A fault correlation approach to detect performance anomalies in Virtual Network Function chains

Domenico Cotroneo, Roberto Natella and Stefano Rosiello
Università degli Studi di Napoli "Federico II", Italy
{cotroneo, roberto.natella, stefano.rosiello}@unina.it

***Abstract*—Network Function Virtualization is an emerging paradigm to allow the creation, at software level, of complex network services by composing simpler ones. However, this paradigm shift exposes network services to faults and bottlenecks in the complex software virtualization infrastructure they rely on. Thus, NFV services require effective anomaly detection systems to detect the occurrence of network problems. The paper proposes a novel approach to ease the adoption of anomaly detection in production NFV services, by avoiding the need to train a model or to calibrate a threshold. The approach infers the service health status by collecting metrics from multiple elements in the NFV service chain, and by analyzing their (lack of) correlation over the time. We validate this approach on an NFV-oriented Interactive Multimedia System, to detect problems affecting the quality of service, such as the overload, component crashes, avalanche restarts and physical resource contention.**

***Index Terms*—NFV; fault correlation; overload; quality of service; anomaly detection;**

I. Introduction

A key challenge for telecom operators and service providers is to efficiently deploy rich network services, and to optimize network resources to improve customer's quality of experience. To achieve these goals, it is important to be able to dynamically combine single network functions, such as IP forwarding, Network Address Translation, Traffic Shaping, Deep Packet Inspection Functions, into *service function chains*, to create service-specific traffic forwarding policies [1].

Network Function Virtualization (NFV) seems to be the promising answer to this problem that promotes a paradigm-shift, from network hardware equipment to *virtual network functions* (VNFs). More specifically, NFV aims to leverage standard IT virtualization technology to consolidate them in industry-standard high volume servers, switches, and storage; and to take advantage of orchestration and monitoring solutions used for cloud computing [2], [3].

Being a software- and a cloud-based solution, NFV inherits the threats coming from these domains and, in particular, the relatively-low reliability and performance of Off-The-Shelf hardware and software components [4], [5], [6], [7]. For this reason, the main consortia behind NFV, including the ETSI and OPNFV, pointed out the need for *fault correlation* algorithms to detect the occurrence of network problems from their symptoms [8], [9]. An important class of problems in this area is represented by faults that restrict the capacity of the whole VNF service chain. Examples of these faults are bottlenecks, virtual machines crashes, hardware faults, overload conditions. We name these faults as *performance anomalies*.

By looking at the existing literature, the classical approach to detect such anomalies in cloud infrastructures is based on anomaly detection [10]. However, these techniques suffer from limited flexibility, as they require to train classification algorithms with data obtained from extensive test campaigns or with historical data [11], [12], [13]. Although there are few recent studies that adopted these approaches in the context of NFV systems [14], [15], [16], the need of data training could be unattainable for the following reasons. First, since new service function chains have to be delivered in a short time, it is very difficult to perform test campaigns to get training data. Second, historical data cannot be used because each service has different characteristics, thus it is very difficult to tailor previous datasets to new contexts. Furthermore, other studies on anomaly detection used threshold-based classifiers, which are easier to deploy. Even in this case, such approaches still need to be calibrated for the specific service, which is very difficult to achieve.

In this paper, we present a novel approach to identify *performance anomalies* in VNF service chains. The key feature of the proposed approach is that it neither requires to train a model, nor to calibrate a threshold to identify performance anomalies inside the VNF chain. Instead, we take advantage from the fact that the VNF chain can be seen as a multistage pipeline, where the output of a network function is the input of the next one. Therefore, the resource utilization metrics of VNFs in a service chain have a strong dependency (e.g., the outgoing network traffic from the first stage is related to the CPU load on the second stage of the chain). Thus, our approach collects metrics from connected VNF stages, as they are naturally correlated. Then, it analyzes their co-variation over time to infer potential performance anomaly at each stage of the chain. Our approach can also be adopted in large-scale NFV systems with the presence of load-balancing and replication.

We evaluated the approach on a real world NFV-oriented IP Multimedia System (IMS), namely Clearwater [17], that has been designed to support massive horizontal scalability. Clearwater adopts popular cloud computing design patterns and technologies, including the OpenStack/KVM virtualization stack. To analyze performance anomalies in the context of this IMS system, we applied a set of test scenarios, which encompass the most common anomalies [18], [19], including:

- A sudden increase of load from subscribers;
- The crash of subsets of VNFs nodes (e.g., due to bugs in the VNF software);

- Avalanche restarts, triggered by the failover of VNF instances;
- Faults at hardware level, causing contention on physical resources.

We show that the proposed detection approach covers all these scenarios, and presents no false positives during the normal behavior of the system.

The paper is structured as follows. Section II discusses the problem of anomaly detection and the related work. Section III presents our anomaly detection approach. Section IV introduces the Clearwater VNF chain and the architecture of the experimental testbed. Section V evaluates the performance of the detector, in terms of accuracy and latency. Section VI concludes the paper.

II. Related Work

Continuous, online monitoring and analysis is a key component for managing cloud infrastructures. The analysis of performance metrics and resource utilization enables a better understanding of application and system behaviour, helps to tune configurations to meet application SLA requirements, and provides insights for troubleshooting.

The most common cloud monitoring and dashboards systems, such as Amazon CloudWatch [20] and Google Stack-Driver [21] monitor the system on per-VM basis and allow to setup and customize simple detection rules (e.g., thresholds on monitored metrics) and trigger maintenance task (e.g., scaling, rebooting). More advanced commercial products, such as Datadog [22], also implement simple data mining features, using seasonal auto regression, trend detection, online adaptive learning, and statistical distribution models. However, since the products focus on symptoms on individual VM instances, they are prone to false alarms: for instance, without any knowledge about the specific applications, they cannot discern if a drop in the load of a VM is caused by a sudden workload decrease in the whole system or by an undetected fault in some component. As discussed later, our approach takes into account the nature of NFV applications (based on pipeline processing of high-volumes of packet streams) to detect these scenarios: it analyzes the correlation of metrics from neighbour VMs in the VNF service chain, to distinguish licit workload variations from faulty conditions that affect the quality of service.

In general, anomaly detection systems aim to automate the discovery and classification of problems by analyzing these data, and checking whether the system behaves accordingly to what is expected. In order to characterize such behaviors, classical approaches use machine learning techniques, such as random forests classifiers [23], neural networks [12], automatic rule learning and fuzzy logic [11], unsupervised clustering [24], [25]. Most of the anomaly detection research has applications in intrusion and misuse detection. Instead, our approach generally considers *performance anomalies* caused by faults in a VNF chain, regardless that they are accidental or intentionally induced by intrusion/misuses (e.g., by forcing software crashes or by overloading resources in order to cause a denial of service). In general, we focus on faults that impact on the quality of service in terms of responsiveness (e.g., latency) and availability (e.g., request timeouts or failures); instead other forms of intrusion/misuse that do not cause performance anomalies are out of the scope of this work.

More recently, these approaches have been applied in the context of NFV applications. Miyazawa et al. [14] proposed a distributed architecture to perform fault detection using unsupervised data clustering techniques and self-organizing maps. In [16], Sauvanaud et al. suggest a supervised learning approach. They perform fault injection experiments in a NFV testbed to collect labeled monitoring data, from both hypervisors and virtual machines instances. Then, they build a classifier using the random forest algorithm, showing high detection accuracy and low false positive. However, both approaches require retraining the models in case of changes in the hardware or software configuration, workload patterns or other influencing factors.

Moreover, all the previous techniques require training models with data coming from extensive test campaign or historical data. However, in the context of NFV, the needed training data may be unattainable, since service function chains must be delivered in a short time (thus limiting the amount of tests for getting training data) and are tailored for each specific service (thus limiting the usefulness of historical data). For the same reasons, most anomaly detection systems used in practice are threshold-based classifiers, which are ease to deploy and provide an acceptable quality of detection (in terms of accuracy and latency), but they still need to be calibrated for the specific service. By contrast, our approach relates metrics coming from multiple interconnected VNF, without the need to build representative datasets to train detection models. Moreover, the approach is robust and easy to deploy as we analyze the correlation between the metrics, rather than comparing metrics with some absolute reference value that would depend on the type of VNF software, the VM sizing, etc. (e.g., 80% CPU utilization could be normal in some contexts, but could be a performance anomaly in others).

In the field of classical (i.e., non-virtualized) network management systems, *alarm correlation* [26] between multiple distributed entities is widely used to detect faults and isolate the causes across a big number of network appliances interconnected [27], [28]. In [29], Kliger et al. defined the networking graph as a causality graph on which nodes can be marked as *problems* or *symptoms*, and use event correlation to find causal relations among the events. They demonstrate that this approach is resilient to high rates of symptom loss (i.e., false negatives) and false alarms. Similarly, we show that it is possible to identify causal relationships in the VNF service chaining model between the VNF instances in the network. We apply the correlation analysis to the monitoring data to recognize symptoms of problems in the network.

III. Fault correlation approach

Our approach is based on the idea that a network packet or request follows a chain of VNFs, as shown by the VNF graph in Fig. 1. Each VNF in the graph can have a different number of replicas, that are scaled according to a preliminary capacity planning or to cloud elasticity. The load is balanced across all the replicas of the VNF.

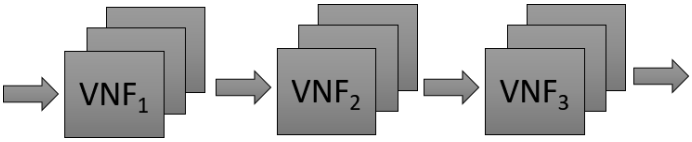


Fig. 1: A pipeline of network functions

In this architecture, there are metrics from multiple stages that are naturally correlated (e.g., the outgoing network traffic of the first stage and the CPU load of the second stage). If resource utilization (e.g., CPU, memory, ...) increases in a VM hosting a network function, an increase should also occur in VMs hosting the subsequent VNF in the service chain. If this is not the case, a VM is obstructing the network flow, causing a performance anomaly. Thus, we analyze the correlation in the time between metrics from two distinct network functions to infer the service health status. Fig. 2 shows the vCPU load of two connected network functions (i.e., the output traffic of VNF$^{(A)}$ is processed by VNF$^{(B)}$). When VNF$^{(A)}$ uses all the available CPU time (e.g., due to an overload condition or a software fault), its throughput start decreasing. As a consequence, the VNF$^{(B)}$ receives less traffic to process and the CPU load on this VNF decreases. This condition can be detected noting that there is a window of time in which the CPU load on VNF$^{(A)}$ increases and the CPU load on VNF$^{(B)}$ decreases. In correspondence of that window, the two time series become negatively correlated. We consider this condition a symptom of a performance anomaly.

The algorithm 1 raises an alarm when an anomaly is detected between a pair of connected VNF stages, namely VNF$^{(A)}$ and VNF$^{(B)}$, with VNF$^{(A)}$ preceding VNF$^{(B)}$ in the chain.

The algorithm takes a window Δt of n samples of a time series describing a resource utilization in the time (e.g., the CPU usage) from both the VNF$^{(A)}$ and the VNF$^{(B)}$ and

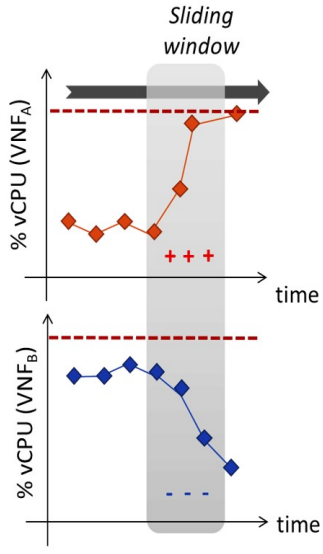


Fig. 2: Running correlation between two VNFs

Algorithm 1: Fault correlation algorithm

Data: n: sampling window size
Data: Δt: $(t-n,\ldots,t)$ time window
Set: counter=0
begin
  foreach *replica* h *of* $VNF^{(A)}$ **do**
    foreach *replica* k *of* $VNF^{(B)}$ **do**
      $\rho_{h,k} = pearson(VNF_h^{(A)}(\Delta t), VNF_k^{(B)}(\Delta t))$
      $\boldsymbol{D}_k = D(\rho_{h,k})$
    $\overline{D} = mean(\boldsymbol{D}_k)$
    if $\overline{D} > 0.5$ **then**
      $counter++$
  if $counter > |VNF^{(A)}|/2$ **then**
    raise alarm

computes the correlation according to the Pearson's index ρ (i.e., equation 1) as the covariance $\sigma_{X,Y}$ of the two variables divided by the product of their standard deviations σ_X and σ_Y. Then, it ranks the correlation by computing a discrete score, namely D-score, according to equation 2.

$$\rho(X,Y) = \frac{\sigma_{X,Y}}{\sigma_X \cdot \sigma_Y} \tag{1}$$

$$D(\rho) = \begin{cases} 1, & \text{if } -1.0 \le \rho \le -0.7 \\ 0.75 & \text{if } -0.7 < \rho < -0.3 \\ 0.50 & \text{if } -0.3 \le \rho \le +0.3 \\ 0.25 & \text{if } +0.3 < \rho < +0.7 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

A zero D-score indicates a strong positive correlation between the two metrics considered, while a D-score equal to 1 indicates a strong negative correlation among them. Intermediate D-score values indicate weak correlations (i.e., $D = 0.75$, $D = 0.25$) or absence of linear correlation ($D = 0.5$). The values used in this equation are widely used in statistics to evaluate the strength of the correlation [30], [31], and do not depend on the specific system to be monitored.

Since each VNF in the chain can have multiple active replicas, the Algorithm 1 processes windows of samples gathered from each replica, and raises an alarm if more than half of the spare nodes exhibit a correlation anomaly. More precisely, for each replica h of the VNF$^{(A)}$ the algorithm computes the D-score with all the replicas k of the VNF$^{(B)}$. If the average D-score is greater than 0.5 (indicating a negative correlation) we account a possible anomaly by increasing the *counter* variable. When all the D-scores are evaluated, if the *counter* is greater than the half of the number of VNF$^{(A)}$ replicas, a majority of VNF instances exhibits a correlation anomaly and an alarm is raised.

The choice to wait for a feedback from a majority of nodes prevents false alarms that may be due to sporadic variations of load balancing across replicas in the same VNF stage. However, it is important to note that the algorithm is not

limited to detect problems caused by multiple nodes; it is still able to detect performance anomalies caused by a single node. In the case of a single faulty node, the algorithm will detect an anomaly for the correlations between the faulty node (for example, $\text{VNF}_3^{(B)}$) and all the replicas of the previous VNF stage (for example, $\text{VNF}_1^{(A)} \ldots \text{VNF}_n^{(A)}$) that access the faulty node. In general, the algorithm is designed to detect performance anomalies that have an impact on the capacity of a VNF stage, which can be either caused by single or multiple failures.

By executing the Algorithm 1 on sliding windows of n samples, at time t, we compute the correlation between the samples from time $t-n$, $t-(n-1)$, $t-(n-2)$, $\ldots$, $t-1$. Samples are collected periodically every p seconds.

The configuration of the window, i.e., the size n and the period of sampling p, is driven by the speed at which performance anomalies are expected to happen. In our context, according to the empirical experience of industries in the ETSI consortium, performance anomalies such as avalanche restarts and overloads are expected to develop within less than 30 seconds [8]. Therefore, the n and p should be chosen such that $n \cdot p \leq 30$, as this represents a lower bound on the detection latency. For example, to have an high enough resolution to notice variations of the resource utilization metrics, and enough values to compute the correlation, the period p should be in the order of few seconds (e.g., $p = 2$s).

Of course, the need to configure an high sampling frequency may expose our approach to false alarms, that may be caused by random fluctuations of measurements. To make the approach robust to the high sampling frequency and to the choice of these parameters, we discuss two strategies to mitigate the downside of this choice:

1) use of smoothing functions on the time series to reduce the noise in the data;
2) filtering the negative correlation events according to the variance contained in the sampling window.

The first strategy requires to pre-process the sliding window with a smoothing function. Multiple algorithms can be adopted to this purpose. In Section V we compare the detection accuracy and the detection latency using three different types of smooth: (1) Running Moving Average (RMA) to lower the impact of values too distant from the average, (2) Running Moving Median (RMM) to lower the impact of values too distant from the median, and (3) Exponential Moving Average (EMA) to lower the impact of older samples in the current sampling window.

The second technique prevents spurious alarms that may occur when there is a negative correlation, but the variability of the measurements is very small and has been likely caused by random fluctuations (e.g., by chance, one of the time series may slightly increase due to random fluctuations, and at the same time the other time series may decrease). Thus, we detect a "representative" anomaly if both there is a negative correlation, *and* the variations of the measurements is large enough to reflect some event that may be occurred in the VNF (e.g., a fault or a workload change). To this purpose, we compute the *coefficient of variation* (cv) on a window of samples W, as the the ratio between its standard deviation σ_W and its mean μ_W, according to the equation 3. Then, a correlation between the time series is taken into account only if the cv is non-negligible, i.e., the variation exceeds the average value of the metric (typically, a coefficient of variation below 10% denotes that variations are very small [32] and could be considered random). This filter has also the advantage to exclude the sampling windows in which the chosen metric remains constant; in this specific case, the correlation index is undefined.

$$cv(W) = \frac{\sigma_W}{\mu_W} \tag{3}$$

Fig. 3 shows an example of this second approach, by considering the vCPU consumption of two consecutive VNF in the pipeline. Before $t = 200s$ there are small variations in the vCPU utilization that are not representative of a change in the workload. After $t = 200s$ an increase in the load brings the VNFx in overload, while reducing the load on VNFy by 23% as a side effect (which is a consequence of resource saturation at VNFx). In correspondence of this negative correlation, there is a peak in the coefficient of variation of vCPU utilizations in both the VNFs. Thus, we consider this correlation an anomaly.

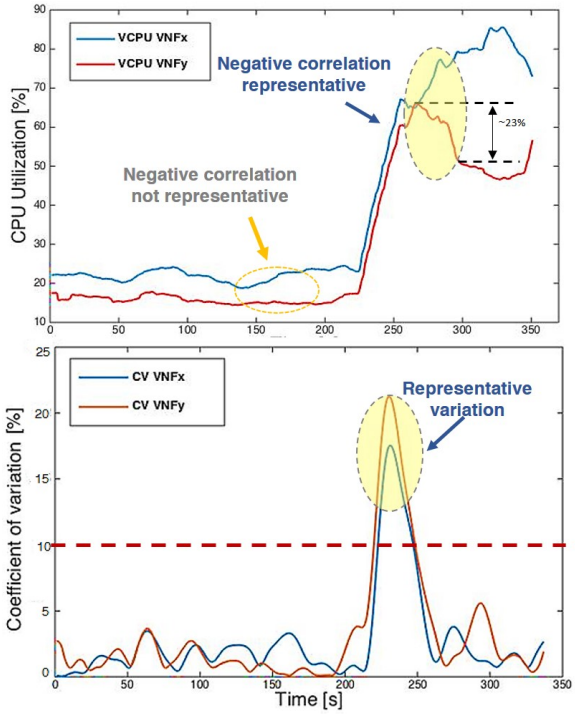


Fig. 3: Coefficient of variation filter

IV. The IMS Case Study

The Clearwater IMS [17] is an open-source implementation of the IMS core standard [33]. IMS functions are implemented in software and packaged in VMs, and are designed to take full advantage of virtualization and cloud computing technology. All components can scale out horizontally using simple, stateless load-balancing based on DNS. Moreover, Clearwater follows common design patterns for scalable and reliable web services, by keeping most components largely stateless,

and by storing long-lived state in clustered data stores. Clearwater is a large software project, mostly written in C++ and Java, and including several subsystems. The architecture of Clearwater core is showed in Fig. 4, and includes the following components:

- ***Bono*** (P-CSCF): The Bono nodes are the first point of contact for an UE (User Equipment), and they represent the edge proxy providing P-CSCF standard interfaces to IMS clients.
- ***Sprout*** (S-CSCF and TAS): The Sprout nodes are SIP registrars and authoritative routing proxies. These nodes implement the S-CSCF and I-CSCF interfaces of the IMS standard. Furthermore, they implement a distributed cache, using Memcached [34], for storing registration data and other short-lived information.
- ***Homestead***: The Homestead nodes are redundant mirrors for the HSS (Home Subscriber Server) data store, using Apache Cassandra [35], for retrieving authentication credentials and user profile information. HSS mirrors are part of both the S-CSCF and I-CSCF interfaces, and provide Web services (over HTTP) to the Sprout layer.
- ***Homer***: A Homer node is a XML Document Management Server (XDMS) to store service settings documents for each user of the system, using Apache Cassandra as the data store.
- ***Ralf*** (Rf-CTF): The Ralf nodes provide charging and billing functions, used by Bono, Sprout and Homestead nodes to report events occurring when the CSCF chain is traversed.

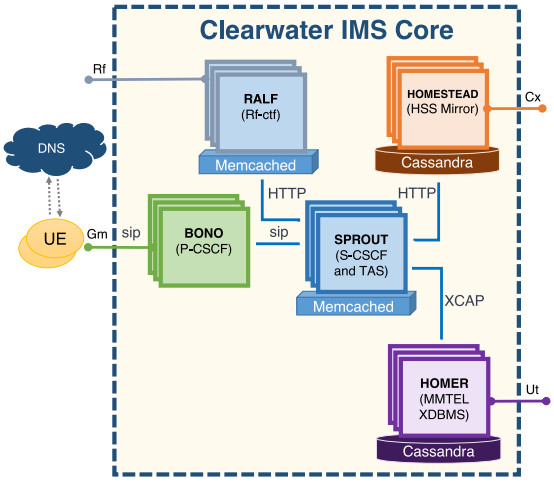


Fig. 4: Architecture of the Clearwater IMS.

The experimental testbed (Fig. 5) consists of four host machines: three Dell PowerEdge R520 servers, equipped with two 8-Core 2.2 GHz Intel Xeon CPU, 64GB DDR3 RAM, two 500GB SATA HDD, two 1-Gbps Ethernet NICs, 8-Gbps Fiber Channel HBA; one Dell PowerEdge R320 server with a 4-Core 2.8 GHz Intel Xeon CPU, 8GB DDR3 RAM, two 500GB SATA HDD, two 1-Gbps Ethernet NICs, 8-Gbps Fiber Channel HBA; A PowerVault MD3620F disk array with 4TB of network storage with a 8-Gbps Fiber Channel link.

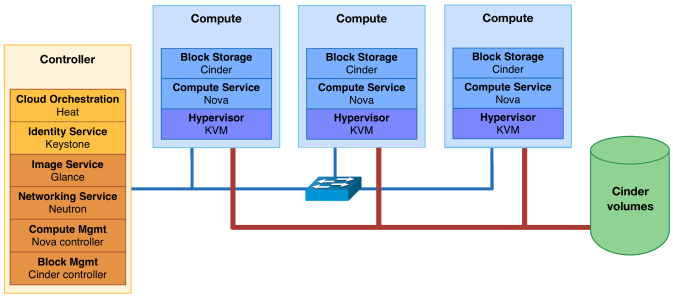


Fig. 5: Experimental testbed.

The hosts are connected to a 1-Gbps Ethernet network for general-purpose traffic, and another 1-Gbps Ethernet network for management traffic. The virtual disks of VMs are stored on three distinct GlusterFS partitions of the PowerVault SAN, which are mounted on the hosts through the Fiber Channel link.

The hosts are configured with CentOS Linux 7 and the KVM hypervisor. The testbed is managed using the *OpenStack* virtualization platform, version Juno [36]. The Dell PowerEdge R320 serves as OpenStack Controller and Network node; the three Dell PowerEdge R520 servers represent the OpenStack Compute and Storage nodes, and run the VMs of the Clearwater IMS. The OpenStack services include: *Nova*, which manages the compute domain; *Neutron*, which manages virtual networks among VMs; *Cinder*, which controls the lifecycle of VM volumes; *Glance*, which manages the cloud images of VMs; *Heat*, which orchestrates, through a native REST API, the virtual IMS deployment; *Horizon*, which supports the Web-based management dashboard.

To determine the number of VMs that had to host specific network services, we made some preliminary capacity tests. We defined a deployment configuration capable to handle *500,000 subscribers* (i.e. **the engineered capacity**) corresponding to (i) 90,000 registration attempts per minute, and (ii) 8,000 call attempts per minute. At this level the average CPU utilization is 80% in all the Clearwater VMs and all the requests are correctly served by the system. The number and type of VMs hosting services is detailed in TABLE I. Each VM hosts a single VNF.

Other VMs are used to generate the IMS workload. Such machines run the **SIPp traffic generator**. Each SIPp instance generates SIP traffic towards a specific P-CSCF instance. Each couple of subscribers will attempt to register or renew the registration every 5 minutes, on average. After a successful registration, one can attempt to setup a call to the other (with 16% of probability) or remain idle until the next registration renewal (with 84% of probability). The call hold time is, by default, 60 seconds. 10 SIPp are used for generating the initial load of 500,000 subscribers in 10 minutes (*Initial Ramp-up period*). To generate the overload conditions in our test scenarios, we run 40 additional SIPp VMs.

TABLE I: Clearwater VMs deployment configuration.

Service	Clearwater Node Name	# of VMs	Flavor Details
Edge Proxy (P-CSCF)	Bono	10	VCPUs: 1 RAM: 2GB Disk Size: 5GB
SIP Router (I/S-CSCF)	Sprout	10	VCPUs: 1 RAM: 2GB Disk Size: 5GB
HSS Mirror	Homestead	5	VCPUs: 1 RAM: 4GB Disk Size: 80GB
Rf CTF	Ralf	4	VCPUs: 1 RAM: 2GB Disk Size: 5GB
XDMS (MMtel services)	Homer	2	VCPUs: 1 RAM: 4GB Disk Size: 100GB
Name service (DNS)	-	1	VCPUs: 1 RAM: 2GB Disk Size: 5GB
Workload generator (SIPp)	-	10-40	VCPUs: 1 RAM: 2GB Disk Size: 100GB

V. Experimental evaluation

In our experimentation, we study the ability of the detection algorithm to identify performance anomalies, and to avoid false positives. In the context of the IMS case study, such anomalies cause the failure of some user registrations and/or some call setup requests. On the opposite, when there are no faults affecting the quality of service, all the registrations and the call setups are correctly processed by the system. Thus, we use the SIPp workload generator to check at client-side the success of such requests, in order to evaluate the outcome of the detection algorithm.

In our evaluation, we consider test scenarios that involve service failures of the IMS system. To have meaningful test scenarios, we induce performance anomalies that cause service failures, that is, the quality of service experienced by clients degrades, either in terms of throughput (i.e., there should be a gap between the request rate from the client, and the throughput of traffic served by the IMS) and latency (i.e., there is a long delay between a request and the corresponding results). In quantitative terms, we cause service failures where the throughput is less than 90% of the request rate for more than 5 seconds, and the 90th-percentile of the request latency is lower than 250ms. The requests that violate the latency requirements are signalled either by the system (i.e., with SIP 500 messages) or by the client (i.e., in case of timeout events). In both cases, these requests are marked as failed and are not accounted in the overall throughput. Thus, in our discussion, we focus on presenting the throughput metric, as in all tests the latency violations were always accompanied by throughput violations during the same periods.

To assess the detection algorithm, we consider a set of overload scenarios (caused by workload surges and faults), and perform r repeated experiments for each scenario, where we evaluate the number of times the algorithm is able to detect the overload. The following *Detection Outcome*s are considered:

- *Overload not detected*: the algorithm detected the overload no more than in 20% of the experiments;
- *Overload detected*: in at least 80% of the experiments, the algorithm was able to detect the overload;
- *Unreliable detection*: in the other cases.

To summarize the detection outcomes across different scenarios, we compute the *Overall Detection Coverage*, which we define as the percentage of the scenarios where the detection outcome is *overload detected*.

Another requirement of NFV services is that anomalous conditions have to be detected as soon as possible, so that mitigation mechanisms can be quickly activated, and the impact on the quality of service can be reduced. Thus, as a further metric for the assessment of the detection algorithm, we consider the *Detection Latency*, which is defined as the time between the occurrence of an overload condition (i.e., the moment at which users' registrations and/or calls start failing) and the detection of such condition by the algorithm.

Finally, we consider the rate of false alarms that are raised by the detection algorithm. To this aim, we perform experiments without anomalies, and keep track of any (false) alarms raised by the algorithm during the experiment.

Ideally, to be deployed in production environments according to the feedback from our industrial partners, our proposed algorithm should have a quick detection latency and no false positives, and a reasonably high detection coverage; this can be a challenging goal considering that we do not rely on any preliminary calibration of thresholds (e.g., we do not fix a minimum or maximum value for CPU or bandwidth utilization in our algorithm).

We applied the proposed approach by correlating the **CPU utilization** of VNFs in the service chain. One of the reasons why we focus on this metric is that, in NFV services, the network consumption is highly correlated to the CPU utilization, since NFV is intended to use standard COTS CPUs to process high volumes of network traffic. Moreover, in a preliminary phase of our work, that we could not present due to the lack of space, we performed an analysis for the dimensionality reduction of the metrics space, which confirmed that metrics were highly correlated to CPU utilization.

Fig. 6 shows an example of correlation, in the presence of a performance anomaly, between the first two components of the Clearwater VNF chain, the *P-CSCF CPU %* (Bono) and *S-CSCF CPU %* (Sprout). The figure shows the time series for vCPU utilization of two instances of these network functions, and the Pearson correlation index (the yellow line) computed between these two, by using a sliding window. A workload surge is generated at minute 10. After minute 10, the Bono node starts dropping new connection attempts due to the overload, thus causing a reduced load on the subsequent Sprout node. When this happens, the correlation index drops close to -1, and our algorithm considers this as a symptom of fault (a performance anomaly). Analogue conditions occur in all the other failure scenarios that we consider in the experimental evaluation.

In summary, we consider the following sets of experiments:

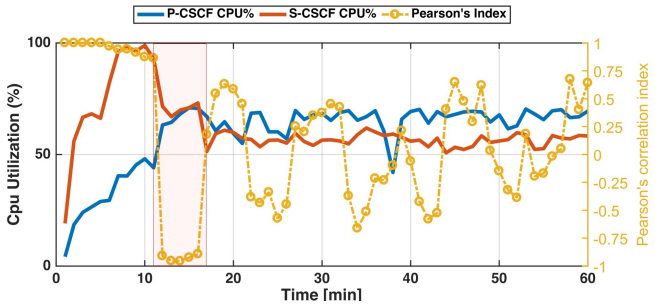


Fig. 6: Example of negative running correlation between P-CSCF and S-CSCF CPU utilization.

1) **Sudden workload surges**: the workload of the system rapidly grows, exceeding the engineered level of the IMS. In such a case, the available resources of the system may not suffice to manage the incoming load.
2) **Component failure**: the failure of a component of the system reduces available resources to satisfy all user requests, thus, causing an overload condition.
3) **Anomaly-free, long-running workload**: we consider long-running tests, with both constant and variable workloads, within the engineered level of the IMS and without any fault, to check whether any normal variation of the workload may trigger false positives.

In each scenario, we apply the fault correlation approach to the main service chain of the Clearwater IMS, including Bono, Sprout and Homestead, as shown in Fig. 7. More precisely, we apply the fault correlation algorithm to both the *Bono-Sprout* and *Sprout-Homestead* VNF pairs. We do not consider the *Sprout-Ralf* VNF pair since the external billing function (required by Ralf) is not included in Clearwater. Moreover, we do not consider homer, since it only provides a secondary functionality (a database service for the Telephony Application Server) that is not included in the IMS standard.

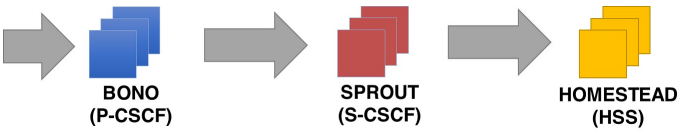


Fig. 7: VNF graph representing the chain of services' utilization.

A. Sudden workload surges

We study the impact of different types of workload surges on the QoS of the IMS and on the effectiveness of our detection approach. In each test with workload surges, we consider a different combination of the following three factors:

- The *number of subscribers*, as the user volume affects the severity of resource contention and of the saturation of the IMS capacity;
- The duration of the *ramp-up period*, that is, the time for the workload to increase from the engineered level to the selected level (the shorter is the ramp-up, the quicker is the workload surge and the on-set of the overload condition);
- The *call hold time*, which affects the the type and frequency of requests to the IMS, and consequently lead to different workload patterns.

TABLE II: Factors and levels for studying the impact of workload surges.

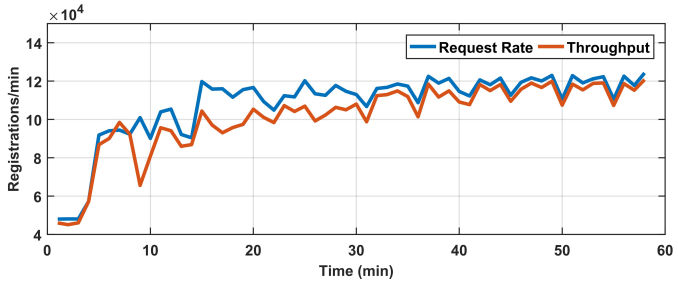
Factor	Level 1	Level 2	Level 3	Level 4
# subscribers	600k 20%-MTN	1M 100%-MTN	3.2M 640%-MTN	5.5M 1000%-MTN
Ramp-up	10 min	6 min	3 min	
Call hold time	2 min	1 min		

TABLE II reports in the detail the possible values that we selected for the three factors (four possible numbers of subscribers, three possible ramp-up periods, and two possible call hold times). In particular, the number of subscribers is expressed in relative terms with respect to the engineered level, that is, the users are $20, 100, 640, 1000\%$ more numerous than normal (denoted with *X%-MTN*). We adopted a full factorial design, with $4 \times 3 \times 2 = 24$ test configurations in total. In these experiments, workload surges are introduced starting at minute 10 since the beginning of the experiment; the time required to reach the peak of subscribers depends on the ramp-up period.

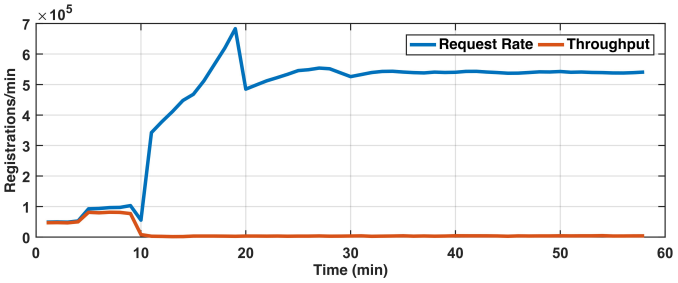
We found that covering boundary conditions (e.g., relatively high and relatively low volumes of users) highlights different behaviors of the IMS: in these extreme cases, either just few registrations and calls fail (but have still a noticeable effect on the perceived QoS), or almost all registrations and calls fail (as the resource competition is too strong to allow any request to get a sufficient amount). These differences also reflected on the performance of the detection algorithm. Instead, we found that the ramp-up period and the call hold time have a limited influence on the performance of detection; thus, for the sake of space, we limit the presentation of detailed results only to a specific ramp-up period (i.e. 10 minutes) and call hold time (i.e. 60 seconds).

We performed 5 repeated experiments for each test configuration ($r = 5$). A performance anomaly condition occurs when a non-negligible percentage ($\geq$10%) of user registrations and/or call setups are not successful, either because the request is not served within a time limit (10 seconds), or the IMS explicitly refuses the request and returns an error message to the client. The anomaly condition is considered detected if the algorithm raises an alarm within 60 seconds from the occurrence of registration and/or call failures.

Fig. 8 presents two examples of overload due to the increase of the number of subscribers. Fig. 8a shows, respectively, the number of *incoming* registration requests per minute, and the number of *completed* registrations per minute, when the workload is 20% larger than the nominal capacity. The difference between the two curves represents the amount of requests that could not be service due to resource contention



(a) Load 20% larger than the engineered level (20%-MTN)



(b) Load 1000% larger than the engineered level (1000%-MTN)

Fig. 8: Registration attempts per minute and registrations completed per minute

and saturation. Similarly, Fig. 8b shows the case where the number of subscribers increases by 1000%. In the former case (20%-MTN), the workload peak affected the quality of service for a small share of users, while the others were still serviced. Instead, in the latter case (1000%-MTN), not only the users in excess could not be serviced; but the workload surge caused a failure of the IMS software (which was unable to allocate resources, such as memory), thus leading to the unavailability of the IMS. Clearly, the larger the increase of the number of subscribers, the larger the number of registrations that are not correctly completed.

To evaluate the detection algorithm based on the running correlation, we consider several values of sample window size, i.e., we vary the number of samples from the time series that are correlated. Also, we evaluate detection performance when using different smoothing algorithms. Specifically, we consider to use (i) 10, (ii) 20 or (iii) 30 samples; and, as a smoothing algorithm, we test (i) Running Moving Median (RMM), (ii) Running Moving Average (RMA), and (iii) Exponential Moving Average (EMA).

The results for the detection algorithm under workload surges are reported in TABLE III. Clearly, the size of the sampling window and the smoothing function have a big impact on the detection performance. In all the considered overload conditions, the RMM and RMA smoothing functions perform better than EMA. This result is probably due to the fact that giving less importance to older samples in EMA, makes the algorithm more sensitive to noisy peeks revealing trends that are not representative. Moreover, the RMM algorithm appears more robust than RMA regarding the size of the sampling window due to the fact that the mean is more sensitive to outliers than the median. This results in lower detection latencies. In general, the average

TABLE III: Detection outcomes and latency under workload surges.

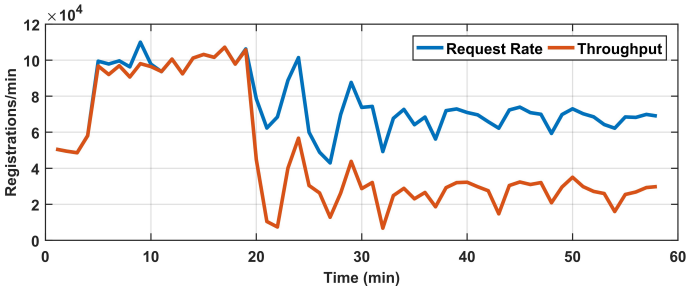
Overload Subs. (MTN)	Window	Smooth	Detection Outcome	Detection Latency (seconds)
20%	10	RMM	Detected (4/5)	29.0
	20		Detected (4/5)	45.6
	30		Not Det. (1/5)	28.0
	10	RMA	Unrel. Det. (2/5)	37.0
	20		Non Det. (1/5)	48.0
	30		Not Det. (0/5)	-
	10	EMA	Unrel. Det. (3/5)	46.0
	20		Non Det. (0/5)	-
	30		Not Det. (0/5)	-
100%	10	RMM	Detected (4/5)	29.0
	20		Detected (4/5)	44.0
	30		Unrel. Det. (2/5)	57.0
	10	RMA	Detected (4/5)	33.2
	20		Detected (4/5)	47.2
	30		Not Det. (0/5)	-
	10	EMA	Detected (4/5)	36.5
	20		Detected (4/5)	42.0
	30		Not Det. (0/5)	-
640%	10	RMM	Detected (5/5)	42.4
	20		Detected (5/5)	58.0
	30		Detected (5/5)	46.2
	10	RMA	Detected (5/5)	49.3
	20		Non Det. (1/5)	58.0
	30		Non Det. (0/5)	-
	10	EMA	Detected (5/5)	46.0
	20		Non Det. (0/5)	-
	30		Non Det. (0/5)	-
1000%	10	RMM	Detected (5/5)	29.4
	20		Detected (5/5)	49.4
	30		Detected (5/5)	46.2
	10	RMA	Detected (5/5)	38.0
	20		Detected (4/5)	57.0
	30		Non Det. (0/5)	-
	10	EMA	Detected (5/5)	36.0
	20		Non Det. (2/5)	57.0
	30		Non Det. (0/5)	-

detection latency varies between 30 and 60 seconds and increases when using bigger sampling windows. Collecting a sample every 2 seconds, a sampling window of 10 samples requires at least 20 seconds to be filled. Longer windows (e.g, 30 samples) result in worst coverage and longer detection latencies, especially with small overload conditions. For this reasons we recommend to use small sampling windows and the more robust RMM smoothing algorithm to achieve good results. With this configuration, we obtain 100% of detection coverage and an average detection latency of 32 seconds.

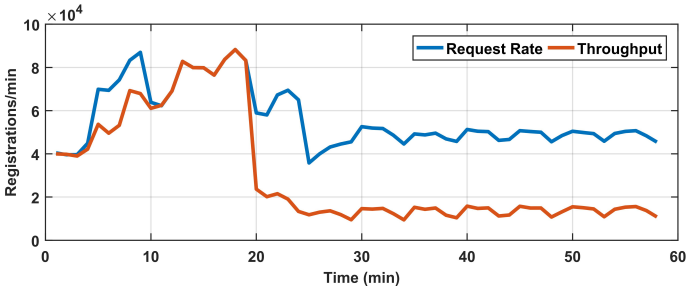
B. *Component failure*

We here analyze how component failure inside the NFV infrastructure (and thus, the variation of the capacity of the service chain) impacts on the QoS and on the effectiveness of the detection algorithm. We consider the following potential failure events:

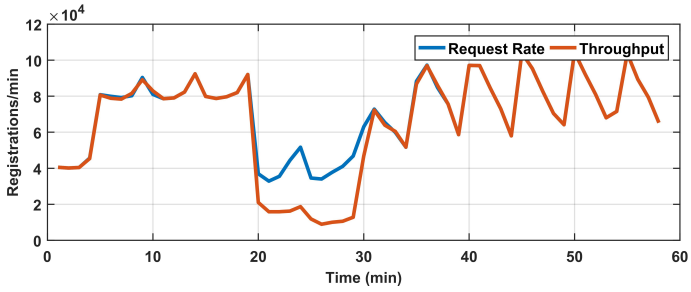
1) The failure of physical CPU cores of a machine that hosts VNFs, which is emulated by deliberately turning off a subset of CPU cores, thus forcing the hypervisor and the VNFs to run on fewer CPU cores and causing physical CPU contention.
2) The crash of VMs that run VNF software, which is emulated by deliberately terminating a VM, thus forcing



(a) the failure of 28 CPU cores over 32



(b) the failure of 8 S-CSCF nodes over 10



(c) the failover of 8 P-CSCF nodes over 10

Fig. 9: Registration attempts and registrations completed per minute, under component failures (injected at minute 20).

the IMS traffic to be load-balanced on the remaining replicas of the VNF.

3) The restart of VMs that triggers the migration and restart of IMS sessions. In the telecom domain, this phenomenon is often referred to as the *avalanche effect*, and is regarded as a problematic event due to the need to quickly restart a high number of connections in a limited time, and to force state migration in the case of stateful network functions [8].

TABLE IV reports in the detail the levels for experimenting with component failures. In total, we consider 4 test configurations, with $r = 5$ repetitions for each configuration.

TABLE IV: Factors and levels for studying the impact of failure events.

Factor	Level 1	Level 2	Level 3	Level 4
Failure	16 out of 32 pCPU failure	28 out of 32 pCPU failure	8 out of 10 S-CSCF failure	8 out of 10 P-CSCF failover

We apply the first type of failure on one of the three physical nodes that run the IMS; the second type of failure on S-CSCF services, by killing 8 VMs running the Clearwater Sprout service; and the third type of failure on P-CSCF services, by restarting 8 VMs running Bono, thus triggering the P-CSCF recovery. All the injections are performed 20 minutes after the start of the experiment.

Fig. 9 shows examples of the impact caused by the failures on the IMS. For three out of four failure events (the levels 2, 3, and 4 in TABLE IV), the injected faults indeed caused an overload of the IMS system, since many users were affected by failures due to unsuccessful registrations. Instead, in the remaining case (the level 1 in TABLE IV), the CPU failure were not enough to cause an overload condition, as the IMS client did not perceive any service degradation. The IMS components tolerate small a amount of physical CPU contention (e.g., the loss of 10% of CPU time, spent in involuntary wait state) with no effects on the throughput and the latency. Therefore, we decided to consider this experiment as anomaly-free (the remaining CPUs were able to tolerate the component failure and to serve the workload, so no anomaly should be detected for this case). This case is further analyzed in the next section.

To analyze the detection algorithm under component failures, for the sake of brevity, we focus the discussion on the case with a workload below the engineered capacity (i.e. 400k subscribers), sampling window size of 10 samples and a sampling period of 2s (i.e., the window length is equal to 20s), and we apply the Running Moving Median (RMM) as smoothing function; these choices for the window size and smoothing were the best ones according to the previous analysis with workload surges. Again, we have a performance anomaly when a noticeable amount of user registrations and/or call setups are not successful. The anomaly is considered detected if the algorithm raises an alarm within 60 seconds from the occurrence of registration and/or call failures.

Results from these injection experiments, reported in TABLE V, reveal a high detection coverage. The mean detection latency (i.e., 18 seconds) is approximately equal to the time required to fill the window (i.e., 20 seconds) with samples collected after the injected fault. The detection of this kind of issues is faster than the case with workload surges, because the injection of the faults caused quicker variations of the CPU utilization in a majority of the VMs, all at the same time. In the case of CPU contention (e.g., caused by the CPU failures), all the VMs deployed on the same injection target experienced involuntary waits due to the hypervisor scheduler. In the case of a reduced number of VMs (e.g., due to the crash

TABLE V: Results for detection based on running correlation for overload conditions due to failures.

Failures	Window	Smooth	Detection Outcome	Detection Latency (seconds)
physical CPU contention	10	RMM	Detected (4/5)	13.0
S-CSCF crash	10	RMM	Detected (5/5)	24.0
P-CSCF failover	10	RMM	Detected (5/5)	18.0

of S-CSCF instances) the algorithm required less feedback to reach the majority before raising an alarm, resulting in lower detection latency. In case of avalanche restarts (e.g., due to the failover of P-CSCF nodes) all the newly started instances immediately experienced overload and the algorithm got a quick feedback from a majority of the nodes.

C. Anomaly-free, long-running workload

To test for the occurrence of any false alarms under anomaly-free conditions, we carry out a set of experiments that are within the engineered capacity of the IMS system. We consider both the case of a stable workload at the engineered capacity, and two scenarios with variable workload (still within the limits of the engineered capacity). Finally, we consider the case of a failure event that reduces the available physical CPU cores (16-out-of-32) while still providing enough capacity for serving the workload (see also the discussion in the previous section). In these conditions, the algorithm should not detect any failure, thus any alarm is considered a false positive.

In the case of stable workload, we exercise the IMS with a constant number of subscribers (500k users). In the case of variable workload, we vary the number of subscribers over time. Periodically (every 20 minutes on average) the number of subscribers is reduced or increased, according to two patterns: in the first pattern (Fig. 10) the workload varies between three levels below the engineered capacity; in the second pattern (Fig. 11), the workload varies between five levels, up to the engineered capacity of the IMS.

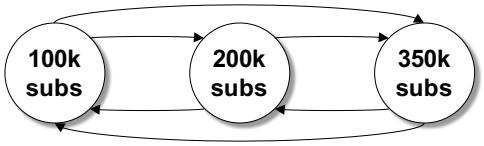


Fig. 10: Variable workload below the engineered capacity.

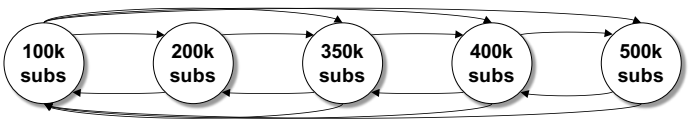


Fig. 11: Variable workload that saturates the engineered capacity.

In all these experiments, the detector provided an encouraging result: no false alarms were raised, for all test configurations. This result is motivated by the robust criteria that we adopt in the algorithm, as we require that (i) the CPU utilization should not simply vary on individual nodes, but the variations should be correlated at different pairs of VNFs; (ii) the correlation should show a high strength; (iii) a majority of the replicas in a VNF tier should be involved in the variation. Indeed, it is very unlikely that a false positive may occur, as confirmed by our anomaly-free experiments.

VI. Conclusion

We presented an approach to ease the adoption of anomaly detection systems in production NFV services. We showed that, by taking into account the VNF service chain topology, we can correlate performance metrics from different VNFs to infer the health of the service chain (e.g., service performance anomalies caused by the occurrence of bottlenecks and component failures). We proposed an algorithm to combine the correlations across multiple VNF replicas, to improve the accuracy of the detection.

We are planning to evaluate in a future work additional cases, such as to correlate non-adjacent VNFs in more complex topologies. In this work, we applied the algorithm to adjacent VNFs only, for two reasons: (1) in non-adjacent VNFs, we need to extend the basic approach to take into account the propagation delay between distant VNFs, but this would require tuning efforts by the users (instead, we are striving for a solution that is easy to deploy); (2) computing the correlation online between all the possible pairs of time series may be unpracticable in long VNF chains, due to the explosion of combinations of VNF pairs.

We validated the approach using an NFV-oriented IP Multimedia Subsystem (IMS). We selected a set of scenarios including overload, contention on physical resources and crash of the VNF instances, and studied the impact on the quality of service. We evaluated the detection coverage (i.e., the percentage of the scenarios where the detection outcome is detected) and the detection latency (i.e., the time between the occurrence of a failure and the detection of the anomaly). The experimental results show that the approach performs well across several conditions when using the Running Moving Median (RMM) smoothing function and a window of 10 samples with a sampling period of 2s. With these parameters, an anomalous condition is detected within half minute on average, with a very high detection coverage and no false positives.

The insensitivity of the algorithm against false positives, along with the freedom from thresholds that depend on the system (that would need to be calibrated with training samples, and to be tuned when the system is upgraded or reconfigured), are two key concerns that we took into account in the design of the algorithm, in order to make easier its adoption in production environments.

Acknowledgments

This work has been supported by the COSMIC project (DIETI department) and by Huawei Technologies Co., Ltd.

REFERENCES

[1] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC Editor, RFC 7665, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7665

[2] ETSI, NFVISG, "GS NFV-MAN 001 V1. 1.1 Network Function Virtualisation (NFV); Management and Orchestration," 2014.

[3] OpenStack Foundation Report, "Accelerating NFV Delivery with OpenStack," 2016. [Online]. Available: https://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf

[4] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping, "Network function virtualization: Challenges and directions for reliability assurance," in *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 37–42.

[5] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, and R. Natella, "Dependability evaluation and benchmarking of network function virtualization infrastructures," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–9.

[6] Wind River Systems Inc., "Achieving Carrier-Grade OpenStack for NFV," *White Paper*, 2015.

[7] VMware Inc., "Delivering High Availability in Carrier Grade NFV Infrastructures," *White Paper. VMware vCloud NFV*, 2016.

[8] ETSI, NFVISG, "ETSI GS NFV-REL 001 V1. 1.1: Network Functions Virtualisation(NFV); Resiliency Requirements," 2015.

[9] Doctor: Fault management and maintenance. [Online]. Available: http://artifacts.opnfv.org/doctor/docs/development_requirements/

[10] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, pp. 10:1–10:42, 2010. [Online]. Available: http://doi.acm.org/10.1145/1670679.1670680

[11] S. M. Bridges and R. B. Vaughn, "Fuzzy data mining and genetic algorithms applied to intrusion detection," in *Proceedings of 12th Annual Canadian Information Technology Security Symposium*, 2000, pp. 109–122.

[12] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, M. Embrechts *et al.*, "Network-based intrusion detection using neural networks," *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 12, no. 1, pp. 579–584, 2002.

[13] M. Ramadas, S. Ostermann, and B. Tjaden, "Detecting anomalous network traffic with self-organizing maps," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 36–54.

[14] M. Miyazawa, M. Hayashi, and R. Stadler, "vnmf: Distributed fault detection using clustering approach for network function virtualization," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 640–645.

[15] B. R. Granby, B. Askwith, and A. K. Marnerides, "SDN-PANDA: Software-Defined Network Platform for ANomaly Detection Applications," in *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*, Nov 2015, pp. 463–466.

[16] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, Oct 2016, pp. 196–206.

[17] Project Clearwater. [Online]. Available: http://www.projectclearwater.org/

[18] M. Boucadair, P. Morand, I. Borges, and M. Tomsu, "Enhancing the Serviceability and the Availability of IMS-Based Multimedia Services: Avoiding Core Service Failures," in *Next Generation Mobile Applications, Services, and Technologies, 2008 The Second International Conference on*, Sept 2008, pp. 444–449.

[19] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, Oct 2010.

[20] CloudWatch, Amazon. Creating amazon cloudwatch alarms. [Online]. Available: http://docs.aws.amazon.com/AmazonCloudWatchEvents/latest/APIReference/Welcome.html

[21] StackDriver, Google. Stackdriver monitoring. [Online]. Available: https://cloud.google.com/monitoring/

[22] Datadog. Introducing anomaly detection in datadog. [Online]. Available: https://www.datadoghq.com/blog/introducing-anomaly-detection-datadog/

[23] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 5, pp. 649–659, 2008.

[24] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of data mining in computer security*. Springer, 2002, pp. 77–101.

[25] J. Zhang and M. Zulkernine, "Anomaly based network intrusion detection with unsupervised outlier detection," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 5. IEEE, 2006, pp. 2388–2393.

[26] G. Jakobson and M. Weissman, "Alarm correlation," *IEEE Network*, vol. 7, no. 6, pp. 52–59, Nov 1993.

[27] A. T. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks," *Communications, IEEE Transactions on*, vol. 42, no. 234, pp. 523–533, Feb 1994.

[28] F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," in *Security and Privacy, 2002 IEEE Symposium on*, 2002, pp. 202–215.

[29] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A coding approach to event correlation," in *Integrated Network Management IV*. Springer, 1995, pp. 266–277.

[30] R. Taylor, "Interpretation of the correlation coefficient: a basic review," *Journal of diagnostic medical sonography*, vol. 6, no. 1, pp. 35–39, 1990.

[31] D. Rumsey, "How to interpret a correlation coefficient r," *Statistics For Dummies*, 2016.

[32] G. F. Reed, F. Lynn, and B. D. Meade, "Use of coefficient of variation in assessing variability of quantitative assays," *Clinical and diagnostic laboratory immunology*, vol. 9, no. 6, pp. 1235–1239, 2002.

[33] ETSI 3GPP, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; IP Multimedia Subsystem (IMS); Stage 2," ETSI, Tech. Rep., 2013.

[34] B. Fitzpatrick, "Distributed Caching with Memcached," *Linux J.*, vol. 2004, no. 124, pp. 5–, 2004.

[35] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[36] K. Jackson, *OpenStack Cloud Computing Cookbook*. Packt Publishing, 2012.