Computer Networks 161 (2019) 14–31

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Guiltiness: A practical approach for quantifying virtual network functions performance

Ricardo José Pfitscher*, Arthur Selle Jacobs, Luciano Zembruzki, Ricardo Luis dos Santos, Eder John Scheid, Muriel Figueredo Franco, Alberto Schaeffer-Filho, Lisandro Zambenedetti Granville

Federal University of Rio Grande do Sul, Brazil

ARTICLE INFO

Article history:
Received 16 October 2018
Revised 20 February 2019
Accepted 3 June 2019
Available online 14 June 2019

Keywords:
Network functions virtualization
Service function chaining
Performance quantification
Bottleneck detection

ABSTRACT

In Network Functions Virtualization (NFV), service providers create customized network services by chaining Virtual Network Functions (VNFs) in forwarding graphs, according to individual client demands. Despite the flexibility promoted by the NFV paradigm, specific VNFs used in network services may become bottlenecks due to a number of factors, *e.g.*, lack of resource capacities, demand overload, incorrect ordering, and interdependency between VNFs. Hence, resource monitoring is crucial to determine which VNFs have a negative impact on the quality of service. However, NFV imposes constraints that hinder the adoption of traditional network monitoring approaches, such as: the heterogeneity of environments with all sorts of VNF purposes (*e.g.*, caching, address translation, and performance optimization), and the atypical functioning of specific VNFs that rely on non-blocking I/O implementations. In this paper, we propose a model to quantify the `guiltiness` of each VNF on degrading the performance of a network service. This model consists of a novel application of the *Utilization Law* from queuing networks theory. Through numeric assessments on typical scenarios, we refined the set of relevant resource metrics and applied a weighted sum to gauge them in the model. Also, a hybrid algorithm based on linear regression and neural networks is introduced to adjust the model's parameters according to the environment particularities, such as the type and number of VNFs in the service. Experimental evaluations confirm the ability of the model to (*i*)detect bottlenecks, and (*ii*)quantify performance degradations. When capacity restrictions were applied to different types of VNFs, our `guiltiness` metric was able to detect the root cause of degradations. Further, the `guiltiness` metric outperformed traditional metrics, being able to characterize 96.15% of the performance issues with a 25.6% reduction in the number of false positives when compared to CPU usage.

1. Introduction

Network functions perform an important role in the communication between clients and end-services in computer networks. These functions are employed to achieve numerous objectives, such as providing security, addressing, routing, caching, and load balancing [1]. In traditional networks, network functions run in dedicated hardware middleboxes, which restricts the development of novel solutions to specific vendors, impacting negatively on the CApital EXpenditure (CAPEX) and on the OPerational EXpendinture (OPEX) of companies [2,3]. Recently, the virtualization trend that drove cloud computing has also been applied to network functions, allowing Virtual Network Functions (VNFs) to run in Commercial-Of-The-Shelf (COTS) servers, decoupled from the underlying hardware [4]. In this context, both industry and academia have been spending efforts in the development and standardization of Network Functions Virtualization (NFV). The European Telecommunications Standards Institute (ETSI) defined an Industry Specification Group (ISG)[1] to study, standardize, and develop NFV; in another movement, the Internet Research Task Force (IRTF) created the NFV Research Group (NFVRG)[2] to deal with NFV-related research problems.

* Corresponding author.
E-mail addresses: rjpfitscher@inf.ufrgs.br (R.J. Pfitscher), asjacobs@inf.ufrgs.br (A.S. Jacobs), lzembruzki@inf.ufrgs.br (L. Zembruzki), rlsantos@inf.ufrgs.br (R.L. dos Santos), ejscheid@inf.ufrgs.br (E.J. Scheid), mffranco@inf.ufrgs.br (M.F. Franco), alberto@inf.ufrgs.br (A. Schaeffer-Filho), granville@inf.ufrgs.br (L.Z. Granville).

https://doi.org/10.1016/j.comnet.2019.06.001
1389-1286/

[1] https://www.etsi.org/technologies-clusters/technologies/nfv.
[2] https://datatracker.ietf.org/rg/nfvrg/.

One of the use cases proposed by ETSI refers to the ability of VNFs being chained and provisioned on demand, bringing elasticity and dynamicity to the network [5]. In particular, forwarding graphs can be used to specify service chains, a pre-established order of VNFs that network packets must traverse. Further, these service chains can be provided as customized network services to meet individual client demands [6,7]. For instance, an NFV Service Provider (NFV SP) may offer a forwarding graph portfolio, including chains for network performance improvement, security controlling, and IP Multimedia Subsystems (IMS) implementation. In these service chains, each VNF has a certain impact on the overall network service performance, depending on several factors (*e.g.*, implementation, provisioned resources, and chaining order) [8]. Hence, network operators should be able to quantify this impact to determine which VNFs are degrading the performance of the chain. Also, even if the network is not experiencing any performance degradations, the knowledge of how each VNF affects network performance can help operators in planning decisions, since providing more resources to VNFs with a higher impact on performance will likely enhance Quality-of-Service (QoS) and Quality-of-Experience (QoE).

Usually, operators rely on monitoring physical and virtual resources to point out VNFs that are causing performance issues. This detection method consists of monitoring Operating System (OS) level metrics (*e.g.*, CPU usage, memory usage, and packet loss) and comparing them to pre-established thresholds. When a given measurement surpasses such threshold, the monitoring system triggers an event to inform the network operator that the VNF is hindering the performance of the service chain. However, the effectiveness and accuracy of such threshold-based approaches depend on the correct prior calibration and specific knowledge about the resources that each VNF in the service chain consumes. These preliminary adjustments limit their employment in NFV environments due to the presence of a variety of network function and service types [4,9,10]. For example, while the performance of VNFs in an IMS implementation can be characterized using solely CPU, and memory usage metrics [11], the performance of a VNF that provides a non-blocking I/O video transcoder function may require the monitoring of resources queues due to the difficulty in distinguishing processing CPU cycles from busy-waiting ones [12].

One approach commonly applied to quantify performance and detect bottlenecks is to represent the various components involved in communication through analytical models [13–15]. For instance, in queueing networks, each node is modeled as a queue and a processing component: the queue stores the requests that are processed and forwarded to the next node. In this approach, historical information on resource usage is applied in mathematical models to predict the residence time (*i.e.*, time taken to process a request) in each node; then, to quantify performance impacts, one must compare the predicted residence times. However, since the VNFs of a service chain intermediate the communication between clients and end-services, performance degradations must be detected in near real-time [16]. This requirement limits the traditional use of mathematical models, because of its offline nature. In the context of NFV, both threshold monitoring [11,17,18] and analytical modeling [19–21] have been exploited in recent literature to detect bottlenecks. However, such efforts require low-level instrumentation of the network, or even a pre-monitoring calibration step, to accurately identify bottlenecks. These requirements hinder the feasibility of bottleneck detection, as they are most commonly impossible to meet in real-world scenarios, which include multiple stakeholders that may have restricted access and control over networks.

In a previous work [22], we propose a first model to circumvent such limitations based on analysis of security-based service chains, which consists in a weighted-sum of three metrics: CPU usage, CPU activity, and NIC queuing. Although this model has shown to be useful in the performance diagnosis of security-related service chains, it did not consider metrics, such as I/O, and memory usage, which later proved to be relevant in diagnosing different scenarios. Thus, in this paper, we extend the model to propose a metric able to quantify the performance impacts that each VNF has in general-purpose service function chains. Our first step towards this objective was to assess the *representativeness*[3] of state-of-the-art metrics in distinct scenarios. Through empirical observations, we reduce the set of monitoring metrics to a subset containing only metrics representative of performance in multiple scenarios (*i.e.*, CPU usage, memory usage, Network Interface Card (NIC) usage, I/O usage, and NIC queueing). Then, by introducing the activity of resources in concepts from queueing network modeling, we combine the subset of metrics in a weighted-sum, defining the `guiltiness` of each VNF on degrading the service chain performance. Also, to increase the accuracy of the model, we propose a learning algorithm that takes advantage of non-linear regressions and neural networks to adjust the model according to the relevance of each metric in the NFV environment.

In summary, this paper presents the following contributions:

- A novel, practical, mathematical model that is able to accurately identify bottlenecks and quantify performance degradation for individual VNFs in a service function chain. With this model, operators are able to determine which VNFs have higher impact on end-service performance, and hence more precisely act to improve user experience by either optimizing resource provisioning or preventing errors with informed planning decision.
- An experimental characterization of NFV performance in distinct scenarios, providing an extensive analysis of the most commonly used monitoring metrics. With this characterization, we present a detailed evaluation of how much each monitoring metric is able to accurately identify and quantify performance degradation in NFV environments.

The remainder of this paper is organized as follows. Section 2 discusses the state-of-the-art on NFV performance quantification and bottleneck detection, giving a summary of approaches and metrics used in the literature. Then, in Section 3, we assess the representativeness of monitoring metrics. For doing so, we analyze experimental results from state-of-the-art evaluation scenarios. Next, Section 4 develops the modeling of the `guiltiness` metric, and also introduces the learning mechanism. Section 5 then evaluates the ability of `guiltiness` to both detect bottlenecks and quantify performance degradations. Finally, we present our conclusions and future research directions in Section 6.

2. Related work

Research efforts have already been carried out on the two NFV-related research areas addressed by this paper: bottleneck detection and performance quantification. Table 1 summarizes such research efforts.

Bottleneck detection can be based on three approaches: *threshold monitoring, customized metrics*, and *iterative resource provisioning*. Threshold monitoring approaches, such as the one we previously proposed [17], are commonly used to detect resource contention-related performance degradations. This approach consists in monitoring a set of metrics and comparing them to thresholds in order to establish status levels. When the measurements cross predefined levels, operators are notified. Other studies propose *customized metrics* for particular scenarios. Wu *et al.* [12] propose the use of network queueing and packet loss observations to

[3] We define the term *representativeness* as a measurement of metrics' relevance in expressing performance degradation of service chains.

Table 1
State-of-the-art approaches and metrics.

Reference	Approach	Metrics of the model
Pfitscher et al. [17]	Threshold monitoring	CPU usage, and steal time
Wu et al. [12]	Customized metric	Network queuing, and packet loss
Naik et al. [23]	Customized metric	Packet processing time, and throughput
Gember et al. [18]	Threshold monitoring, and iterative provisioning	Packet processing time, and CPU usage
Cao et al. [11]	Threshold monitoring, and iterative provisioning	CPU usage, NIC usage, and memory usage
Gallardo et al. [19]	Analytical model	CPU usage, and arrival rate
Suksomboon et al. [20]	Analytical model	CPU usage, NIC usage, cache misses, and arrival rate
Sauvanaud et al. [24]	Pattern recognition	Black-box, and gray-box
Koizumi and Fujiwaka [21]	Analytical model, and statistical regression	CPU usage, NIC usage, and memory usage

detect resource contention situations. Their solution consists on instrumenting VNFs to measure resource's queues, and this points out bottlenecks on specific resources. *Iterative resource provisioning* combined with resource monitoring to search for bottlenecks is another approach, as proposed by Gember *et al.* [18]. A bottleneck is pointed-out if the processing time of packets increases in a time window or if memory and CPU usages exceed a threshold. Also, an approach for scaling-up and scaling-down resources of each VNF is applied to observe whether the provisioning action resulted in improvements or deterioration of the overall chain performance, where the most impactful VNFs are considered bottlenecks. Similarly, Naik et al. [23] use packet processing times and throughput of VNFs to detect bottlenecks. By mirroring VNFs' network interfaces, a central management instance computes and learns the performance profiles of each VNF. Then, if an abnormal behavior occurs in one instance, the bottleneck is pointed out. Another option is to combine multiple methods. Cao et al. [11] propose NFV-VITAL framework for performance characterization of general NFV scenarios. The approach combines both *threshold-based* and *iterative resource provisioning* approaches by stressing the chain with distinct workloads and VNF sizes, monitoring resources in each VNF, and comparing thresholds to detect bottlenecks.

The performance quantification problem has been well stressed in the computer networks literature. In terms of its application to NFV, the three major approaches consist of: developing *analytical models*, performing *statistical regressions*, and applying *pattern recognition* techniques. Gallardo et al.[19] propose an *analytical model* for the performance evaluation of virtual switches. Although switches are network appliances intrinsic to the network communication, virtual switches have been considered VNFs because of their flexibility in resource allocation. In the proposed solution, RX queues of each vNIC and each CPU core are modeled as the elements that compose a virtual switch. Then, the performance analysis is applied to the underlying computation of the global system, regarding CPU core utilization, loss rate, sojourn time of packets, and buffer occupancy. The main strategy of the model is to decompose the system into a set of independent queuing models, and to introduce service vacations models. Vacations refer to the in-between packets time, *i.e.*, the time where the server was collecting the next packet from the RX queue before processing it on CPU. Through measurements of arrival rates, the model uses probabilities to estimate the missing parameters and, as a consequence, estimate the global system performance.

Suksomboon et al. [20] also use *analytical modeling* to predict the performance of software routers. The authors propose predicting the maximum throughput a software router can achieve for packet forwarding, according to its allocated resources. The paper considers three factors as influencing in the performance: Ethernet bandwidth, CPU speed, and shared Last Level Cache (LLC). Instead of using queueing theory, the model for estimating VNF throughput is based on a mathematical formulation that includes cache contention as a factor of performance degradation. Cache contention occurs when *corunner* VNFs compete for the same core, resulting in cache misses. In turn, Sauvanaud et al. [24] propose a solution for anomaly detection and root cause identification based on *pattern recognition* techniques. The proposal combines black-box and gray-box monitoring data with machine learning to detect anomalous situations. The root cause detection of problems derives from a database trained with both monitoring metrics and SLA violations. A detection model compares the behavior of VNFs through a knowledge database to result in a classification probability. Next, a probability threshold defines whether a VNF is performing anomalously: if true, such a VNF is probably the root cause of problems.

Finally, Koizumi et al. [21] combine *statistical regressions* and *analytical modeling* to develop a model for estimating NFV service chain performance. The proposal takes into account the *processing overlap* to predict the execution time. Processing overlap is the time taken by an execution node (*i.e.*, VNF) to process distinct services simultaneously. The performance estimation process integrates a Colored Timed Petri Net (CTPN) model and statistical regressions to compute the node's processing time dynamically. In this process, historical information from CPU, memory, and network usage are applied in multiple regression models to estimate parameters of the mathematical formulations.

Given the state-of-the-art, and the summary presented in Table 1, one can notice that different metrics (traditional and customized) have been considered for bottleneck detection and performance quantification in NFV. A set of major concerns arise when solving these problems simultaneously: (*i*)threshold monitoring approaches [11,17,18] are not able to quantify performance degradations and require a calibration step to accurately diagnose bottlenecks; (*ii*)customized metrics depend on low-level instrumentation of operating system queues [12], or need access to network devices to perform mirroring [23]; (*iii*)iterative provisioning can delay the bottleneck diagnosis process, and fail to quantify degradations; (*iv*)analytical models [19–21] require a deep knowledge of each type of VNF and their internal communications; and (*v*)analytical models, pattern recognition, and statistical regression need a large set of results before providing precise estimates of performance. Nonetheless, the set of available metrics and VNF types are heterogeneous, and this increases the complexity, for operators, to decide which metrics they must monitor. We argue in favor of an approach based on a customized metric integrated with an *online* learning mechanism able to weigh the importance of distinct resources in each VNF type. Thus, in the next sections, we discuss the representativeness of state-of-the-art metrics in multiple scenarios to then propose a novel customized metric which we call `guiltiness`.

3. Metrics representativeness

The first step towards developing a model able to quantify performance degradations is to understand the representativeness of monitoring metrics. As discussed in Section 2, diagnosing metrics are so diverse as the kind of VNFs that are available in NFV sce-

narios. Wu et al.[12] points out that some VNFs use non-blocking I/O call implementations, which imply in 100% of CPU usage. Similarly, Cao et al.[11] show that overloaded VNFs can exhibit fluctuations on CPU usage. In turn, Naik et al. [23] argue in favor of computing packet processing times because resource usage information are inconclusive. Taking this into account, in this Section, we perform a set of experimental evaluations to verify how the state-of-the-art metrics behave in diversified scenarios.

3.1. Monitoring metrics

Based on the state-of-the-art and the well-accepted metrics from the general field of computer networks, we select the ones that depict information about each VNF individually. For this reason, we neglect from the analysis metrics whose meaning depends on multiple VNFs (*e.g.*, delay and jitter). The resultant set of metrics is divided into four categories:

- *network-related:* NIC usage and NIC queueing;
- *memory-related:* resident memory usage, committed memory usage, and cache hit/miss rate;
- *processor-related:* CPU usage, steal usage, and active percentage time; and
- *io-related:* I/O transactions per second.

Network-related metrics are usual indicators of performance degradations. NIC throughput denotes the amount of data, or amount of requests, that one VNF can process in a time interval [25]. The intuition is that the VNF with the highest throughput in the chain probably retains a higher incidence of network flows. Hence, such VNF is more likely to impact the SFC performance when stressed. As a consequence, NIC usage computes the rate between NIC throughput and the link bandwidth over time. The greater the NIC usage in the VNF, the greater the chance of the VNF being a bottleneck. NIC queuing occurs in two situations: when the VNF's transmission demand is higher than the available bandwidth, datagrams are enqueued on the output buffer of the virtual network interface; and when the reception rate is greater than the processing throughput, queueing occurs in the input buffer [26]. Intuitively, if one detects that there is queuing occurring at a node, then it may be a bottleneck. Packet drops occur just after queueing, when a buffer is full, so incoming requests are dropped. If packets are being dropped in a given VNF, it is because one of the resources inside the VNF is saturated, and thus, the VNF may be a bottleneck [12].

Concerning memory usage in VNFs, we analyzed the representativeness of three metrics: resident percentage, committed percentage, and cache hit/misses. The resident memory represents the amount of memory that is effectively in use by the running process and the operating system. Resident memory is given by the subtraction of free memory, cache and I/O buffer from the total physical memory. In turn, the committed memory expresses the amount of memory allocated for a given process [27]. As applications do not use all the requested memory, modern operating systems use virtual memory management techniques to reserve more than the memory available in the physical machine, and thus, it is common to observe more than 100% of committed memory. Hence, if a machine presents high usage percentages on these metrics, the VNF's performance decreases, which can result in flow contention. In addition to memory usage, statistics about cache hits/misses can provide meaningful information about the status of VNFs. When a system exposes a high rate of cache misses, this could indicate a cache contention situation, and thus, the VNF can be a bottleneck in a service chain [20].

Among processor-related metrics, which are most commonly resorted to as performance indicators, we analyzed the three most prominent ones: CPU usage, steal usage, and active percentage time. The CPU usage represents the percentage of time that a processor is busy during a time interval. Two aspects suggest an investigation about CPU usage. First, CPU occupation is a signal of resource stress: the higher the usage, the greater the likelihood that the system cannot support an increasing demand. Second, using this metric, alongside the network throughput, one can predict the VNF's service demands through analytic model laws, such as the Utilization Law [15,25]. The steal time metric represents the portion of time that a virtual machine waits to run its processes while virtual CPU was not allocated due to physical resource contention [28]. The reasoning behind this metric is that if steal occurs, the performance of the host decreases, which in turn affects the VNF too. Finally, active percentage time refers to the average amount of time a given resource — in this case, CPU — is active in an observed time series. This metric is derived from the active duration time metric [29], which accounts for the period of time that a resource is in a non-idle state. Thus, regarding bottlenecks, the more active is a node, the higher is its performance impact on the service chain processing result. For an in-depth discussion about active time, refer to Section 4.

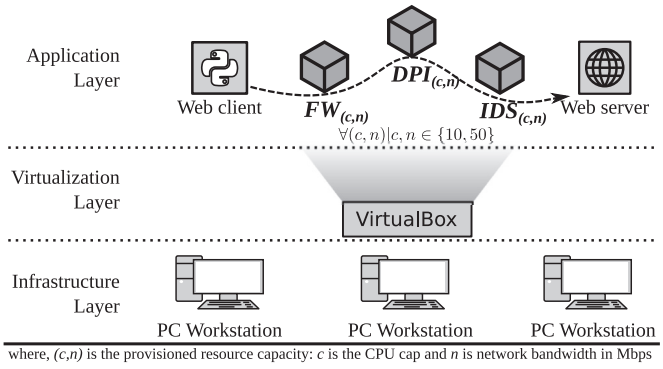


Fig. 1. Scenario 1 – Security-based VNFs with network capacity restrictions.

Lastly, we asses an I/O-related metric in our evaluation. None of the works discussed in Section 2 consider I/O-related metrics in their analysis. This is because VNFs are essentially network packet forwarders, which means that they receive packets, process them, and forward to the next node; such process does not require storing information about network flows. However, several VNFs rely on I/O operations, such as when an Intrusion Prevention System (IPS) access a database to detect malicious flows, or when an IMS component registers incoming calls. Thus, we include the I/O operations per second metric in our analysis. Instinctively, the behavior of this metric denotes that the closer this resource comes to its limit, the higher the probability of it causing a bottleneck.

To properly assess the representativeness of each metric, we create scenarios that closely resembles the ones used in the state-of-the-art. In the following subsections we discuss the representativeness results for each of these scenarios.

3.2. Capacity constrains in a security chain

Four works [17,18,21,22] performed experiments on a *security-based* scenario. Our first scenario (Fig. 1) reproduces these evaluation efforts, and consists of three security VNFs — a firewall (FW), implemented through `iptables`[4]; a DPI implemented through nDPI[5], and an Intrusion Detection System (IDS), implemented through Suricata[6] — with two levels of network capacity, 10 Mbps

[4] https://linux.die.net/man/8/iptables.
[5] https://www.ntop.org/products/deep-packet-inspection/ndpi/.
[6] https://suricata-ids.org/.

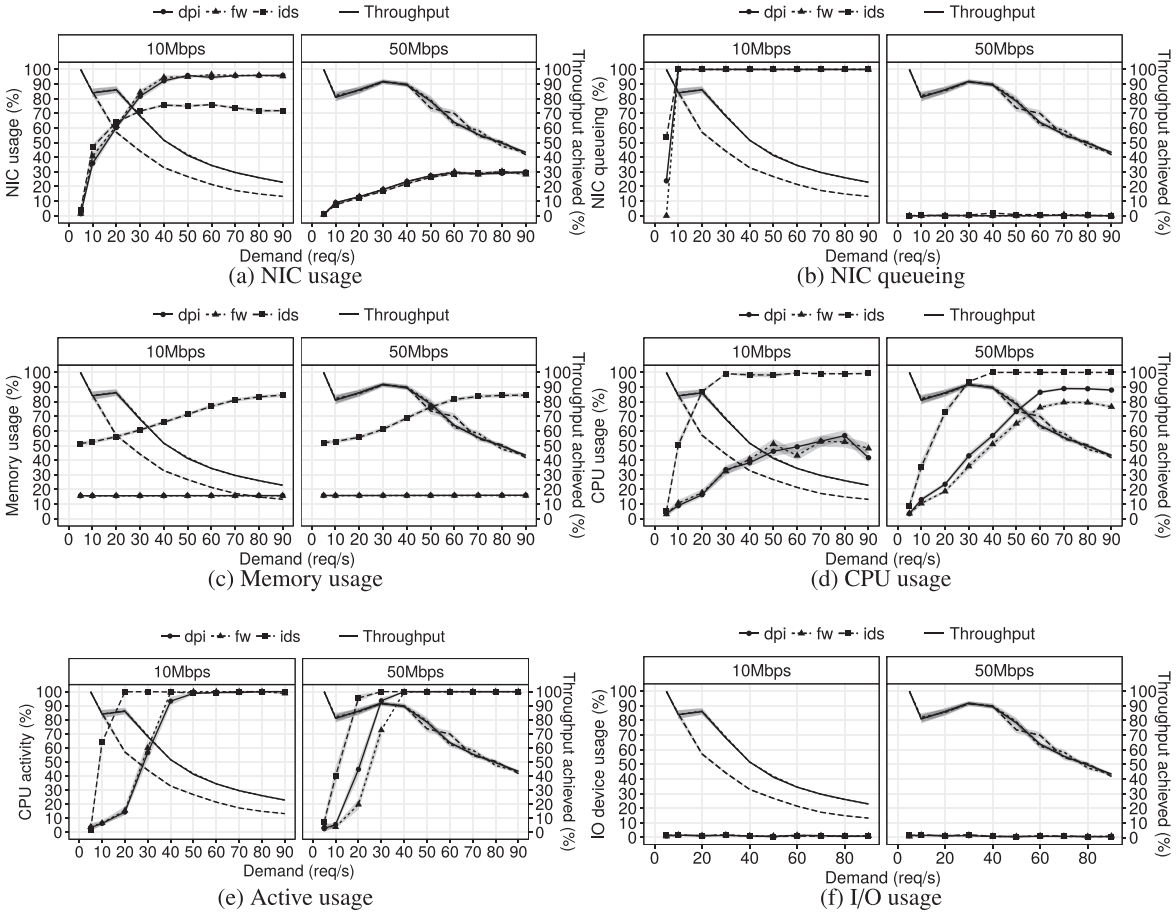


Fig. 2. Monitoring metrics in a security chain with capacity constraints.

and 50 Mbps. The VNFs were configured to provide an increasing level of inspection, ranging from the FW to the IDS. To observe the behavior of metrics, we submitted an increasing demand of Web requests passing through the service chain. The client consists of a Python application with four threads, which creates multiple TCP flows in an amount proportional to the number of requests. Hence, at a specific timestamp, distinct VNFs can be a bottleneck of the Web server performance. The amount of performance degradation varies according to the characteristics of the packet inspection and to network capacity. The deeper the analysis of packets, the higher is resource consumption, and as a consequence, the delay in processing requests. Similarly, as the provisioned network capacity gets more restricted, the end-to-end throughput decreases.

This scenario was evaluated with standard PC workstations with 8 CPU cores and 16 GB of memory, running Debian 9.3 (Stretch). VirtualBox was used as the hypervisor for the Virtual Machines (VM) that run clients, servers, and VNFs. Each VM runs an Ubuntu server 16.04 LTS and was configured with a single core and 512 MB of memory, except for the IDS that has 1 GB of RAM. Static routes are used to forward the traffic from clients, passing through the VNFs, to the servers. We use the application throughput as end-service metric.

Fig. 2 depicts the measurements of monitoring metrics from the chained VNFs. Each line/point in the charts of Fig. 2 depicts the average of 30 repetitions and the computed standard error with 95% confidence level. We expect this experiment to reflect a major impact from the FW and IDS, since the FW is the first VNF in the chain, and the IDS has a deeper level of inspection. It is also expected that the throughput increases from 10 Mbps to 50 Mbps network capacity. This means that the effect of network restriction in throughput occurs with a higher demand of requests with 50 Mbps than with 10 Mbps. The gray lines depict the application throughput measured when distinct VNFs are bottlenecks, where the gray dashed line refers to the throughput obtained when the restrictions are imposed to the IDS. As depicted in the results, the end-service performance behaves as expected: when provisioned capacity is of 10 Mbps the application throughput starts to fall down at a demand of 20 requests per second, while with 50 Mbps the throughput decrease starts at 40 reqs/s. It is important to highlight that bottlenecks imposed to the IDS has a greater impact then when imposed to the other two VNFs.

Regarding the extent of metrics representing application performance, NIC usage (Fig. 2a), NIC queueing (Fig. 2b), and CPU usage (Fig. 2d) are the best candidates for characterizing the performance of the security chain. When network capacity is restricted, the three VNFs exhibit NIC queuing, thus being a good signal of bandwidth congestion. As we add more capacity to each NIC, the queuing on NIC interfaces almost ends and NIC usage reduces. However, the application throughput still experiences a drop even with the scale in network capacity. The behavior of CPU usage help to explain this throughput decrease: when provisioned network capacity is of 50 Mbps, the VNFs' CPU usage increases with the demand, and when the IDS reaches around 90% the fall on application throughput starts. On the flip side, memory usage, active usage, and I/O usage do not aid in the performance diagnosis. As the demand of requests increases, the IDS is the unique VNF that exhibits an increasing memory usage, going from 50% to around 90%. In the case of CPU activity and I/O usage, their behavior remains

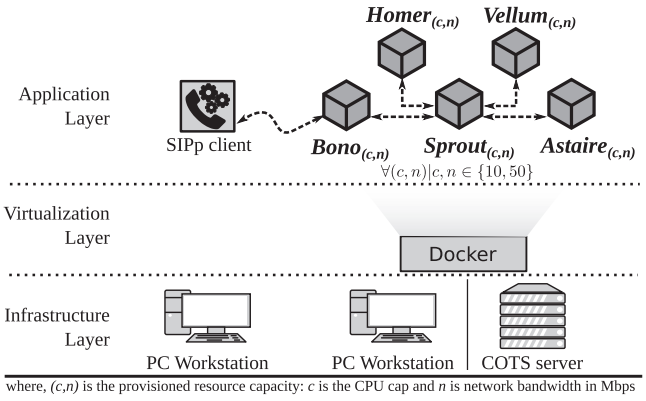


Fig. 3. Scenario 2 – Clearwater IMS with network capacity restrictions.

indifferent to network capacity, and the measurements do not express the variances of application throughput.

Despite the fact that these three metrics are useful for diagnosing application performance, the challenge of establishing thresholds for them remains open. For example, in the case of NIC usage, should one consider amounts of resources occupation superior to 50% as a warning signal? Regarding CPU usage, is 80% acceptable for signalizing degradation? Such questions give rise to doubt about using threshold-based approaches, as in [11,17,18], for bottleneck identification purposes. Nonetheless, it is worth highlighting that none of the other evaluated metrics exhibits a behavior able to explain the variances of application throughput.

3.3. Capacity constrains in Clearwater IMS

The second scenario is an implementation of the internal components of an IMS, as used for the evaluations presented in [11,23,24]. The Clearwater project[7] published an open-source implementation of the IMS core standard [30]. Clearwater takes advantage of a load-balancing DNS to provide dynamic horizontal scaling for all internal components, which aids the deployment of evaluations scenarios that require parallel chaining.

Fig. 3 depicts the Clearwater architecture and its relation with clients and the NFV infrastructure. In this scenario, IMS clients perform multimedia calls to communicate through the Internet. For these IP-based communications to be successful, traditional telecommunication implementations rely on well defined protocols, such as Session Initiation Protocol (SIP), and a standardized architecture with components to initiate and manage calls. IMS functions consist of software packages implemented in VMs or containers as depicted in Fig. 3. Also, an embedded DNS is provided by the docker host for load-balancing of client calls.

Each of the internal components of the current Clearwater IMS implementation are described below:

- **Bono** - Bono components are the Proxy-Call Session Control Function (P-CSCF) standard interface to IMS clients, and they serve as edge proxies providing access points for the client's connection to the Clearwater system.
- **Sprout** - This component implements the Serving Call State Control Function (S-CSCF) and Interrogating Call State Control Function (I-CSCF) interfaces of the IMS standard, and it provides a scalable SIP register and authoritative routing proxy.
- **Homer** - Homer is an XML Document Management System (XDMS) responsible for storing settings for each user of the system in XML documents. Documents are managed using a standard XML Configuration Access Protocol (XCAP) interface.
- **Vellum** - Vellum is the component responsible for maintaining long-lived state in the deployment. It includes multiple distributed storage clusters: *Cassandra* for storing authentication credentials and profile information, *etcd* for sharing internal Vellum configurations, *Chronos* for reliable time service, and *Memcached* for storing registration and session states.
- **Astaire** - Astaire components help provide elasticity to Clearwater. When nodes experience scaling up/down actions, Astaire is responsible for synchronizing data across all nodes that use *Memcached* as a database. The Astaire node runs as a daemon in background, acting when triggered by a "reload" request. When requested, it pulls data from the *Memcached* servers that have data that must live locally and push them to the local *Memcached* server.

For the sake of fairness of results, we use the same workload as NFV-VITAL [11]. To simulate scaling up actions, in addition to the previous standard PC workstation, we run a Clearwater docker implementation[8] on top of a COTS server with 64 CPU cores and 64 GB of memory running an Ubuntu server 16.04 LTS. By doing this, we can understand the effects of memory and CPU capacities over the analyzed metrics. Also, akin to the previous scenario, we provisioned two levels of network capacity (*i.e.*, 10 Mbps and 50 Mbps) between IMS internal components. The Home Subscriber Server (HSS) database is populated with 50k subscribers and the IMS system is subjected to an increasing demand of REGISTER requests (ranging from 2^5 to 2^{12}), generated by `sipp`[9]. Each request flow is as follows: a client makes a non-authenticated REGISTER message; then, it receives a 401 – authentication fail response from the server; next, the client sends the REGISTER message with valid authentication parameters and waits for a 200 – OK response from the server. Because of the lack of isolation between containers, it is worth mentioning that using a container-based implementation of VNFs compromises the individual accounting of three metrics: CPU steal usage, committed memory, and cache miss rate.

To provide a reliable service to the largest number of users possible, Clearwater implements a best-effort mechanism based on a token bucket. In such a mechanism, each request to the system consumes a token from the limited bucket, and if the bucket is empty, then the request is rejected. Thus, instead of processing every incoming request, some of them are rejected as the system reaches its limit. Each node estimates this limit by monitoring the latency of requests to update the token replenishment rate. This rate is adjusted periodically: every twenty requests, the node compares the current latency to the previous measurement. If there is an increase, then the token refill rate decreases; if the latency decreases, then the rate increases [11,31]. The consequence, in our experiments, of such a token bucket mechanism is that the throughput sustained by the system varies according to the hardware capacity.

Fig. 4 depicts the observed metrics of Clearwater running on top of a PC workstation. To have an objective comparison between scenarios, we also plot the application throughput as end-service metric. The results show that network capacity does not influence the application throughput, and that Clearwater is able to sustain up to 2^8 calls/s. As we increase the available bandwidth from 10 Mbps to 50 Mbps, the call throughput maintains exactly the same levels. With a demand of 2^{11} calls/s, Clearwater throughput is only able to cope with around 20% of submitted requests, signaling a severe drop in performance. Since the workload to which we submitted the scenario is composed solely of REGISTER requests, Sprout and Bono nodes are the more likely to impact on service performance, given that Bono is the entry point for clients

[7] http://www.projectclearwater.org/.

[8] https://github.com/Metaswitch/clearwater-docker.

[9] http://sipp.sourceforge.net/.

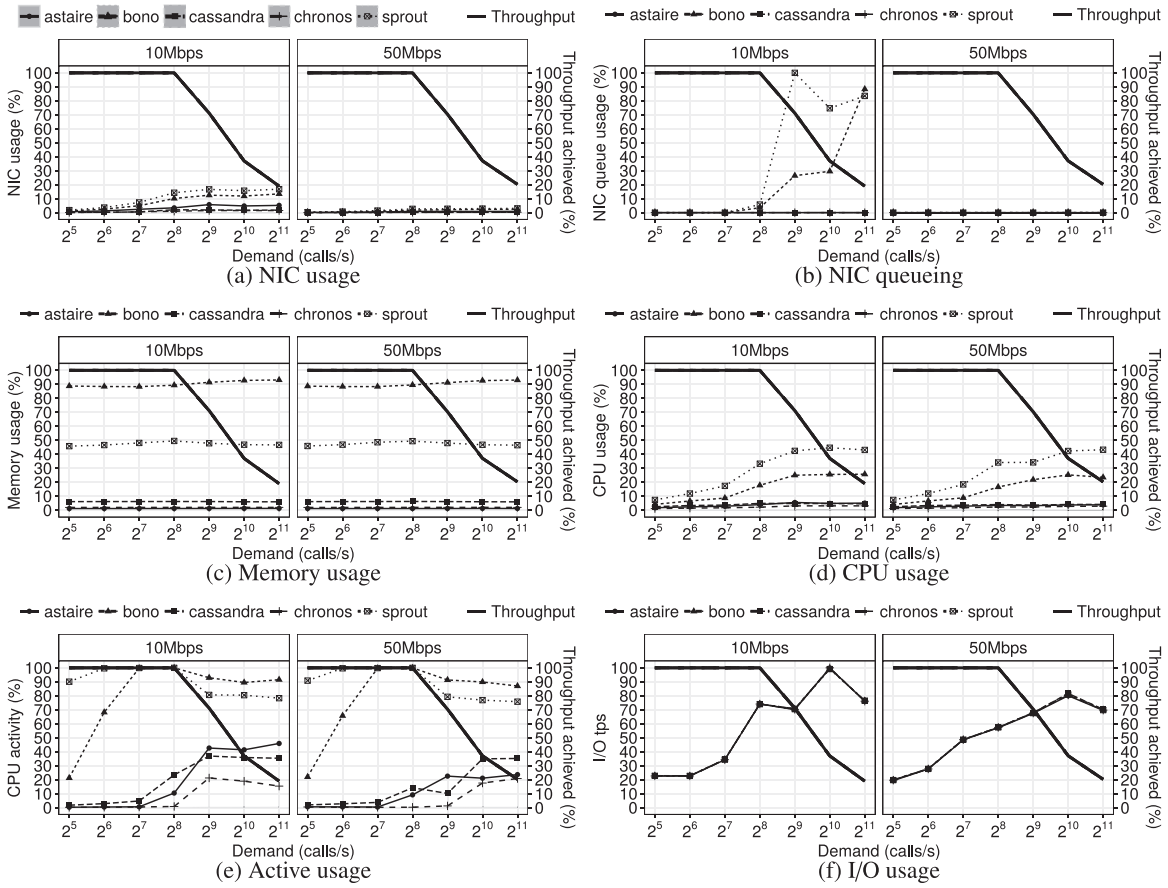


Fig. 4. Monitoring metrics in Clearwater IMS with capacity constrains.

and Sprout deals with the authentication aspects of connections. Hence, both VNFs have a much higher probability of becoming bottlenecks in the service chain.

Four metrics aid in diagnosing the drop in application throughput in this scenario: memory usage, CPU usage, CPU activity, and I/O usage. Fig. 4c shows that the Bono node consumes around 90% of available memory when subjected to a small demand (2^5), and that Sprout nodes use around 40% of available memory. The high memory usage is likely the result of two factors: (*i*)the Bono node holds information about each active TCP connection, and (*ii*)Clearwater uses Apache Cassandra as database, which consumes a significant amount of memory [32]. Considering that memory consumption grows with the increasing demand, and that the sum is greater then 100%, we can conclude that performance degradations occur due to memory swapping in the server, which increases the latency of requests and, in consequence, causes the node to reduce the token bucket size. In the case of CPU usage, from a demand of 2^8 calls/s and onwards, there is an increase of both Sprout and Bono nodes consumption. However, even when combining the consumption of both nodes ($0.45 + 0.25 = 0.7$), we are far away from the resource usage limit, which complicates the task of establishing monitoring thresholds. CPU activity is also meaningful, as the value increases proportionally to demand. Finally, I/O usage directly relates to the degradation of the application throughput, as the throughput falls drastically when I/O devices are used more. Still, it is worth mentioning that measurements of I/O transactions per second were taken in the host, and thus, it is difficult to identify which VNF is the cause of performance degradation. Nonetheless, one can argue for NIC queueing observation, since it indicates network capacity is restricted. However, network capacity is clearly not the root cause of the throughput drop because bandwidth improvement does not solve the problem.

To understand the effects of scaling up resources, we run the Clearwater scenario in a hardware with more resource capacity. Fig. 5 depicts the measurement results for this experiment. As the plots show, the amount of sustained demand increases with scaling up action; throughput application only starts to fall with a demand of 2^{10} calls/s. Other two aspects must be highlighted: first, the NIC capacity influences the general performance; second, the sum of memory usage does not exceeds available capacity. This is explained by the eight times larger memory capacity that the second hardware has in relation to the first one. By taking this into account, we can update the subset of representative metrics to include NIC queuing.

The difficulty in establishing thresholds is confirmed in the Clearwater IMS scenario. Even thought we can define a set of representative metrics, values of metrics depend on available hardware capacity. For example, while the CPU usage indicates saturation of VNFs with around 40% in the PC workstation (with 2^8 calls/s in Fig. 4), in the COTS server (Fig. 5), this values is far away from the amount where performance degradation starts (80% when the demand is 2^{10}). Such discrepancy suggests that the analysis of metrics must be done case by case, and they must be combined for better results.

3.4. Stress injections in security chain

In addition to using capacity restrictions to generate bottlenecks, we evaluate how metrics behave when VNFs are subjected to injections of stress in multiple resources. By doing this, we are able to reproduce the experiments from [12]. In such scenario, the

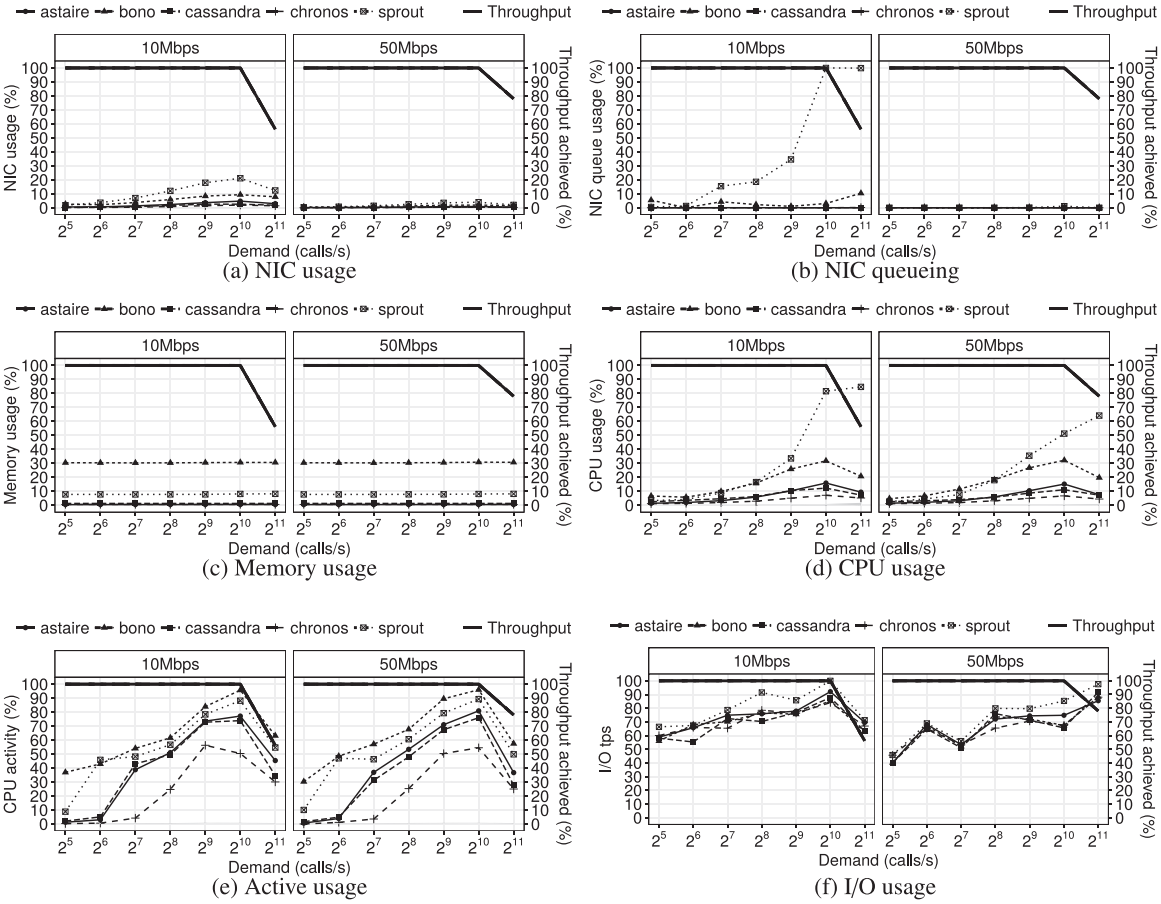


Fig. 5. Monitoring metrics in Clearwater IMS after vertical scaling.

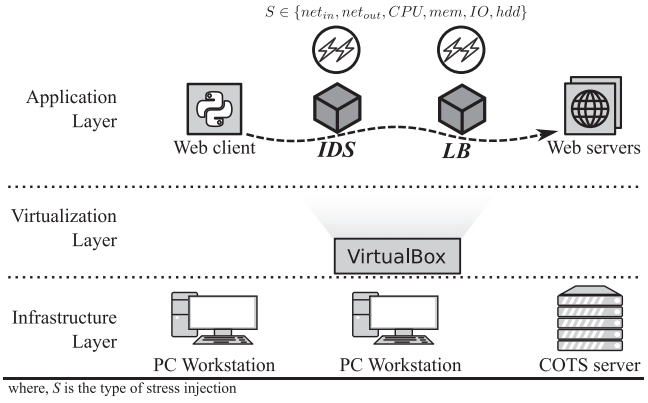


Fig. 6. Scenario 3 – Security-based VNFs with stress injections.

NFV service chain consists of an IDS (implemented through Suricata[10]) and a proxy Load Balancer (LB) (implemented through Balance[11]), sequentially ordered (Fig. 6). During a 260 s interval, a Web client submits a constant demand of 30 reqs/s for a 1 MB file hosted in two replicated Web servers. Also, the workload combines periods of regular traffic, where VNFs only process the Web demand, and periods of resource contention. For the latter, synthetic programs saturate the VNFs with: inbound flood traffic, outbound flood traffic, CPU computation, memory allocation/writing, I/O bus writing, and HDD writing. The first stress load is inflicted to the IDS, starting at the 10 s mark, lasting for 10 s, and repeating every 20 s stressing another resource. From the 130 s and onwards the LB is subjected to the same stress loads, in the same time frames. Regarding system configuration, we use the same standard PC workstation and VirtualBox images as in the first scenario.

Fig. 7 depicts the behavior of the metrics analyzed in the experiment with stress injections. The gray vertical bars depict the time intervals in which resources are stressed (*i.e.*, rx bw, tx bw, CPU, I/O bus, mem, and HDD), where dotted bars refers to stresses in the IDS and solid bars to stresses in the LB. While the continuous lines show the application throughput, the lines with points show metrics behavior in both the VNFs. The intention of injecting stresses is to impact on application throughput independently of the VNF being stressed. However, results show that the IDS becomes a bottleneck only when it is subjected to an inbound flood traffic (rx bw), causing degradation on application throughput. In the opposite, the injection of a network output flood (tx bw) in the LB is the only type of stress that does not cause a performance degradation to the Web service. At a first glance, NIC usage may be thought as a signal of performance problems: the lower is NIC usage, the higher is degradations in throughput. This claim is valid if we consider the service as a whole, but, if the aim is to point out bottlenecks, the metric fails because it produces misleading results. By looking more closely at the results, what we see is that NIC usage points out the wrong VNF: during the interval 10–20 the LB is the VNF with the lower NIC usage, indeed, the stressed VNF is the IDS. Such behavior repeats for all the other intervals where application throughput degrades.

[10] https://suricata-ids.org/.
[11] https://www.inlab.de/balance.html.

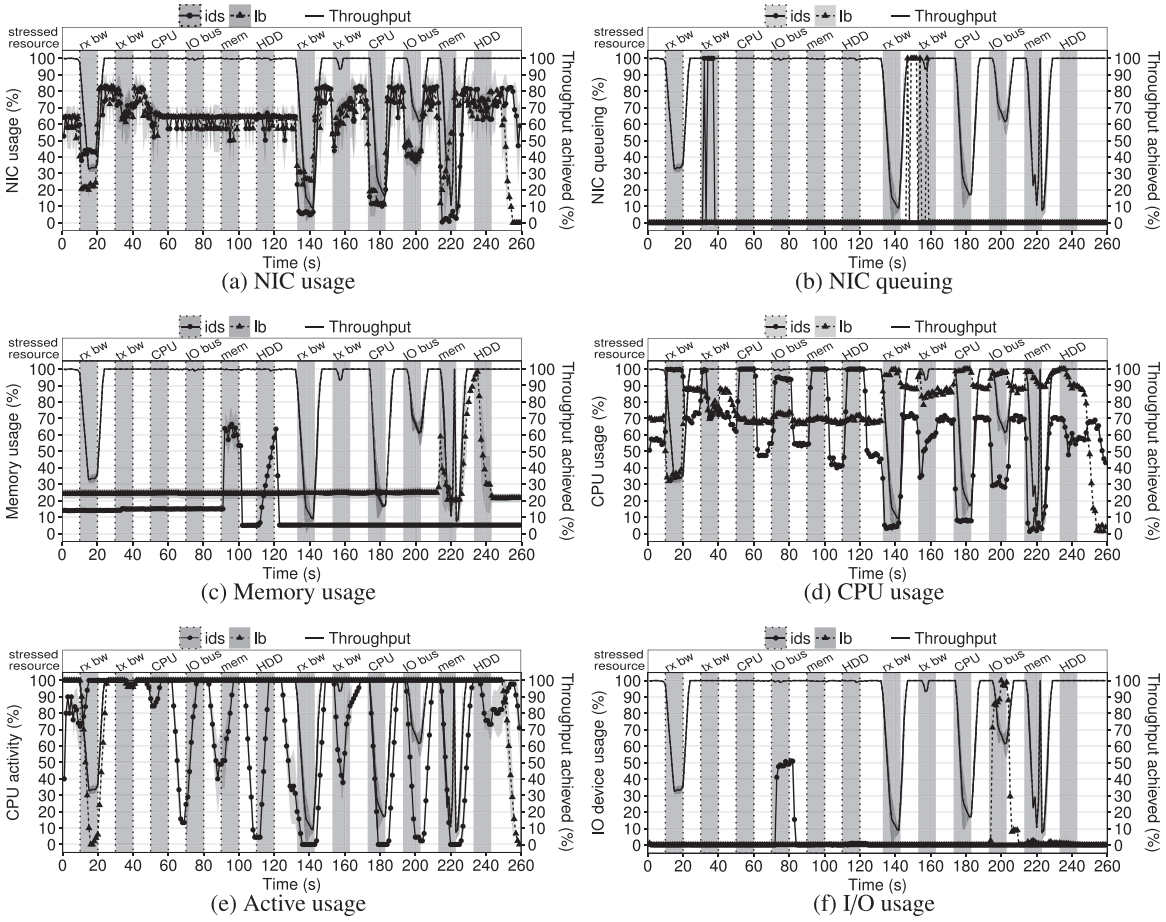


Fig. 7. Monitoring metrics in Security with stress injections. Gray bars represent the slices of time in which a resource has been stressed, where dotted and continuous borders represent the IDS and LB, respectively.

Since the observation of NIC usage may imply in misleading results, we must now evaluate how the remaining metrics behave against the injections. NIC queuing occurred in both VNFs when we submit an output flood traffic load. The TCP congestion algorithm explains this behavior; when the flood starts, TCP tries to increase the congestion window size, but, as the available bandwidth has not changed, packets remain in backlog until the congestion window reduces to the previous value, which is proportional to the bandwidth. Regarding degradations, Fig. 7b shows that the output traffic flow does not influence any of the VNFs, being thus NIC queuing irrelevant for bottleneck diagnosis in the stress injection scenario. Similarly to NIC queuing, memory usage is useless for diagnosing bottlenecks, stress loads in memory, and HDD makes an increase of memory usage, which does not alter the application throughput. In turn, CPU usage is most more promising for diagnosing degradations; for all the cases where application throughput struggled, the CPU exhibits a high load in the stressed VNF. However, the metric also presents a lot of false positives: there are cases where CPU is high and there are no performance reduction. An insight appears from the CPU activity: when the LB is subjected to any kind of load, the activity of the IDS reduces. Unfortunately, this insight is not effective for diagnosing purposes. Finally, I/O usage is valuable for one of the five performance issues: when injection occurs in the I/O bus of IDS, I/O usage increases and accurately signalizes the performance issue.

3.5. Section insight

This section evaluated a set of metrics to uncover which subset of metrics provide more insights into NFV performance degradations. By analyzing all the provided results, one can conclude that the subset of the most representative metrics used to diagnose NFV performance, in distinct scenarios, must include CPU usage, memory usage, NIC usage, NIC queuing, and I/O usage. However, two main reasons still hamper such diagnosing process: the influence of each metric varies severally across the multiple scenarios, and it is difficult to establish a threshold for each of them. In the next section we discuss how to properly combine these metrics to achieve a model that quantifies performance degradations.

4. Modeling the `guiltiness` of VNFs

In the previous section analysis, we found a set M of performance representative metrics composed of CPU, memory, NIC, and I/O usages. Also, the assessments show that these metrics have different levels of importance on signaling performance degradation in the three evaluated scenarios. Thus, in this section, we investigate how to combine such metrics to diagnose performance issues in heterogeneous NFV scenarios effectively.

4.1. `Guiltiness` model

As in the Koizumi's study [21], one approach to solve this problem, and quantify the performance P_v of a VNF v in a service chain,

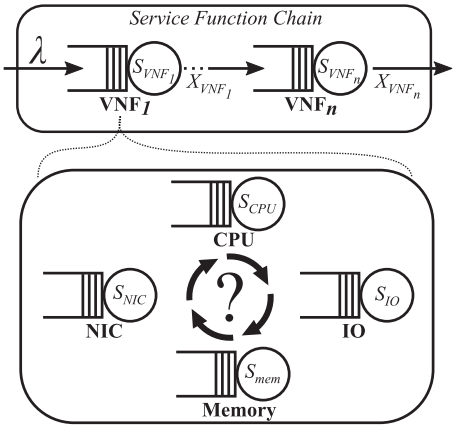


Fig. 8. Queueing networks model for Service Function Chain.

is to combine each metric m_i in M through a simple weighted sum:

$$P_v = \sum_{i}^{m} w_i \times m_i \tag{1}$$

where, weights $w_1 \ldots w_m$ define the importance of each metric in the evaluated scenarios, and m is the number of metrics in M. Although this approach can provide an estimate of performance, it relies on linear regressions based on historical measurements, which can make it unfeasible for real-time bottleneck detection because it consumes an amount of time until having enough information for regressions. As a first step towards circumventing this limitation, we study the queueing networks theory to model the `guiltiness` of performance degradation caused by each VNF on a service chain. Fig. 8 depicts an abstract view of this model. Let us consider each VNF as a queue, which receives requests with an arrival rate λ, processes and forwards them to the next queue (*i.e.*, VNF) in the chain with an outgoing rate X_v (also known as throughput). In this model, the amount of time a request consumes to pass through a queue v is denoted as residence time R_v, and the total time taken by a request to traverse the entire service chain denotes the chain's response time Rt_c, which is the sum of all residence times. Also, as we uncovered in Section 3, four resources can relate to the internal processing of a given VNF: CPU, NIC, memory, and I/O devices.

One of the most well-known tools for modeling residence times in queueing networks is the Utilization Law [15,25]. The Utilization Law defines the residence time R of a request in a queue as the relation between service time S and queue usage ρ, with ρ impacting R in an exponential manner. This means that at low values, ρ has a small impact on the performance, but as ρ increases the performance falls quickly. This happens because, at high utilization, the resource cannot cope with the processing demand, and this leads to the queuing of processing tasks. Considering that four queues represent the internal components of a VNF, and that request residence times in such resources depend on their utilization, one can model the residence time of a request in a VNF v as in Eq. (2).

$$R_v = \frac{S_{cpu}}{1-\rho_{cpu}} + \frac{S_{mem}}{1-\rho_{mem}} + \frac{S_{IO}}{1-\rho_{IO}} + \frac{S_{NIC}}{1-\rho_{NIC}} \tag{2}$$

To be precise, the model depends on measurements of each queue usage and also on the correct estimation of service time. Regarding the former, monitoring tools can provide online values about queues. As for the latter, to find out correct values of S requires a deep knowledge of how each queue perform the processing of requests. Another option is to rely on historical observations of arrival rate and throughput to estimate service time. However, this dependency on historical data leads us to the same problem we had by using the simple weighted sum, *i.e.*, it hinders the detection of performance issues in real-time. The heterogeneity of VNFs also increases the complexity of performance modeling. In queueing network models, to have accurate estimates, analysts must understand the relations between the internal components of each modeled node; requiring knowledge about the order in which components receive requests and how long each component takes to process them. NFV scenarios can include VNFs of distinct types, commonly provided by different vendors, and thus, to accurately model the performance of VNFs, an in-depth understanding of their behavior is required.

We deal with the aforementioned problems (*i.e.*, uncertainty about relations between queues, and knowledge of queue internals) using two different strategies. First, as in the weighted sum models, we add a weight for each resource as an approximation of service time. Second, we include the resource activity (A) to gauge the importance of each resource. We rely on the study of Wang et al. [29], which discusses the active duration time as a metric to detect bottlenecks. The authors show that this metric faithfully characterizes bottlenecks in production networks. The original meaning of active duration time corresponds to the period of time that a node is in a non-idle state. However, as each resource can show fluctuations of low consumption, we use a threshold to establish when the resource is idle. A is calculated in the following way: a monitor collects P samples of ρ and for each value greater than a threshold T it increments a counter C. Afterward, A is computed as the ratio between C and P. The threshold T is set by network operators according to their perspective of relevant load. As a start point, T can be the measured value of ρ when there are no requests being sent to the VNF.

The following example helps understand how A is computed. Consider two time-series with measurements of CPU usage from distinct VNFs: $\rho_{cpu}^{vnf_1}$:[0.3, 0.4, 0.2, 0.3] and $\rho_{cpu}^{vnf_2}$:[0.1, 0.5, 0.1, 0.5]. Both VNFs have an average CPU usage of 0.3, which means that they are not overloaded. However, while A_{vnf_1} is equal to 1.0 ($C_{vnf_1} = 4, P = 4 \rightarrow A_{vnf_1} = 4/4 = 1.0$), A_{vnf_2} is 0.5 ($C_{vnf_2} = 2, P = 4 \rightarrow A_{vnf_2} = 2/4 = 0.5$), making it clear that, to properly identify utilization patterns of each VNF, we must be able take into account the active time of each resource.

Theorem 4.1. *Let $v = \{cpu, mem, NIC, I/O\}$ be a VNF with four major resource components: CPU, memory, I/O, and a network interface. The normalized residence time of packets in v, R'_v, is given by the weighted sum of normalized residence times of each of its resources.*

The proof of Theorem 4.1 is in Appendix A. Theorem 4.1 provides the basis for modeling the `guiltiness` metric, and its intuition is as follows. As we adapted the Utilization Law to our purposes, we define `guiltiness` of a VNF v, G_v, as an approximation of the normalized value of residence time R_v. This means that the maximum value of G_v will be 1.0, which corresponds to the maximum observation of residence time possible[12]. By using weights and resource activity, we can now define the `guiltiness` G_v, for any VNF v, on the performance degradation of a service chain as in Eq. (3):

$$G_v = \sum_{m \in M} w_m . A_m . \frac{1}{1-\rho_m}, M = \{CPU, mem, NIC, I/O\} \tag{3}$$

Even though Eq. (3) takes into consideration all the general resources (*e.g.*, CPU, memory, NIC and I/O usages) for distinct VNFs, a more precise signal of performance degradations can be achieved by including the measurement of queueing in the NIC interface. In

[12] In fact, the residence time of a queue can grow to the infinite, we assume this only occurs in critical situations, and don't consider this in the model.

Section 3, the results from the assessment of metrics' representativeness show that NIC queueing is a valuable symptom of performance degradation, since the occurrence of network queuing indicates that a VNF is not being able to cope with the incoming demand. In our previous work [22], we already recognized the need to contemplate NIC queuing on performance degradations. This is also supported by the findings of Wu et al. [12], who argue for the instrumentation of every buffer/queue of VNFs' internal components. Thus, in our first proposal [22] we settled the following relation between CPU activity and NIC queuing:

$$G_v = \ldots + w_2 \times A_{CPU} + \ldots - w_4 \times \frac{A_{CPU}}{1 + NICqueue} \tag{4}$$

where, w_2 and w_4 define the relevance of these terms. The idea is that if queueing does not occurs in NIC, the amount that A_{CPU} will impact on `guiltiness` reduces relatively to the difference between w_2 and w_4. As this previous model does not take into account I/O, memory and NIC resources, it showed not to be accurate enough for general-purpose scenarios. However, we take advantage of this network queuing logic, unifying term w_2 and w_4 into a single term to be added to Eq. (3). Considering the assessment results (Section 3) showed CPU activity tend to exhibit high values when the VNF is submitted to any load level, our intent is to increase CPU activity impact on `guiltiness` proportionally to NIC queueing, as stated in Theorem 4.2.

Theorem 4.2. *Let R_v be the normalized residence time of packets in VNF $v = \{cpu, mem, NIC, I/O\}$, given by the weighted sum of normalized residence times of each of its resources. The fifth term of Eq. (5) increases and reduces the `guiltiness` of a VNF v, G_v, according to the CPU activity related to the processing of queued packets in NIC.*

The proof of Theorem 4.2 is in Appendix A. The fifth term of G_v corresponds to the active time taken by the CPU to process NIC queued packets. Hence, the fifth term added to the model (Eq. (5)), in this paper, emerges naturally.

$$G_v = \sum_{m \in M} w_m . A_m . \frac{1}{1 - \rho_m} + w_5 . \frac{(A_{CPU} \times \rho_{NIC_{queue}} - A_{CPU})}{1 + \rho_{NIC_{queue}}},$$
$$M = \{CPU, mem, NIC, I/O\} \tag{5}$$

4.2. Adjusting `guiltiness` weights

Weights in Eq. (5) have two purposes: normalize values and gauge metrics. As the aim of `guiltiness` is to produce a value that denotes how much a VNF is responsible for degrading the performance of a service chain, the domain of G_v is $\{G_v \mid 0 \leq G_v \leq 1\}$, where $G_v = 1.0$ means the likelihood of v being the cause of degradations is 100%. As a consequence, we must establish weights that normalize the sum of terms to be at max 1.0. For the sake of explanation, let us neglect the fifth term of Eq. (5) and assume the other four are equivalent (with same weights). Considering each term produces a value that comes from the ratio between the resource activity (A) and its usage (ρ), and that both A and ρ are subjected to the same range of values of G_v (*i.e.*, from 0.0 to 1.0), the default values of weights w_1 to w_4 are equal to 0.0025. With these values, when a given resource is fully used (*i.e.*, ρ and A equal to 100%) the relative term will be in charge of 0.25 of `guiltiness`. Regarding the fifth term, the value of w_5 express the percentage that NIC queuing impacts on `guiltiness`, using $w_5 = 0.1$ implies that NIC queuing accounts for 10% of performance degradations. Thus, with this value, in a case without NIC queuing the maximum value of G_v would be 0.9.

In addition to normalization, the importance of each resource may vary because of individual aspects of service chains and network functions. For example, if the CPU has a higher impact on service chain performance than the memory, w_{CPU} must be greater than w_{mem}. The results presented in Section 3 also support this claim: while NIC usage, NIC queuing, and CPU usage are better indicators in the security scenario with capacity restrictions, memory usage and I/O usage to diagnose performance issues in the Clearwater IMS. Since each metric has a different impact in specific scenarios, we must find a method to properly adjust the equation weights.

Let us again assume that a VNF service chain is a queueing network model. As such, we can adapt queueing network laws to our purpose. Given that the residence time is the time that a VNF takes to process a request, then the sum of the residence time of each node in the chain is the total chain residence time, according to the *General Response Time Law*. Therefore, if we take into account that G_v is a representation of residence time in VNF v, the sum G_c of all G_v values in a service chain c is the total residence time. To a better understand of the derivation, we first split the five terms of `guiltiness` as:

$$\begin{aligned}
t_1 &= w_1 . A_{CPU} . \frac{1}{1 - \rho_{CPU}} \\
t_2 &= w_2 . A_{mem} . \frac{1}{1 - \rho_{mem}} \\
t_3 &= w_3 . A_{NIC} . \frac{1}{1 - \rho_{NIC}} \\
t_4 &= w_4 . A_{IO} . \frac{1}{1 - \rho_{IO}} \\
t_5 &= w_5 . \frac{(A_{CPU} \times \rho_{NIC_{queue}} - A_{CPU})}{1 + \rho_{NIC_{queue}}}
\end{aligned} \tag{6}$$

Then, in Eq. (7), we derive G_c to extract weights. Note that w_1, w_2, w_3, w_4, and w_5 are constant for all v in c. Bear in mind this assumption implies in a generalization of resources' importance for all VNFs in a given service chain.

$$\begin{aligned}
G_c &= \sum_{v \in c}^{n} G_v \\
&= \sum_{v \in c}^{n} (w_1 t_1 + w_2 t_2 + w_3 t_3 + w_4 t_4 + w_5 t_5) \\
&= \sum_{v \in c}^{n} w_1 t_1 + \sum_{v \in c}^{n} w_2 t_2 + \sum_{v \in c}^{n} w_3 t_3 + \sum_{v \in c}^{n} w_4 t_4 + \sum_{v \in c}^{n} w_5 t_5 \\
&= w_1 \sum_{v \in c}^{n} t_1 + w_2 \sum_{v \in c}^{n} t_2 + w_3 \sum_{v \in c}^{n} t_3 + w_4 \sum_{v \in c}^{n} t_4 + w_5 \sum_{v \in c}^{n} t_5
\end{aligned} \tag{7}$$

where n is the number of VNFs in the chain c.

The equation has the following outcome: to fit G_c as a representation of the chain response time Rt_c, we must find a quintuple $< w_1, w_2, w_3, w_4, w_5 >$ that makes the relation exact to the metric observations. To achieve that, the first step is to aggregate the measurements of both resource usages and activities and then compute the sum of the terms from all VNFs. Next, each of the five terms in Eq. (7), as well as, the respective $< Rt_c >$ are stored in a historical information database, with which, we can compute the weights that make $G_c \cong Rt_c$.

We rely on a hybrid learning procedure, combining both neural networks and nonlinear regression, to compute the weights. The reason why we combine these methods is to avoid fluctuations of regression results. Considering that not all scenarios are available *a priori*, and that non-linear regressions take into account the entire dataset of measurements, outliers can directly affect weight estimation. As we filter the training dataset, neural networks perform more abstract regressions, which make them less sensitive to outliers. To cope with the real time diagnosis requirement of the *guiltiness* metric, we train the hybrid learning procedure with *online learning* [33], using each collected measurement to reduce

training error and make more accurate predictions for future samples.

Training data consists of metrics (*i.e.*, $t_{1,\ldots,5}, Rt$) and weights (*i.e.*, $w_{1,\ldots,5}$). Training can be either manual (the operator inserts training data) or automatic (using measurements and regressions). Once there is enough information to perform the predictions, the hybrid learning procedure combines results from neural network and regressions to compute the weights. Algorithm 1 presents the learning procedure using automatic training.

Algorithm 1 Hybrid learning algorithm.

```
1:  nn ← NeuralNetwork()
2:  dataset_nn ← ∅
3:  dataset_nlr ← ∅
4:  w^default_1,...,5 ← [0.0025, 0.0025, 0.0025, 0.0025, 0.1]
5:  w^current_1,...,5 ← w^default_1,...,5
6:  procedure LearnWeights(t_1,...,5, Rt)
7:      Rt_Norm ← Rt / Rt_Max
8:      dataset_nlr ← dataset_nlr ∪ (t_1,...,5, Rt_Norm)
9:      w^nlr_1,...,5 ← NonlinearRegression(dataset_nlr)
10:     w^nn_1,...,5 ← nn.Predict(t_1,...,5, Rt_Norm)
11:     for x ← (current, default, nn, nlr) do
12:         G^x_c ← Guiltiness(w^x_1,...,5, t_1,...,5)
13:         r^2_x ← RSquared(G^x_c, Rt_Norm)
14:     end for
15:     if r^2_nlr > threshold then
16:         dataset_nn ← dataset_nn ∪ ([t_1,...,5, Rt_Norm], [w^nlr_1,...,5])
17:     end if
18:     if r^2_nn > threshold then
19:         dataset_nn ← dataset_nn ∪ ([t_1,...,5, Rt_Norm], [w^nn_1,...,5])
20:     end if
21:     nn.Train(dataset_nn)
22:     r^2_max ← Max(r^2_current, r^2_default, r^2_nn, r^2_nlr)
23:     w^current_1,...,5 ← Weights(r^2_max)
24:     UpdateMonitors(w^current_1,...,5)
25: end procedure
```

For each new set of metric measurements the algorithm decides between weights estimated through regressions and neural networks. For doing so, Algorithm 1 leverages two training datasets: $dataset_{nlr}$ that contains all measured data, and $dataset_{nn}$ that contains only the weights that achieve the desired accuracy. Algorithm 1 starts when the metric measurements of a service chain (*i.e.*, $t_{1,\ldots,5}, Rt_{Norm}$) are retrieved. Next, in lines 7 and 8, it computes the normalized response time Rt_{Norm} and save the values into the general training dataset ($dataset_{nlr}$). Then, if it already has enough information, the algorithm calls NonlinearRegression and Predict (lines 9 and 10) to compute the set of weights that fit the `guiltiness` function with the metric history, using the nonlinear regression and the neural network, respectively. After the weights are computed, the algorithm checks `guiltiness` accuracy. To do so, it computes, in lines 11 to 14, the coefficient of determination (R-squared) of the fit using `guiltiness` values from each weight set (current, default, neural network, and nonlinear regression). If the computed R-squared values are greater than a threshold (lines 15 to 20), the knowledge base of the neural network ($dataset_{nn}$) is updated to include a new match between the measured values and the computed weights ($[t_{1,\ldots,5}, RtNorm]$, $[w_{1,\ldots,5}]$). Next, the algorithm calls Train, so the neural network can learn with the filtered measurements. Finally, the algorithm calls UpdateMonitor to renew the weights used by each VNF monitoring agent to compute `guiltiness`.

Although we propose an adaptive algorithm to define the model weights, it depends on the capacity of operators to monitor end-metrics. We argue that the combination of default weights $w_1 = w_2 = w_3 = w_4 = 0.0025, w_5 = 0.1$ with the activity of resources can represent the majority of scenarios (see results from Section 5). In cases in which learning is involved, we recommend feeding the knowledge database with data from both under-stressed and overloaded chains. By doing so, the mechanism can learn weights based on the intermediate cases.

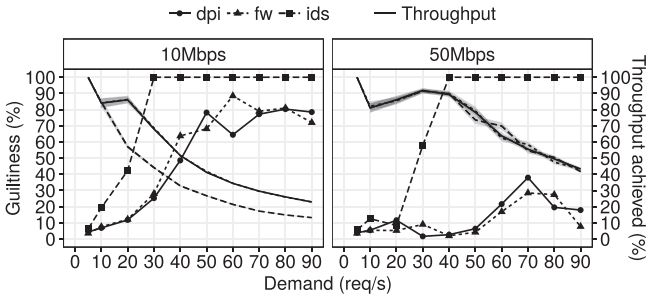


Fig. 9. `Guiltiness` measurements on security chain with capacity restrictions using default weights.

5. Evaluation

To assess the ability of the `guiltiness` metric in quantifying performance degradation, we performed experimental evaluations. First, we analyze how the model applies to detect performance degradation using default weights. Next, we show the benefits of adjusting weights for each scenario. For doing so, we performed non-linear regressions in each of the three evaluated scenarios and plotted adjusted `guiltiness` values.

5.1. `Guiltiness` results using default weights

As we previously discussed in Section 4, when the importance of the `guiltiness` metrics to performance degradation is the same, the default weights (w_1 to w_4) for terms t_1 to t_4 are equal to 0.0025. Also, using $w_5 = 0.1$ implies that NIC queuing accounts for 10% of degradations. We now assess how `guiltiness` explains the application throughput of the evaluated scenarios when these weights apply.

Fig. 9 depicts `guiltiness` results for the security chain with capacity restrictions scenario. In the both provisioned capacities `guiltiness` unequivocally identifies the performance degradations. With 10 Mbps network capacity, the metric increases to a higher level as soon as application throughput starts to fall, pointing out the IDS with 100% of `guiltiness` from 30 req/s onward. Also, the firewall and the DPI exhibit a much smaller `guiltiness` when compared to the IDS, which is expected given the higher throughput achieved when these VNFs are bottlenecks. The 50 Mbps case shows a clear distinction among VNFs: the IDS remains as the major responsible for performance degradations, being responsible for 100% of decreases; in turn, the firewall and the DPI respond for a max of 40% of `guiltiness`, suggesting these VNFs have a lower impact on chain performance. Although the approach of combining metrics we use in `guiltiness` allows identifying the performance issues faithfully, there is still room for improvement. By looking to the 50 Mbps case from a demand of 70 req/s and onwards, both the firewall and the DPI exhibit an abnormal `guiltiness` drop, emphasizing the need for weights adjustment.

Fig. 10 depicts `guiltiness` results for the Clearwater IMS with capacity restrictions. When the available bandwidth is equal to 10 Mbps, and with a demand of 2^8 requests per second, `guiltiness` of the Bono node increases to 45%, followed by the

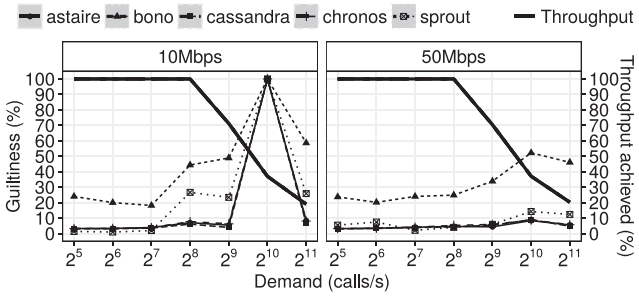


Fig. 10. `Guiltiness` measurements on Clearwater IMS with capacity restrictions using default weights.

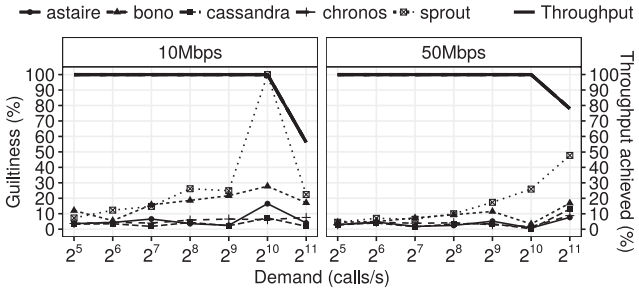


Fig. 11. `Guiltiness` measurements on Clearwater IMS with capacity restrictions and vertical scaling, using default weights.

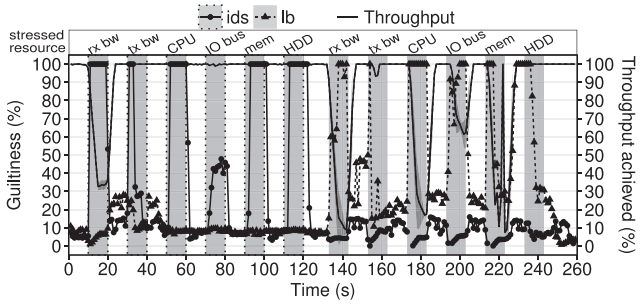


Fig. 12. `Guiltiness` measurements on security chain with stress injections, using default weights. Gray bars represent the slices of time in which a resource has been stressed, where dotted and continuous borders represent the IDS and LB, respectively.

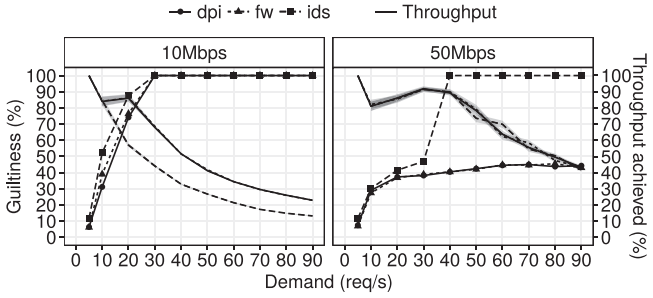


Fig. 13. `Guiltiness` measurements on security chain with capacity restrictions using learned weights.

Sprout node (30%). The value of G indicates that the system collapses with 2^{10} reqs/s, where all nodes have a `guiltiness` of 100%. Such behavior mainly occurs due to the high utilization of I/O and memory, and also due to the occurrence of NIC queuing. As we already discussed, the increase of network capacity to 50 Mbps does not affect the application throughput, and the high level of memory and I/O usages are the only metrics that were able to explain such a behavior. Regarding `guiltiness`, even by exhibiting a reduction in its general value, it indeed points out the Bono node as the root cause of problems, with a `guiltiness` of 25% at 2^8 reqs/sec, 35% at 2^9 reqs/sec, and 50% at 2^{10} reqs/sec. This behavior can be explained due to the nature of the `guiltiness` that combines multiple metrics.

When we vertically scaled resources for the Clearwater IMS scenario, we observed that the degradation of throughput was decelerated, occurring at a demand of 2^{10} requests per second. Fig. 11 shows `guiltiness` accurately identifies such performance issue, independently of the provisioned network bandwidth. The main difference from the previous scenario is that now the node being pointed as a bottleneck is the Sprout node, instead of Bono. The higher usages of CPU and I/O in Sprout help explain this behavior. It is worth highlighting that, similarly to results from PC workstation, `guiltiness` decreases severally at a demand of 2^{11} req/sec with 10 Mbps network bandwidth. We argue, however, that this does not hinder the metric usage for diagnosing the performance issues because it signalizes them right before their occurrence, in this case, with 2^{10} calls/s.

Concerning the scenario with stress injections, Fig. 12 shows that `guiltiness` can point out the correct VNF, *i.e.*, the one that is causing application throughput degradation. During the interval 0–120, the `guiltiness` value of the IDS increases in distinct moments, following the stresses it is subjected to. Similarly, from instant 140 and onwards, the VNF pointed as the cause of problems is the LB. We argue that not providing misleading results (indicating the wrong VNF) is the first expected result of a diagnostic metric. As such, these results show that based on `guiltiness` network operators can be sure when distinguishing the impact of each VNF. However, given the unexpected behavior of the IDS, in which only the RX bandwidth flood impacts on application throughput, as occurred with CPU usage (see Fig. 7d), the `guiltiness` indicates false positives; the metric wrongly points out performance issues in the intervals 30–33, 50–60, 90–100, and 100–120. On the other hand, when stresses occur in the LB, application throughput struggles, which is correctly indicated by the `guiltiness` value.

5.2. `Guiltiness` results using learned weights

In this section, we discuss how the learning procedure applies to weights' adjustment in each of the evaluated scenarios. As we already measured resource usages and response times from the previous scenarios, we rely on a *post-mortem* weight prediction. However, it must be clear that we simulate an iterative learning process, feeding each data entry sequentially to our prediction procedure. As we rely on *online learning*, there is no need for a train-test split as it is usually done with *batch learning* [33].

When we feed the learning procedure with data from the security chain with capacity restrictions, the procedure returns the following weight set: (0.0006,0.057,0.0005,0.0008,0.075). This set of weights means that t_5 and t_2 are the most important terms of Eq. (5) for determining the `guiltiness` of VNFs, and that t_1, t_3, and t_4 are almost irrelevant. The results depicted in Fig. 13 show a clear improvement when compared to the results obtained with default weights (Fig. 9). Using learned weights, there is a clear identification of bottlenecks: with 10 Mbps network capacity, all the VNFs exhibit high `guiltiness` values (*i.e.*, more than 80%) from 30 req/s and onward, which faithfully represents the observed application performance. Also, with 50 Mbps the learned weights reduce the abnormal `guiltiness` behavior observed in the firewall and DPI. This is explained by the lower importance given to NIC usage and CPU usage, that suffer from small oscillations with higher demands (see results of Fig. 2), as all metrics

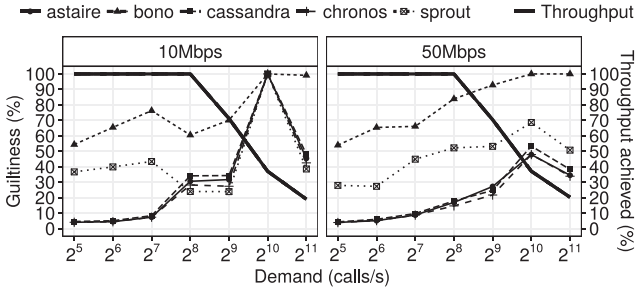


Fig. 14. `Guiltiness` measurements on Clearwater IMS with capacity restrictions using learned weights.

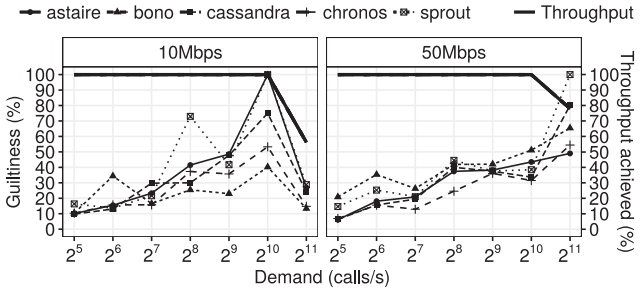


Fig. 15. `Guiltiness` measurements on Clearwater IMS with capacity restrictions and vertical scaling, using learned weights.

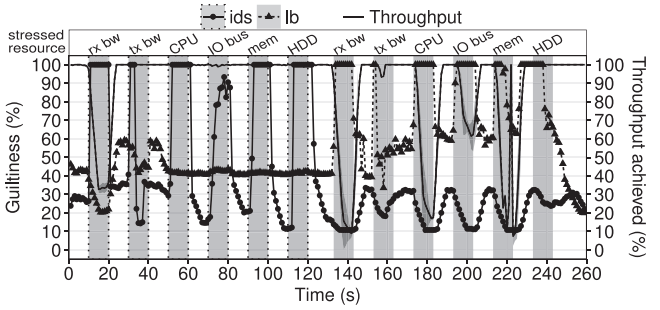


Fig. 16. `Guiltiness` measurements on security chain with stress injections, using learned weights. Gray bars represent the slices of time in which a resource has been stressed, where dotted and continuous borders represent the IDS and LB, respectively.

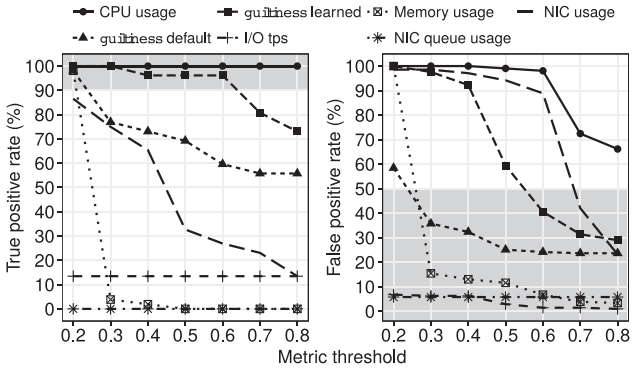


Fig. 17. True and false positive rates of metrics with distinct thresholds. In the left side, the gray area highlights the points where metrics achieve more than 90% of TPR. In the right side, the gray area highlights points where metrics achieve less then 50% of FPR.

have exponential relation to `guiltiness` when ρ has high values even small decreases imply in glaring impacts.

As we go a step further, we found the best results of the learning procedure, which came from the Clearwater IMS scenario (Figs. 14 and 15). With this dataset the learning mechanism produced the following weight set: $(0.046, 0.133, -0.0001, -0.208, 0.050)$. The outcome is that t_3 (NIC term) is almost irrelevant, t_4 is inversely proportional to `guiltiness` (the higher the I/O usage the lower is the guilty), and t_2 (memory term) is the most relevant for the `guiltiness` of VNFs. In the case without vertical scaling, `guiltiness` with the learned weights (Fig. 14) can now provide a better characterization of the performance downfall of the 50 Mbps case, without losing expressiveness for the 10 Mbps case.

The benefits of learning weights also appear in the results with vertical scaling (Fig. 15). As we previously discussed, when the learning mechanism is not used (default weights, Fig. 11) the `guiltiness` of the Sprout node only differs from the other VNFs with 2^{10} reqs/s. Besides being a reliable signal of performance degradation, using the learned weights makes the `guiltiness` values be more influenced by the overall increase of metrics measurements. As a consequence, the indication of performance issues starts earlier (with 2^7 reqs/s), which can aid network operators to take preventive actions. The counterpart is that using learned weights in the vertical scaling Clearwater IMS scenario makes it harder to distinguish the impacts of individual VNFs.

Finally, we analyze the worst-case scenario for learned weights, the case of security chain with stress injection. As we already discussed, the atypical behavior of the IDS VNF makes the performance diagnosis process more challenging. This can also be justified because in Eq. (7) we generalize weights computation by taking into account metrics measurements from the two VNFs simultaneously. Regarding the learned weights, the adaptive algorithm results in the following set: $(-0.034, -0.050, -0.0001, 0.097, -0.153)$. This values point out t_4 (I/O term) and t_5 (CPU activity *vs* NIC queuing relation) as the most relevant to `guiltiness`, also, makes t_1, t_2, and t_3 be inversely proportional to `guiltiness` value. The consequence, regarding `guiltiness` behavior, is that with learned weights (Fig. 16) the values are slightly higher compared to when the model uses default weights (Fig. 12). However, even if it does not clear the total of false positives, the learned weights do not affect the ability of `guiltiness` in (*i*)pointing out the VNF being the cause of performance degradation, and (*ii*)indicating situations of application throughput drop.

5.3. Bottleneck detection precision

The analysis presented so far showed that `guiltiness` can point out bottleneck VNFs and indicate when service performance starts to collapse. We now analyze `guiltiness` regarding its precision in diagnosing performance issues. Considering that in this last scenario we leveraged controlled stress injections, we can account for the number of true positive and false positives performance detection cases. For each instant of time, we assume a true positive occurs when the application throughput falls bellow 80%, and the diagnostic metric is superior to a threshold. In turn, a false positive occurs when the diagnostic metric points out a performance issue (*i.e.*, its measurement is superior to the threshold), and the application throughput remains stable (*i.e.*, throughput is superior to 80%). As metrics have different threshold values, we must find out the best case of each one for a fair comparison of performance diagnosis precision. In Fig. 17 we plot the True Positive Rate (TPR) and False Positive Rate (FPR) for each metric using distinct values of threshold to diagnose performance issues.

Table 2
Summary of true positive rate (**TPR**) and false positive rate (**FPR**) for each metric in the security chain with stress injections scenario.

Metric	thold	TPR	FPR
I/O tps	0.8	13.46%	0.97%
NIC usage, as in [11,20,21]	0.2	86.54%	98.55%
NIC queueing, as in [12]	*any*	0.00%	5.80%
Memory usage, as in [11,21]	0.2	98.08%	100.0%
CPU usage, as in [11,17,19–21]	0.8	100.0%	66.18%
`guiltiness` with default weights	0.2	98.08%	58.45%
`guiltiness` with learned weights	0.6	96.15%	40.58%

We argue that a good pair (metric, threshold) must prioritize the identification of performance issues over reducing the occurrence of false positives. Thus, in the precision results depicted in Fig. 17, we highlight the cases with more then 90% of true positive rate and less than 50% of false positive rate. When the threshold is set to 0.2, memory usage, CPU usage, `guiltiness` with default weights, and `guiltiness` with learned weights provide more than 90% of TPR. In the interval between 0.3 and 0.6, the set of metrics with more than 90% TPR reduces to CPU usage and `guiltiness` with learned weights. From 0.7 and onward, CPU usage is the unique metric able to detect more then 90% of the performance issues. Regarding the false positive rates, considering only the metrics that have high TPR, with a 0.2 threshold, the `guiltiness` with default weights is the unique that produces a FPR close to 50% (*i.e.*, 58.45%); then, only with 0.6 the `guiltiness` with learned weights can result in less then 50% FPR at same time that produces high TPR. In turn, the best result for CPU usage appears at the 0.8 threshold, with a rate of 100% for true positives and 66% for false positives. To offer a clear view of `guiltiness`' benefits we summarize in Table 2 the absolute values of TPR and FPR results for each metric using the best cost-benefit threshold (*i.e.*, the ones with the higher TPR and the lower FPR).

The numeric results presented in Table 2 confirm that `guiltiness` is able to detect performance issues, positively identifying 96.15% of performance degradations when learned weights are used. Also, in the case using default weights, `guiltiness` detects 98.08% of performance issues. However, the major benefit appears concerning false positives: while a diagnosis based on CPU usage can detect 100% of performance issues, it presents 66.18% of false positives. In turn, while `guiltiness` with default weights presents 58.45% false positives, the learned weights reduces this amount to 40.58%, a 25.6% reduction when compared to CPU usage, and 17.87% when compared to default weights.

The results from the stress injection scenario evidence an open challenge for weights adjustment. Even though there are significant improvements regarding diagnosis precision with `guiltiness` – *i.e.*, the metric differentiates VNFs regarding their impact on performance – the anomaly behavior of the IDS VNF still produces a number of false positives. This indicates that the learning process must be refined to establish weights for each VNF, instead of generalizing values for the entire SFC.

6. Discussion

Application scenarios. As the `guiltiness` metric relies on traditional metrics from network performance analysis, one could argue that the metric could also be applied to diagnose the performance of a traditional computer network. Indeed, the `guiltiness` metric can be applied to diagnose the performance of general computer networks scenarios. However, in those scenarios, it is traditional to have the support of vendors, and the behavior of network devices is usually well-known by administrators. We were motivated by the heterogeneity and dynamism of NFV environments, with VNFs developed by some sort of third-parties, which requires a model able to, at the same time, abstract particularities and be precise on diagnosing performance issues.

Inspection depth. Another aspect susceptible to discussion is our approach of abstracting the VNFs' internals instead of understanding their in-depth functioning. Generally, when an operator aims to diagnose the performance of a target system, he or she relies either on an in-depth understanding approach, or an abstraction approach. The choice between each method depends on many factors, including: familiarity with performance modeling, knowledge about system internals, variety of components in the analysis, access level to performance metrics, expected precision of results, and urgency of diagnostic results. The related work, discussed in Section 2, also rely on either of these approaches: while [11,12,19,20,23] performed an in-depth investigation of their diagnosed systems, [18,21,24] prefer the abstraction approach. In this paper we followed a balanced approach: we first understand how general metrics apply to multiple scenarios, then model the `guiltiness` to abstract the specificity of each one. We understand that if an operator has in-depth access of VNFs internals, such an approach can result in more accurate diagnosis. However, as NFV environments are characterized by having a variety of VNFs, with multiple vendors involved, we argue that the abstraction provided by `guiltiness` is the best approach to diagnose the performance of general-purpose scenarios.

Timestamp analysis. In a similar direction, one could ask why not just measure the timestamps of requests in each VNF, and then point out the bottleneck as the one with the bigger interval between ingress and egress. In fact, that is probably the most precise way to diagnose bottlenecks. However, this requires low-level instrumentation to classify the multiple flows and then account for the timestamps, which is prohibitive in scenarios where operators must deal with a myriad of VNF types. Also, in the case of VNF-as-a-Service (VNFaaS), where network operators acquire network functions from a NFV provider [34], operators may not have full access to the VNFs' internals, hindering the analysis of logs to capture the timestamps.

Resource dependency. Although the `guiltiness` metric seems to exhibit an independent relation among metrics, in fact, the behavior of each metric is somehow dependent on others, particularly in the case of CPU. Since the CPU centralizes the data communication among resources, the processing on these resources implies in CPU consumption. In [20], the authors acknowledged such dependency to model the performance of virtual routers/switches. However, as the model takes into account the monitoring of resource usage metrics, we argue that the dependency effects are already contemplated in Eq. (5).

Model linearity. Finally, one can argue that, given the dependency among resources, other linear and non-linear combinations of metrics may result in more precise estimates of VNFs' performance. Although a non-linear combination can improve the diagnostic precision, the relations between metrics may vary from one scenario to another. This variation in available combinations makes diagnosing general-purpose scenarios prohibitive, as operators would have to understand the internal relations among VNFs' resources to model their dependencies accordingly.

7. Conclusions

The chaining of virtualized network functions is one of the driving forces of NFV. As VNFs run decoupled from the underlying hardware, NFV SPs take advantage of elasticity and dynamicity to offer customized network services to meet the requirements of their clients. Such network services typically rely on heterogeneous VNFs with distinct objectives (*e.g.*, security, performance, and routing). In addition, the performance of VNFs on forwarding network

packets directly impacts on end-services that depend on the network. Hence, network operators must be able to quantify individual impacts to diagnose performance issues.

In this paper, we reviewed the state-of-the-art approaches for performance diagnosis in NFV: monitoring multiple metrics and comparing to thresholds, and estimating the service chain performance through analytic models. Through an extensive set of experiments, we replicated NFV scenarios in the literature and analyzed the relation between monitored metrics and end-service performance. Based on these results, we concluded that *it is not possible to diagnose every performance problem using metrics from individual resources*; indeed, a model that combines metrics from CPU, memory, I/O, and NIC resources is better suited for the job. Based on that finding, we then resorted to queueing network theory to develop a novel mathematical model combining these metrics. We called this model `guiltiness`, which represents the amount of performance degradation each VNF is responsible for. The model consists of a weighted sum that includes resource usages, their activity, and a novel relation including NIC queuing. Results showed that `guiltiness` accurately diagnoses 98% of performance issues in the evaluated scenarios.

With the aim of reproducing the experiments of Wu et al. [12], we injected stress loads to two slightly different VNFs: a security IDS and a proxy load balancer. In this scenario, in contrast to what the authors presented, the IDS exhibited an atypical behavior: only one of the six distinct stresses affected the end-service throughput. Although this finding supported our claim for heterogeneity, neither the monitoring metrics nor the `guiltiness` were able to totally explain this anomaly. However, in favor of `guiltiness`, it was the sole metric able to explain all the cases where application performance dropped, even with false positives. In view of the above, our second conclusion is that *`guiltiness`can detect all performance problems caused by VNF processing capacities, but may exhibit false positives when stress loads do not generate impact on overall performance.*

As a first attempt to adapt the model to VNF particularities, we introduced a learning procedure. The procedure consists in a hybrid learning mechanism that combines linear regression with neural network abstractions to refine the weights used to calculate `guiltiness`. We then fed the learning mechanism with data from each evaluated scenario, and re-plotted `guiltiness` values using the learned weights. The results have showed an improvement in performance characterization in three of the four tested cases, *i.e.*, the cases where application throughput fell with stress loads. However, the IDS case with stress injections showed no significant improvement, bringing us to our last conclusion: *the weighted sum approach is suitable to customize `guiltiness`for scenarios' particularities, but research efforts are still required to properly adjust weight values.*

Regarding future work, the appropriate learning of weights is still an open issue, and thus, it is the next step towards developing a performance diagnosis model able to self-adjust to environments particularities. To this end, we envisage that learning approaches must find weights for each VNF individually, instead of generalizing values for the entire chain, as we have done in this paper.

Conflict of Interest

None.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nıvel Superior - Brasil (CAPES) - Finance Code 001, the Brazilian National Research and Educational Network (RNP), and the Brazilian National Council for Scientific and Technological Development (CNPq).

Appendix A. Proof of Theorem 4.1

We prove Theorem 4.1 through two lemmas. First, we show weights can normalize the residence time computation. Second, we demonstrate that the active percentage time can filter the residence time to consider the fraction of the time resource utilization relates to the processing of requests.

Lemma A.1. *Let R_m be the residence time of requests in a given computational resource m, computed as the quotient of the average service time S_m and the inverse resource utilization $1-\rho_m$. A normalized approximation of residence time R'_m can be achieved by replacing the service time S_m for a controlled weight w_m.*

Proof. By Little's Law [35], the average number of requests, L, in a resource is given by the equation: $L = W \times \lambda$, where W is the average waiting time in the resource (represented by the residence time R in our case) and λ is the arrival rate. Then, by simple algebra, we have:

$$L = W \times \lambda \iff L = R \times \lambda \iff R = \frac{L}{\lambda} \tag{8}$$

Also by Little's Law, the mean number of requests in a resource, L, can be computed by the ratio:

$$L = \frac{\rho}{1-\rho} \tag{9}$$

where ρ represents the resource utilization. By replacing L in Eq. (11), the average residence time of the resource can be computed as:

$$R = \frac{\frac{\rho}{1-\rho}}{\lambda} = \frac{\rho}{1-\rho} \times \frac{1}{\lambda} \tag{10}$$

As the resource utilization ρ can also be given by the product of arrival rate λ and the service time S [15], we can simply replace the first ρ in Eq. (10) by $\lambda \cdot S$, and define R as a function of S and ρ:

$$R = \frac{\lambda \cdot S}{1-\rho} \times \frac{1}{\lambda} = \frac{S}{1-\rho} \tag{11}$$

Next, to find the normalized residence time R', we can divide the actual residence time R by the maximum residence time $R_{\max}$:

$$R' = \frac{R}{R_{max}} = \frac{\frac{S}{1-\rho}}{\frac{S_{max}}{1-\rho_{max}}} = \frac{S}{1-\rho} \times \frac{1-\rho_{max}}{S_{max}} \tag{12}$$

where, $S_{\max}$ and $\rho_{\max}$ are the maximum allowed values of service time and utilization, respectively. Then, if we replace S by w_m in Eq. (11), and assume it as a normalization factor, we can solve the following equality:

$$\frac{w_m}{1-\rho_m} = \frac{S_m}{1-\rho_m} \times \frac{1-\rho_{max}}{S_{max}} \tag{13}$$

which, by simple algebra, makes:

$$w_m = \frac{S_m \cdot (1-p_{max})}{S_{max}} \tag{14}$$

as $S_m/S_{\max} \in [0, 1]$, and $p_{\max} \to 0.99$, then $w_m \in [0, 0.01]$. Thus, the approximate normalized residence time of requests in a resource m is:

$$R'_m = \frac{w_m}{1-\rho_m} \tag{15}$$

for $w_m \leq 0.01$ and $\rho_m < 1$. □

Lemma A.2. *The active percentage time A_m filters the residence time R_m to consider the approximate fraction of the time where a given resource m is effectively processing requests.*

Proof. Let $(\rho_t)_{t\in[1,n]} = (\rho_1, \rho_2, \ldots, \rho_n)$ be a resource utilization time series with n measurements, and $(A_t)_{t\in[1,n]} = (A_1, A_2, \ldots, A_n)$ be an resource active time series, where each value of A_t is defined by:

$$A_t(\rho_t) = \begin{cases} 1, & if \quad \rho_t \geq T \\ 0, & if \quad \rho_t < T \end{cases} \tag{16}$$

where, T is the threshold that defines when the resource is effectively processing traffic flow requests. Given the side effect of operating system processing [36], then $\rho_t \in (0, T)$ even when the arrival rate λ_t is 0. Thus, the fraction of the time where a resource m was active (*i.e.*, requests have been processed) is given by:

$$A_m = \frac{\sum_{t\in[0,n]} A_t}{n} \tag{17}$$

By the Little's law $\rho = \lambda \cdot S$, then, for the residence time computation purposes, $\rho_t \to 0$ when $\lambda_t \to 0$. Recall that by Eq. (11), the average residence time is given by:

$$R = \frac{S}{1-\rho} \tag{18}$$

Thus, given that $A_t = 0$ when $\rho_t < T$, then, Eq. (19) approximates the average residence time to the fraction of the time where resource m was effectively processing requests:

$$R_m = A_m \frac{S_m}{1-\rho_m} \tag{19}$$

□

Theorem 4.1. *Let $v = \{cpu, mem, NIC, I/O\}$ be a VNF with four major resource components: CPU, memory, I/O, and a network interface. The normalized residence time of packets in v, R'_v, is given by the weighted sum of normalized residence times of each of its resources.*

Proof. Lemma A.1 demonstrates that w_m replaces S_m in the computation of normalized residence time, and Lemma A.2 confirms that A_m delimits the residence time to consider only intervals of effective resource utilization. Then, the effective normalized residence time R'_m of requests in each resource m is given by:

$$R_m = A_m w_m \frac{1}{1-\rho_m} \tag{20}$$

Then, by the General Response Time Law [37], the normalized residence time in the VNF v is:

$$R'_v \quad = \sum_{m\in M} w_m . A_m . \frac{1}{1-\rho_m}, M = \{CPU, mem, NIC, I/O\} \tag{21}$$

□

Theorem 4.2. *Let R'_v be the normalized residence time of packets in VNF $v = \{cpu, mem, NIC, I/O\}$, given by the weighted sum of normalized residence times of each of its resources. The fifth term of Eq. (5)increases and reduces the `guiltiness`of a VNF v, G_v, according to the CPU activity related to the processing of queued packets in NIC.*

Proof. Let $A_{CPU,t}$ be a measurement of CPU activity in epoch t, such that $A_{CPU,t} > 0$. By Lemma A.2, if $p_{CPU} > T$ in epoch t, the ρ_{CPU} is due to the effective processing of requests. Assume that, in epoch t, $\lambda \to 0$ and $\rho_{NIC_{queue}} \geq 0$, then, that measured CPU consumption reflects the processing of queued requests. Following the assumption that packets are queued by contention [12], even when $\rho_m \to 1$, $\forall m \in \{CPU, mem, NIC, I/O\}$, $R'_v < G_v$, where G_v is the `guiltiness` of v when request are queued in the NIC buffer.

In turn, consider $A_{CPU} \times \rho_{NIC_{queue}}$ the fraction of time where CPU is active due to computation of queued requests and A_{CPU}, the overall fraction of the time where CPU was active. Thus, if we subtract A_{CPU} from $A_{CPU} \times \rho_{NIC_{queue}}$, and divide by $1 + \rho_{NIC_{queue}}$ (added by one to prevent zero division), we have the proportion of time where processing refers to non-queued requests:

$$\frac{A_{CPU} \times \rho_{NIC_{queue}} - A_{CPU}}{1 + \rho_{NIC_{queue}}} \tag{22}$$

Thus, by adding this ratio to G_v (Eq. (5)), and gauge its impact through w_5, we make $G_v \to 1$ when $\rho_{NIC_{queue}} \to 1$, and make $G_v \to (1 - W_5)$ when $\rho_{NIC_{queue}} \to 0$. □

References

[1] A. Nandugudi, M. Gallo, D. Perino, F. Pianese, Network function virtualization: through the looking-glass, Annal. des Telecommun./Annal. Telecommun. 71 (11–12) (2016) 573–581, doi:10.1007/s12243-016-0540-9.

[2] V.-G. Nguyen, T.-X. Do, Y. Kim, SDN and virtualization-based LTE mobile network architectures: a comprehensive survey, Wirel. Pers. Commun. 86 (3) (2016) 1401–1438, doi:10.1007/s11277-015-2997-7.

[3] R. Mijumbi, J. Serrat, J. Gorricho, S. Latre, M. Charalambides, D. Lopez, Management and orchestration challenges in network functions virtualization, IEEE Commun. Mag. 54 (1) (2016) 98–105, doi:10.1109/MCOM.2016.7378433.

[4] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F.D. Turck, R. Boutaba, Network function virtualization: state-of-the-art and research challenges, IEEE Commun. Surv. Tutor. 18 (1) (2016) 236–262, doi:10.1109/COMST.2015.2477041.

[5] P. Wang, J. Lan, X. Zhang, Y. Hu, S. Chen, Dynamic function composition for network service chain: model and optimization, Comput. Netw. 92 (2015) 408–418, doi:10.1016/j.comnet.2015.07.020. Software Defined Networks and Virtualization

[6] A. Muhammad, M. Fiorani, L. Wosinska, J. Chen, Joint optimization of resource allocation for elastic optical intra-datacenter network, IEEE Commun. Lett. 20 (9) (2016) 1760–1763.

[7] F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, O.C.M.B. Duarte, Orchestrating virtualized network functions, IEEE Trans. Netw. Serv. Manage. 13 (4) (2016) 725–739, doi:10.1109/TNSM.2016.2569020.

[8] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: challenges and opportunities for innovations, Commun. Mag., IEEE 53 (2) (2015) 90–97.

[9] Y. Li, M. Chen, Software-defined network function virtualization: a survey, IEEE Access 3 (2015) 2542–2553, doi:10.1109/ACCESS.2015.2499271.

[10] R. Jain, S. Paul, Network virtualization and software defined networking for cloud computing: a survey, Commun. Mag., IEEE 51 (11) (2013) 24–31.

[11] L. Cao, P. Sharma, S. Fahmy, V. Saxena, Nfv-vital: a framework for characterizing the performance of virtual network functions, in: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015, pp. 93–99, doi:10.1109/NFV-SDN.2015.7387412.

[12] W. Wu, K. He, A. Akella, Perfsight: Performance diagnosis for software dataplanes, in: Proceedings of the 2015 ACM Conference on Internet Measurement Conference, 2015, pp. 409–421.

[13] P.J. Denning, J.P. Buzen, The operational analysis of queueing network models, ACM Comput. Surv. (CSUR) 10 (3) (1978) 225–261.

[14] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action, Cambridge University Press, 2013.

[15] N.J. Gunther, Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services, Springer Science & Business Media, 2007.

[16] B. Yi, X. Wang, K. Li, S. k. Das, M. Huang, A comprehensive survey of network function virtualization, Comput. Netw. 133 (2018) 212–262, doi:10.1016/j.comnet.2018.01.021.

[17] R.J. Pfitscher, E.J. Scheid, R.L. dos Santos, R.R. Obelheiro, M.A. Pillon, A.E. Schaeffer-Filho, L.Z. Granville, DReAM - A Distributed Result-Aware Monitor for Network Functions Virtualization, in: 2016 IEEE Symposium on Computers and Communication (ISCC), 00, 2016, pp. 663–668, doi:10.1109/ISCC.2016.7543813.

[18] A. Gember, A. Krishnamurthy, S.S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, V. Sekar, Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud, Technical Report, 2013.

[19] G.A. Gallardo, B. Baynat, T. Begin, Performance Modeling of Virtual Switching Systems, in: 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016, pp. 125–134, doi:10.1109/MASCOTS.2016.22.

[20] K. Suksomboon, M. Fukushima, S. Okamoto, M. Hayashi, A Dilated-CPU-Consumption-Based Performance Prediction for Multi-Core Software Routers, in: 2016 IEEE NETSOFT - IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services, 2016, pp. 193–201, doi:10.1109/NETSOFT.2016.7502413.

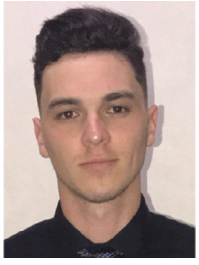
[21] S. Koizumi, M. Fujiwaka, SDN + Cloud Integrated Control with Statistical Analysis and Discrete Event Simulation, in: 2015 International Conference on Information Networking (ICOIN), 2015-Janua, 2015, pp. 289–294, doi:10.1109/ICOIN.2015.7057898.

[22] R.J. Pfitscher, A.S. Jacobs, E.J. Scheid, M.F. Franco, R.L. d. Santos, A.E. Schaeffer-Filho, L.Z. Granville, A Model for Quantifying Performance Degradation in Virtual Network Function Service Chains, in: 2018 NOMS - IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–9, doi:10.1109/NOMS.2018.8406268.

[23] P. Naik, D.K. Shaw, M. Vutukuru, Nfvperf: Online Performance Monitoring and Bottleneck Detection for NFV, in: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2016, pp. 154–160.
[24] C. Sauvanaud, K. Lazri, M. Kaaniche, K. Kanoun, Anomaly Detection and Root Cause Localization in Virtual Network Functions, in: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016, pp. 196–206, doi:10.1109/ISSRE.2016.32.
[25] D.A. Menasce, V.A. Almeida, L.W. Dowdy, L. Dowdy, Performance by Design: Computer Capacity Planning by Example, Prentice Hall Professional, 2004.
[26] V. Jacobson, Congestion avoidance and control, SIGCOMM Comput. Commun. Rev. 18 (4) (1988) 314–329, doi:10.1145/52325.52356.
[27] R.J. Pfitscher, M.A. Pillon, R.R. Obelheiro, Customer-oriented diagnosis of memory provisioning for iaas clouds, ACM SIGOPS Oper. Syst. Rev. 48 (1) (2014) 2–10.
[28] R. van Riel, Measuring Resource Demand on Linux, in: 2006 Linux Symposium, 2006, p. 287.
[29] Y. Wang, Q. Zhao, D. Zheng, Bottlenecks in production networks: an overview, J. Syst. Sci. Syst. Eng. 14 (3) (2005) 347–363.
[30] E.G. Specification, Digital Cellular Telecommunications System (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; IP Multimedia Subsystem (IMS), Technical Report, European Telecommunications Standards Institute, 2013.
[31] Clearwater, Project Clearwater, 2018. http://www.projectclearwater.org/.
[32] A.T. Kabakus, R. Kara, A performance evaluation of in-memory databases, J. King Saud Univ. - Comput. Inf. Sci. 29 (4) (2017) 520–525, doi:10.1016/j.jksuci.2016.06.007.
[33] D. Kriesel, A brief introduction to neural networks, 2007. http://www.dkriesel.com/en/science/neural_networks.
[34] R. Cziva, D.P. Pezaros, Container network functions: bringing NFV to the network edge, IEEE Commun. Mag. 55 (6) (2017) 24–31.
[35] J.D. Little, OR FORUM—little'S law as viewed on its 50th anniversary, Oper. Res. 59 (3) (2011) 536–549.
[36] A. Sheoran, X. Bu, L. Cao, P. Sharma, S. Fahmy, An Empirical Case for Container–Driven Fine-Grained VNFResource Flexing, in: Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on, IEEE, 2016, pp. 121–127.
[37] R. Jain, The Art of Computer Systems Performance Analysis, John Wiley & Sons, 2008.

Ricardo José Pfitscher is a postdoctoral researcher at the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil and holds a Ph.D. in computer science from the same university (2019). In 2014 he received a M.Sc. degree in Applied Computing of the Santa Catarina State University (UDESC), Brazil, from which he also held a Bachelor degree in computer science (2009). His topic of interest include network management, Network Functions Virtualization (NFV), and network performance.

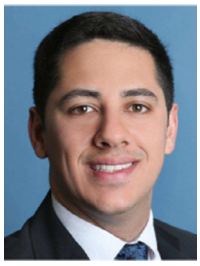
Arthur Selle Jacobs is a second year PhD student in Computer Science, under the supervision of Prof. Dr. Lisandro Granville, at the Federal University of Rio Grande do Sul, in Brazil. His research interests include network management, Network Functions Virtualization, Self-Driving Networks, programmable networks and artificial intelligence. Arthur is currently researching the use of machine learning for network management, as a mean to achieve Self-Driving Networks.



Luciano Zembruzki achieved his B.Sc. Degree in Computer Science from the Integrated Regional University (URI) - Campus of Frederico Westphalen, in 2017. He is an M.Sc. student in computer science at the Institute of Informatics (INF) of the Federal University of Rio Grande do Sul (UFRGS), Brazil. His research interests include Network Management, Network Functions Virtualization (NFV) and Software-Defined Networking (SDN).

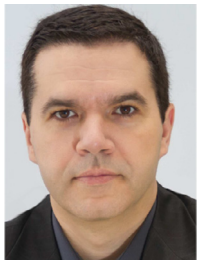
Ricardo Luis dos Santos is a professor at the Federal Institute of Rio Grande do Sul (IFRS) in Brazil. He holds Ms.C. (2012), and Ph.D. (2018) degrees in computer science, from Federal University of Rio Grande do Sul (UFRGS), both under the supervision of Professor Dr. Lisandro Zambenedetti Granville. In the recent past, He worked at a project named Malia: Management of Applications Lifecycle through IT Analytics, funded by Hewlett-Packard R&D Brazil, in cooperation between UFRGS and HP Labs Bristol and Palo Alto. His research interests include network virtualization, network programmability, network marketplaces, and future Internet.

Eder John Scheid is pursuing his Ph.D. under the supervision of Prof. Dr. Burkhard Stiller at the University of Zurich (UZH). He is also a Research Assistant at the Communication Systems Group (CSG) since December 2017. Eder holds an MSc. Degree in Computer Science from the Federal University of the Rio Grande do Sul (UFRGS), which he obtained in 2017 under the supervision of Prof. Dr. Lisandro Zambenedetti Granville. His topics of interest include network management, SLA management, Network Function Virtualization (NFV), and blockchain-based smart contracts.



Muriel Figueredo Franco is pursuing his Ph.D. under the supervision of Prof. Dr. Burkhard Stiller at the University of Zurich (UZH). He is also a Research Assistant at the Communication Systems Group (CSG). He holds an M.Sc in Computer Science from the Federal University of the Rio Grande do Sul (UFRGS) under the supervision of Prof. Dr. Lisandro Granville and obtained a B.Sc in Computer Science from the Federal University of Pelotas (UFPEL). His research interests include network functions virtualization, network management, information visualization, and blockchain.

Alberto Schaeffer-Filho holds a Ph.D. in Computer Science (Imperial College London, 2009) and is Associate Professor at Federal University of Rio Grande do Sul (UFRGS), Brazil. From 2009 to 2012 he worked as a research associate at Lancaster University, UK. Dr. Schaeffer-Filho is a CNPq-Brazil Research Fellow and his areas of expertise are network/service management, network virtualization and software-defined networks, policy-based management, and security and resilience of networks. He has authored over 50 papers in leading peer-reviewed journals and conferences related to these topics, and also serves as TPC member for important conferences in these areas, including: CNSM (2018), IEEE/IFIP NOMS (2018), CNSM (2017), IFIP/IEEE IM (2017), IEEE CCNC (2017), and IEEE INFOCOM CNTCV Workshop (2017). He is the general chair for SBRC 2019, co-chair for IEEE ICC 2018 CQRM Symposium, demo co-chair for IFIP/IEEE IM 2017, and workshop co-chair for ManSDN/NFV 2015. He is also a member of the IEEE.



Lisandro Zambenedetti Granville is an Associate Professor at the Institute of Informatics of the Federal University of Rio Grande do Sul. He served as the TPC Co- Chair of IFIP/IEEE DSOM 2007 and IFIP/IEEE NOMS 2010, the General Co-Chair of IFIP/IEEE CNSM 2014, the TPC Co-Chair of IEEE NetSoft 2018, and the TPC Vice-Chair do IEEE ICC 2018. He is the current president of the Brazilian Computer Society. His interests include network management, software-defined networking, and network functions virtualization