, L. Wang, and M. Debbabi, "Artificial packet-pair dispersion (APPD): A blackbox approach to verifying the integrity of NFV service chains," in *2021 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2021, pp. 245–253.

[83] E. S. Parildi, D. Hatzinakos, and Y. Lawryshyn, "Deep learning-aided runtime opcode-based windows malware detection," *Neural Computing and Applications*, vol. 33, no. 18, pp. 11 963–11 983, 2021.

[84] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, “Powerful: Mobile app fingerprinting via power analysis,” in *IEEE INFOCOM*, 2017, pp. 1–9.

[85] *Amazon CloudWatch Pricing*. [Online]. Available: https://aws.amazon.com/cloudwatch/pricing/

[86] *Open5GS*. [Online]. Available: https://open5gs.org/open5gs/

[87] *ETSI TS 123 502: Procedures for the 5G System*. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.02.00_60/ts_123502v150200p.pdf

[88] *Code Injection*. [Online]. Available: https://owasp.org/www-community/attacks/Code_Injection

[89] *A Complete Guide to Cloud-Native Application Security*. [Online]. Available: https://www.trendmicro.com/en_no/devops/21/k/a-complete-guide-to-cloud-native-application-security.html

[90] H. Dehling, R. Fried, and M. Wendler, “A robust method for shift detection in time series,” *Biometrika*, vol. 107, no. 3, pp. 647–660, 2020.

[91] *Outliers Detection and Intervention Analysis*. [Online]. Available: https://datascienceplus.com/outliers-detection-and-intervention-analysis/

[92] *LevelShiftAD*. [Online]. Available: https://arundo-adtk.readthedocs-hosted.com/en/stable/notebooks/demo.html#LevelShiftAD

[93] *Anomaly Detection Toolkit*. [Online]. Available: https://adtk.readthedocs.io/en/stable/

[94] *MongoDB*. [Online]. Available: https://www.mongodb.com/

[95] *UERANSIM on GitHub*. [Online]. Available: https://github.com/aligungr/UERANSIM

[96] *Amazon CloudWatch*. [Online]. Available: https://aws.amazon.com/cloudwatch/

[97] *Feature Selection Using Variance Threshold in sklearn*. [Online]. Available: https://lifewithdata.com/2022/03/13/feature-selection-using-variance-threshold-in-sklearn/

[98] B. Liu, J. T. Chiang, J. J. Haas, and Y.-C. Hu, "Coward attacks in vehicular networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 14, no. 3, pp. 34–36, 2010.

[99] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, "DBSCAN: Past, present and future," in *IEEE ICADIWT*, 2014.