Artificial Packet-Pair Dispersion (APPD): A Blackbox Approach to Verifying the Integrity of NFV Service Chains

A S M Asadujjaman*, Momen Oqaily*, Yosr Jarraya†, Suryadipta Majumdar*,
Makan Pourzandi†, Lingyu Wang*, Mourad Debbabi*
*CIISE, Concordia University, Montreal, QC, Canada
Email: {a_asaduj, m_oqaily, majumdar, wang, debbabi}@encs.concordia.ca
†Ericsson Security Research, Ericsson Canada, Montreal, QC, Canada
Email: {yosr.jarraya, makan.pourzandi}@ericsson.com

Abstract—**Network Functions Virtualization (NFV) implements virtual network functions (such as firewall, IDS, etc.) as service chains over a cloud computing infrastructure to provide dynamic, scalable, and cost-efficient network services. This layered design of NFV is a double-edged sword that may also lead to unique security concerns for NFV tenants including the breach of the integrity of their service chains through various attacks (e.g., VNF bypassing, packet injection, etc.). To make things worse, the underlying infrastructure-level data is typically owned by third-party cloud providers, which renders such data unavailable to NFV tenants to directly examine the actual deployment of their service chains. In this work, we propose a blackbox approach, namely, *artificial packet-pair dispersion (APPD)*, which can work around this limitation of unavailable infrastructure-level data to still allow NFV tenants to verify the integrity of service chains. To that end, APPD first estimates the throughput of incoming NFV traffic based on inter-packet delay by creating an artificial congestion (as natural congestion may not always be present in a high bandwidth environment involving cloud and NFV) for a short period of time. APPD then verifies service chain integrity by comparing this estimated throughput with the throughput of the actual traffic flowing through the service chains. Our experimental results with both real and synthetic datasets confirm the effectiveness and negligible overhead of APPD.**

I. Introduction

Network Functions Virtualization (NFV) is considered as one of the cornerstones of the emerging 5G technology due to its various benefits such as cost efficiency and greater flexibility [1]. NFV allows virtual network functions (VNF), such as firewall, 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 [2]–[4]. 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).

978-1-5386-4633-5/18/$31.00 ©2021 IEEE

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 [5]–[8]. Such integrity breaches are mainly caused by misconfigured (e.g., [9]) or compromised (e.g., [10]–[12]) 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 the access to infrastructure-level resource or data*.

Most existing efforts (e.g., [5]–[8], [13]–[15]) fall short to fulfill this need. Specifically, some existing works (e.g., [13]–[15]) 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., [5]–[8]) 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 paper, 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

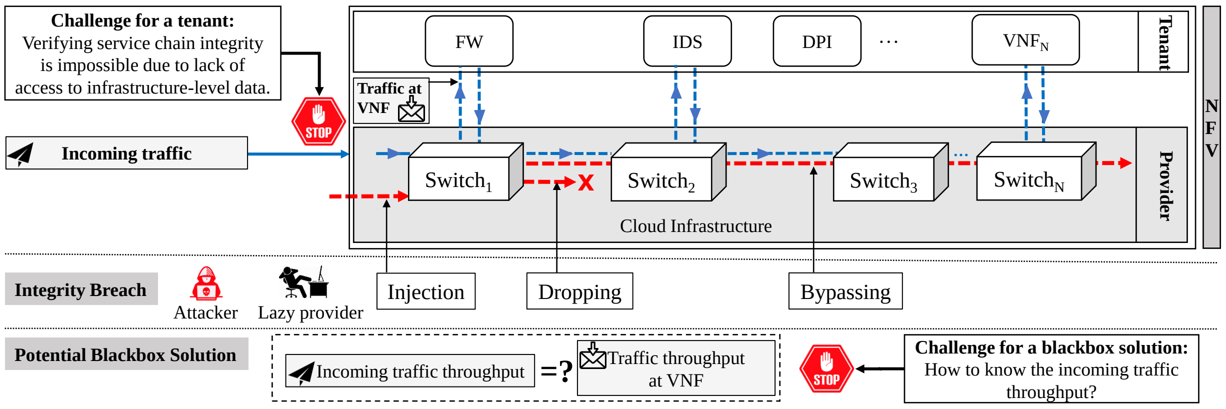

Challenge for a tenant:
Verifying service chain integrity is impossible due to lack of access to infrastructure-level data.
Incoming traffic
FW
IDS
DPI
···
VNF$_N$
Tenant
NFV
Traffic at VNF
Switch$_1$
Switch$_2$
Switch$_3$
Switch$_N$
Provider
Cloud Infrastructure
Integrity Breach
Attacker
Lazy provider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

Fig. 1: An example of integrity breaches in NFV (top) and a possible solution and its key challenge (bottom)

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 a 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 II.

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 create artificial congestion in a high-bandwidth network like in an NFV infrastructure (where a 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.

The remainder of this paper is organized as follows. Section II provides preliminaries. Section III presents detailed methodology of APPD. Section IV describes the implementation details of APPD. Section V presents our obtained experimental results. Section VI describes the literature review. Finally, Section VII concludes the paper.

II. Preliminaries

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

A. Motivating example

The top of Fig. 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. 1 illustrates a potential solution and its key challenge.

NFV Deployment. The top part of Fig. 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 of Fig. 1 shows various integrity breaches including *injection* of fake packets, *dropping* legitimate packets, and *bypassing* one or more VNFs due to misconfigurations (e.g., [9]) by a cheap/lazy provider or attacks by exploiting compromised resources (e.g., [10]–[12]). 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 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 part of Fig. 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

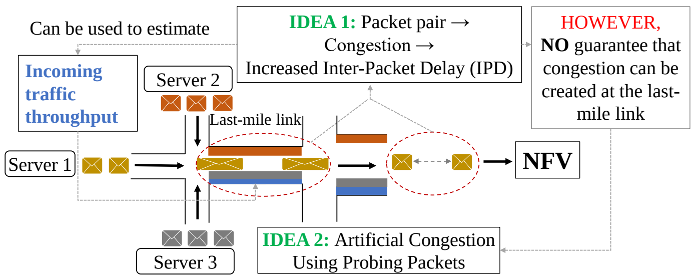

Can be used to estimate
Incoming traffic throughput
IDEA 1: Packet pair → Congestion → Increased Inter-Packet Delay (IPD)
HOWEVER, NO guarantee that congestion can be created at the last-mile link
Server 2
Last-mile link
Server 1
NFV
Server 3
IDEA 2: Artificial Congestion Using Probing Packets

Fig. 2: The main ideas of APPD

to this blackbox solution is: "*How to know the incoming traffic throughput?*".

B. Main Idea

Fig. 2 illustrates our main ideas as follows.

Idea 1: Estimation of Incoming Traffic Throughput. Our first idea is to estimate (instead of measuring) the tenants network traffic throughput by extending the concept of Packet-Pair Dispersion (PPD) [16], 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. 2) are transmitted at a rate that can cause a 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 a 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 V). 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 III will further elaborate on these ideas.

C. 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 [10]–[12]), or (ii) a cheap-and-lazy cloud provider [9] is misconfiguring (intentionally or by mistake) the underlying forwarding devices.

We consider a stronger threat model in comparison to existing works (e.g., [5]–[8]) by including a wide range of attacks and attacker capabilities as follows. (i) *VNF bypassing:*

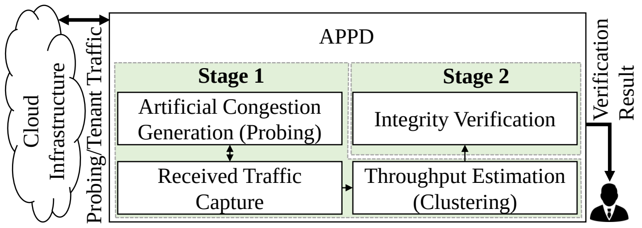


Fig. 3: A high-level overview of APPD

Compromised or misconfigured switches may bypass one or more VNFs in the service chain; compromised switches may also collaborate with each other [17] 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 [17].

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 [18]. 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 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) [19].

III. Methodology

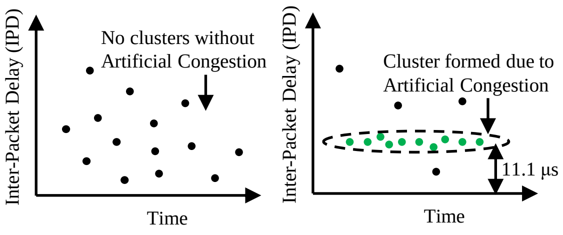
This section presents APPD methodology.

A. Overview

Fig. 3 shows an overview of our methodology, which contains two major stages. Stage 1 performs incoming traffic throughput estimation (detailed in Section III-C), and Stage 2 performs integrity verification (detailed in Section III-D). 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 throughput of the received traffic at a VNF in a service chain.

B. The APPD Effect

The *APPD effect* refers to the formation of distinguishable cluster of IPD values due to the artificial congestion created by APPD (as shown in Fig. 4). These clusters are distinguishable



(a) No artificial congestion (b) Artificial congestion

Fig. 4: Inter-Packet Delay (IPD) clusters are formed due to artificially created congestion (the APPD effect)

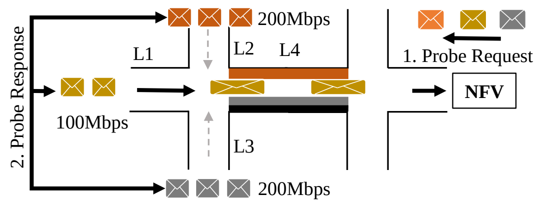


Fig. 5: An example of probe generation

by their high density in the data space as shown in Fig. 4b. Moreover, they have a very specific shape of being spread in horizontal direction and having a very small height. Although the cluster in Fig. 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.

C. 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 an artificial congestion.

Example 1: As shown in the road-junction analogy in Fig. 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, an 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 throughput based on the actual received traffic throughput at VNF (which should be 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 III-B and evaluated in Fig. 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 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. 5, 1,000 probe response packets are generated and received at the first VNF having timestamps $P_1 \rightarrow \text{t}_{P1}$, $P_2 \rightarrow \text{t}_{P2}$, $P_3 \rightarrow \text{t}_{P3}$, ..., $P_{1000} \rightarrow \text{t}_{P1000}$. 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 III-A, under 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 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. 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. 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 $\lambda' = C - T_{P2} = 1Gbps - 500$Mbps = 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.

D. 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 (as shown in Fig. 6) approaches: (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 of integrity verification.

Step 2.1: Cluster-based Verification. For the first VNF in the service chain, verification is performed based on 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 which 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 result 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, $T_E > T_{VNF}$, 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. Which means, the expected throughput may also differ from the actual throughput by δ_T.

IV. Implementation

This section presents the implementation of APPD.

APPD Architecture. There are four major components of APPD (Fig. 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

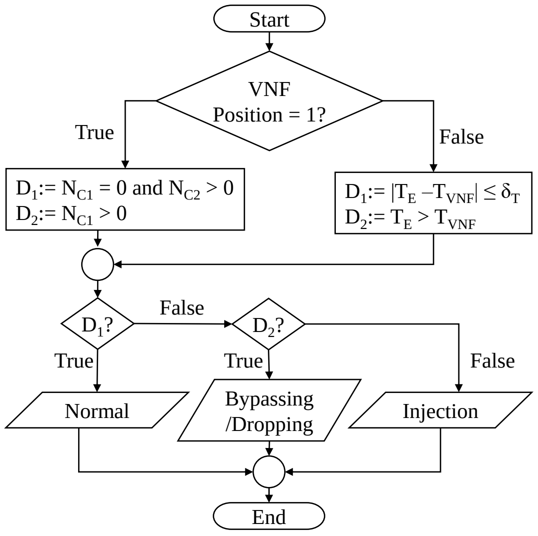


Fig. 6: Integrity verification logic

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 [20]. We choose Linux because it is the most popular operating system in the cloud [21]. 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 [22]), prepares input for the clustering and throughput estimator by reading 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 [23], 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 number of searches by using convex hull. The details of these extensions are not included due to the space constraint.

V. Experiments

This section presents our experimental results.

A. Experimental Settings

To conduct our experiments, we build our NFV testbed using Tacker [24] and OpenStack [25], 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 [26] to set up the tenant network and Internet links (between tenant network and NFV) with virtual hosts, virtual links and Open vSwitch (OVS) [27] 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 tenant network to NFV using the Linux traffic control module NetEm [28]. 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 [29]) or video clients (using MPlayer [30]). 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 [31], hosts outside tenant network act as video clients to stream video from video servers on the Internet.

B. Experimental Results

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

Effectiveness in Verifying Service Chain Integrity. Table I 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 due to the space limitation. In the following, we explain the scenarios listed in Table I. 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

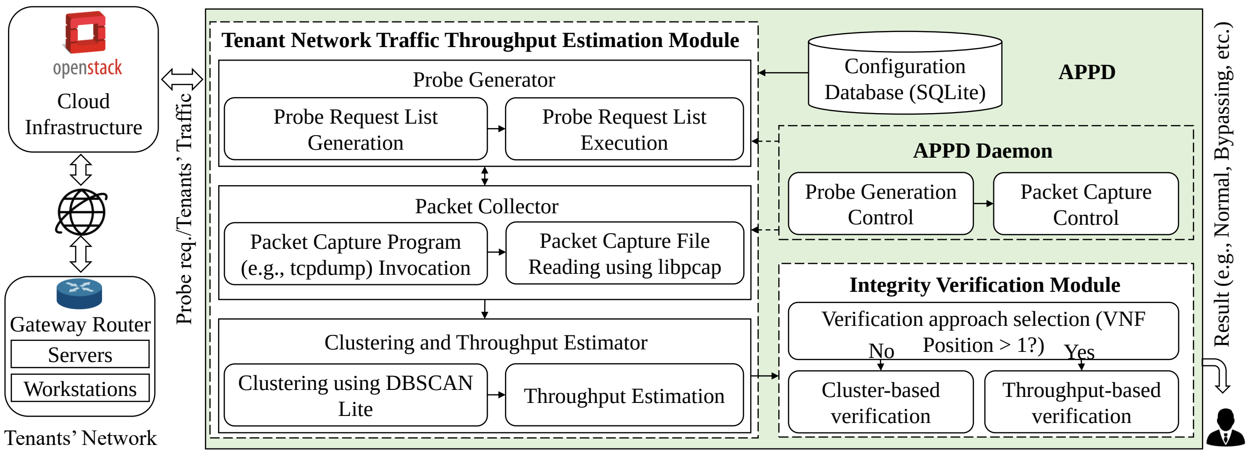
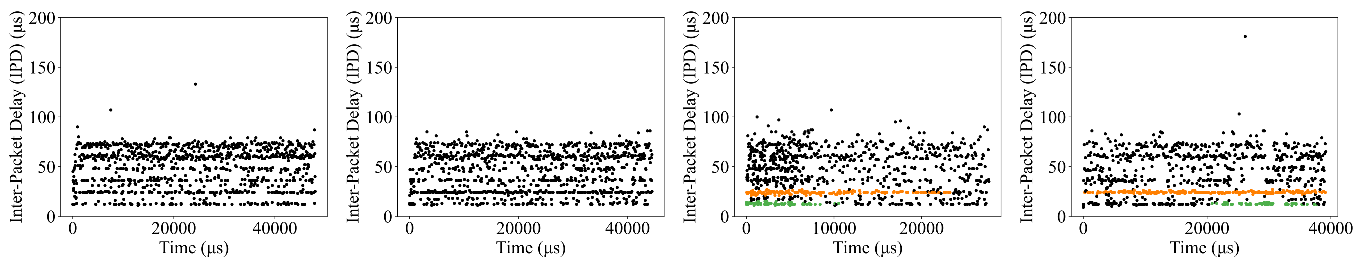

openstack
Cloud Infrastructure
Gateway Router
Servers
Workstations
Tenants' Network
Probe req./Tenants' Traffic
Tenant Network Traffic Throughput Estimation Module
Probe Generator
Probe Request List Generation
Probe Request List Execution
Packet Collector
Packet Capture Program (e.g., tcpdump) Invocation
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.)

Fig. 7: The architecture of APPD

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

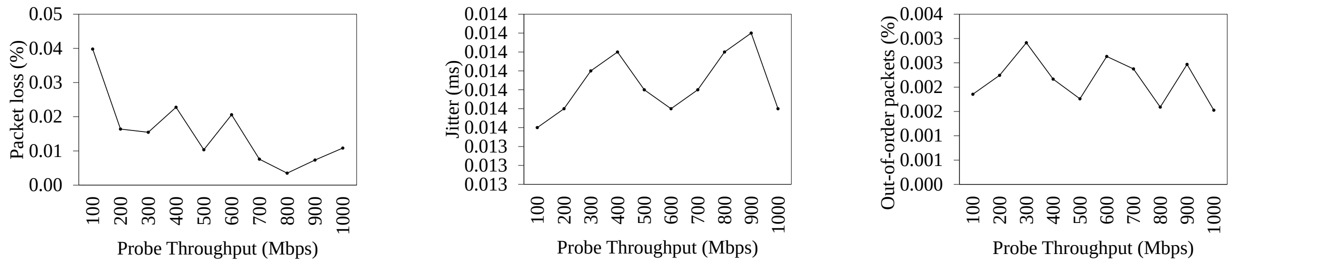
Exp No.	**Experimental Integrity Scenario**	**VNF Position**	**Actual Received Throughput (T_{VNF})**[1]	**Estimated Tenant Throughput (λ')**[1]	**Probe Throughput (T_{P1})**[1]	**Probe Throughput (T_{P2})**[1]	**No. of IPD Clusters (N_{C1})**	**No. of IPD Clusters (N_{C2})**	**Dropped/Rerouted Throughput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



(a) Probe Throughput: 400Mbps (b) Probe Throughput: 440Mbps (c) Probe Throughput: 450Mbps (d) Probe Throughput: 500Mbps

Fig. 8: Applying DBSCAN clustering algorithm on inter-packet delay (IPD); Noise (●), Cluster 1 (●), Cluster 2 (●)



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

Fig. 9: Measuring APPD overhead in terms of network performance metric (packet loss, jitter, out-of-order packets)

$T_{P1} = C - T_{VNF} - \delta_T = 440$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. 8b resulting in $N_{C1} = 0$. However, at $T_{P2} = 500$Mbps, two clusters are found as shown in Fig. 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.
- *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. 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. 8a) and 440Mbps (Fig. 8b) no cluster is formed, and for higher probing throughputs: 450Mbps (Fig. 8c) and 500Mbps (Fig. 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 III. Similarly, APPD expects one or more clusters for its second round of probing ($T_{P2} = 500$Mbps), as calculated from Equation 2 in Section III.

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. 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.

VI. Related Work

Table II summarizes the comparison between existing works and APPD. The first column lists the works. The next two columns indicate different design goals, such as 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) [32] is not needed). The next seven columns list different integrity breaches and threats that are mentioned in Section II-C. In the following, we discuss the existing works in more detail.

NFV Verification. Existing works mainly focus on verification of correct processing and arrangement of VNFs themselves [33]–[36] and little attention has been given to verifying the integrity of service chains. 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 [5]–[8], [17]). Additionally, some works propose on-premise mechanism [17] (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 [5]–[8], [17]. 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 [5]–[8], [17]. This is because existing works need to collect statistics which is available only when the packets arrive at least at the first VNF.

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 [16], [37]–[42]. 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 [43] and cannot provide an in-cloud solution.

VII. Conclusion

This paper 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 resulted by 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

TABLE II: Comparing APPD with existing works

Proposals	Goals		Capabilities						
			Bypass			Drop		Injection	
	In-cloud	Blackbox	Full Service Chain	Partial (except last VNF)	Partial (last VNF)	Before first VNF	After first VNF	Before first VNF	After first VNF
FlowCloak [7]	✓	-	-	✓	-	-	-	-	-
vSFC [6], REV [5]	✓	-	-	✓	-	-	✓	-	✓
Thang et. al. [17]	-	-	✓	✓	✓	-	✓	-	✓
Shin et. al. [13]	-	-	✓	✓	✓	✓	✓	✓	✓
ChainGuard [14]	✓	-	✓	✓	✓	✓	✓	✓	✓
APPD	✓	✓	✓	✓	✓	✓	✓	✓	✓

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.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments and suggestions. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair (IRC) in SDN/NFV Security.

REFERENCES

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

[2] 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.

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

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

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

[6] 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.

[7] 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.

[8] ——, "Securing middlebox policy enforcement in SDN," *Computer Networks*, vol. 193, p. 108099, 2021.

[9] 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.

[10] 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.

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

[12] 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.

[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] 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.

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

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

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

[19] 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.

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

[21] *Ubuntu Linux is the Most Popular Operating System in Cloud*. [Online]. Available: https://tinyurl.com/2wnchhtw

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

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

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

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

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

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

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

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

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

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

[32] 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.

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

[34] 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 Transactions on Network and Service Management*, vol. 14, no. 2, pp. 317–331, 2017.

[35] 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.

[36] 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.

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

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

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

[40] 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.

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

[42] 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.

[43] 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.