An AI-empowered framework for cross-layer softwarized infrastructure state assessment

Alessio Diamanti, José Manuel Sánchez Vílchez, Stefano Secci, *Senior, IEEE*

***Abstract*—Network softwarization technologies challenge legacy fault management systems. Coordination and dependency among different novel software components for orchestration, switching, virtual machine and container management creates novel monitoring points, besides novel sources of faults, bugs and vulnerabilities. To cope with the high heterogeneity and granularity of software components, we propose a modular network AI framework to detect anomalies, toward closed-loop automation. We design an AI-empowered anomaly detection framework able to assess the running state and the state deviations of a softwarized infrastructure, monitored through different features grouped depending on their layer and connect-compute stack component. Our framework learns the nominal working conditions of the infrastructure, with respect to which anomalies are detected, and characterized tracing back the layer and component root cause; it includes a network state assessment technique that leverages anomalies characterization through their most visible symptoms. We implement and validate the proposed framework through experimental tests on a containerized IP Multimedia Subsystem platform.**

***Index Terms*—Anomaly detection, Network function virtualization, network automation, Artificial Intelligence, Autoencoder, Autonomic and cognitive management.**

I. Introduction

Network automation is a research area targeting the deployment of auto-reconfiguration algorithms and protocols for operational communication networks. Research works in this area date back to a few decades ago. Only recently, network automation become a viable industrial direction thanks to the programmability capabilities offered by SDN (Software Defined Networking) and NFV (Networks Functions Virtualization) architectures. Indeed, SDN advocates for open interfaces to support network equipment programmability by moving network intelligence out of the switches device to support fine-grained flow configuration. NFV goes further beyond this advances, with the support of additional flexibility in network service deployment, namely the decoupling between network functions and hosting hardware.

In the past few decades, the networking community has addressed solutions to let distributed sets of agents self-organize themselves based on the learned network states, hence operating the necessary system reconfiguration. This was the focus of many research projects in the area of autonomic networks [2]. Also standardization activities addressed network automation requirements, as the ones related to autonomic signaling protocols among distributed decision-making agents [3].

A preliminary version of this article was presented at ITC 32, 2020 [1].

A. Diamanti and J.M. Sanchez are with Orange Labs, France. Email: firstname.lastname@orange.com. A. Diamanti and S. Secci are with CNAM, France. Email: firstname.lastname@cnam.fr

Recently, the relative maturity of network virtualization and softwarization systems has focused the industry specification efforts on the interfaces required for network automation, somehow meeting the expectation of former autonomic networking research, but now with an operational environment ready for their integration. On the other hand, network automation platforms recently emerged, notably the Open Network Automation Platform, chosen by many operators as a reference platform for network automation [4], [5]. Network softwarization eases the adoption of so-called 'cognitive network' approaches, e.g. referring to a closed-loop process consisting of 'sense' 'learn' 'decide' 'policy' and 'act' phases [6], [7]. The observations captured by the sensors (sense) help to build a model from the useful observations (learn), which is in turn used by a decision-making module to choose (decide) the actions to be taken based on possible moves and learned experience. Potential actions, i.e. strategies stored in the policy module (policy), are shortlisted by the planning module, so that, finally, the actuators execute (act) selected re-configurations [8].

In a softwarized environment, modern monitoring tools now allow retrieving thousands of metrics at different levels to sense a connect-compute platform composed of both computing and networking components. However, automatically extracting relevant features from such a massive amount of data to assess the system state is a challenge that we firstly addressed in [1], where we set the base of the so-called SYRROCA (System Radiography and Root Cause Analysis) framework. In this article, we further develop the SYRROCA framework, extending it to cover anomaly detection not only at the virtual layer, but the physical layer as well. We also propose a cross-layer system state characterization methodology. We show how this novel framework allows characterizing anomalies at any layer and their propagation across layers. We run tests in a containerized IP Multimedia Subsystem (IMS) architecture, that is the conventional network function cluster used for voice-over-IP traffic routing and processing in telecommunication networks. Simulated call distributions and used datasets are available at [9].

In Section II, we detail the background on anomaly detection in softwarized networks. We present our framework, SYRROCA, in Section III, and discuss experimental results in Section IV including analysis of state graphs and anomaly propagation across layers. Section V concludes the paper.

II. Background and Related Work

A common approach to qualify the running state of a networking system is to link its numerical measure to a notion of resilience. Commonly, resilience is the ability of a system to maintain an acceptable level of service in presence of impairments such as faults or attacks. Its modeling is challenging because it is a system-wide property that can depend on many factors. Authors in [10] deem that complexity in modeling network resilience comes from the varied nature of provided services, the numerous layers and related parameters, and the possible impairments threatening the network stability.

Resiliency measurement in softwarized networks, such as based on SDN and NFV technologies, is even more challenging due to the multi-layered and modular nature of softwarized networks; indeed, SDN separates control-plane from the forwarding plane functions [11], and NFV implies a layered architecture composed of the NFV Infrastructure (NFVI) layer, the Virtual Network Function (VNF) layer, and the network service layer [12] .

In a softwarized network infrastructure, an intensive use of NFV opens the way for new types of faults that either did not exist, or had a negligible impact on legacy hardware-based communication infrastructures. Novel software-originated vulnerabilities can be specific to hypervisor, VNF application, SDN controller and SDN south-bound and northbound protocol subsystems. For example authors in [13] produce a taxonomy of the possible security threats and attacks that can affect an NFV system. Similarly, in [14] authors propose another layer-specific threat taxonomy from which they build a set of recommendations on securing NFV based services. They also compare several security mechanisms that are applied in traditional scenarios and in NFV environments. About SDN systems, a detailed analysis in [15] reports on the amplitude of factors impacting software reliability of SDN controllers, not only related to software bugs but also the time required to integrate updates. Furthermore, in both SDN and NFV architectures, we observe a centralization of control, orchestration and configuration functions that makes them vectors of attacks and failures [16]. Detecting system-level faults due to overload, attacks and changing network and service conditions is therefore of higher importance in softwarized infrastructures than in legacy hardware-based systems.

The softwarized infrastructure environment favors an elastic usage of the compute, storage and networking resource stacks, such as for instance on-demand VNF instantiation, and related traffic routing adaptation, to react to faults at the physical substrate. The decision-making related to the provisioning and reconfiguration of already-provisioned services is therefore challenged by the dependencies of the delivered services from the underlying resources. As the state of the connect-compute stack continuously evolves with the changing demands, and insurgence of impairments, an adaptive real-time management of network resources should be based on a data-driven approach, where the overall stack is sensed to spot varying conditions. These factors originate a complex dynamic, and heterogeneous environment, with a high number of features, difficult to be taken into account in a single model. This motivates a data-driven framework rather than a model-based one, to scale with the heterogeneity of infrastructure components. In this direction, we propose a cross-layer machine learning approach to detect and characterize anomalies in a softwarized network, based on a potentially very large set of features related to connect-compute stack monitoring, with the goal to then trigger the appropriate orchestration actions to mitigate them as fast as possible.

A. Network resilience modeling and anomaly detection

There are some attempts to model network resilience. Authors in [17] use a measure of the network capability to get back to a normal state after a disruptive event. Authors in [18]–[20] model it with a multi-layer approach, with as application use-case mobile ad-hoc networks. Network resilience is assessed as a function of the deviation between the network states involved when anomalies occur. In [21] authors formalize the notion of probabilistic resilience as the largest number of component failures such that the network is still connected with a given probability.

When modeling resilience, the nominal and the anomalous states have to be recognized as such. When the system is coupled with a monitoring solution, the current practice is to intuitively establish a nominal baseline range for each group of monitored features, to reflect what is the acceptable behavior. Usually, the baseline limits are set by the network administrator through manual observation. However, this is not feasible in softwarized networks as the nominal behaviour may vary with time [22] and the number of features to observe may be huge [23]. A well-known solution consists of a set of association rules and frequent episode patterns to classify events as anomalous or normal ones.

Statistical approach show good performance when detecting unknowable anomalies [24]. For example, in [25] the chi-square (χ^2) test value is used as a distance measure to detect anomalies: when an observation chi-square value is greater than a fixed threshold, the observation is tagged as an anomaly. In [26] authors present a simple frequency-based approach to calculate the 'anomaly score'of a packet: the fewer times a given packet was seen, the higher is its anomaly score.

With the growth in availability and in the amount of monitoring data that characterize the latest and future networks, Machine Learning (ML) started to be applied to anomaly detection. Even though most of the ML AD techniques first profile normal instances and then identifies anomalies as instances that do not conform, authors in [27] propose the Isolation Forest (IF) method that explicitly isolates anomalies using binary trees. When dealing with high dimensional data, the most widely used approach is to project the data into a lower dimensionality sub-space where anomalies fall apart from the spontaneous clusters of data. Principal Component Analysis (PCA) and K-Means are the most used algorithms respectively for dimensionality reduction and clustering [28], [29]. For example, authors in [30] trained a k-mean algorithm on unlabeled flow records to cluster normal traffic. Then, the distance from clusters centroids is used to affect to samples an anomaly score. In [31] instead, first PCA is applied to the

KDD99 dataset for network IDS, and then a Support Vector Machine (SVM) is used to classify anomalous and nominal samples. However, clustering-based algorithms showed to be of low efficiency, mainly for a high false-positive rate [32]. Furthermore, PCA is recognized to fail in capturing temporal correlation [33] and in analyzing non-linear correlated metrics. Even though several non-linear approaches were proposed in the literature, it is broadly recognized that Deep Neural Networks (DNN) are very flexible and they can introduce a theoretically infinite level of non-linearities by using non-linear activation functions [34].

As in [24]–[33], we aim at defining a practical framework for network anomaly detection. Nonetheless, differently from previous works we aim at characterizing the detected anomalies working on a potentially very large set of observed features (i.e., thousands), coming from distinct network domains and layers (e.g., virtualized and physical layers) concurrently, while being able to trace back the root cause analysis of the anomaly. The scales involved justifies our approach to use deep neural networks to scale with the number and length of the involved time-series. We also recommend how the fine-grained cause analysis, by using our framework, can be employed for reconfiguring the network stack, in a closed-loop network automation setting.

B. Deep LSTM-based autoencoders for anomaly detection

An AutoEncoder (AE) is a multi-layer Neural Network (NN), composed of two blocks, an encoder and a decoder. The typical architecture of an AE is shown in Fig. 1.

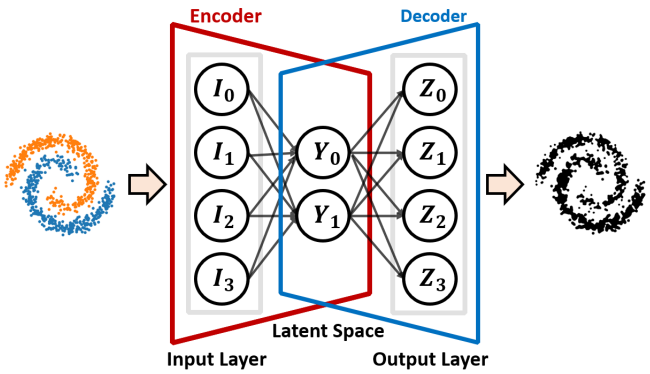


Fig. 1: Autoencoder architecture to characterize anomalies

The encoder reduces the F dimensions of the input X to a 'latent-space' composed of $s < F$ dimensions, while the decoder takes those s dimensions back to reconstruct the input. The autoencoder is trained to learn to reproduce the input vector X of F features $\in \mathbb{R}^F$ by optimization of:

$$\underset{f,g}{\text{minimize}} |I - g \circ f(I)| \tag{1}$$

where $f\colon I \in \mathbb{R}^F \mapsto Y \in \mathbb{R}^s$ with $s < F$ is the function representing the encoder, and $g\colon Y \in \mathbb{R}^s \mapsto Z \in \mathbb{R}^F$ is the function representing the decoder. During the learning phase, weights and biases are tuned to minimize the reconstruction error on I. Autoencoders are commonly used to detect anomalies, as for instance in [35]–[38]. In fact, when training autoencoders with anomaly-free data, anomalous samples projected into the latent space look significantly different from nominal samples; as a result, their reconstruction error is greater when compared to nominal samples. Autoencoders are considered as an auto-supervised neural network, as the target value used to train is the input itself, so no labels are required in the training phase. For this reason, we consider autoencoders particularly suitable for anomaly detection in softwarized network infrastructures: labeling anomalies for such an environment is a time-consuming and error-prone task, due to the great extent of faults and threats that can affect NFV environments [39], [40]. Hence data-driven approaches appear as an alternative approach to legacy, threshold-based ones attempting to model a network system using few alert-generating metrics.

Two important requirements shall be met when using autoencoders for anomaly detection: efficiency and scalability of the learning process. Stacking several encoder and decoder layers is commonly proposed in the literature as a solution to extract from data more general properties, rather than using a single layer [41]. Even though deep autoencoders are employed to compress high dimensional vectors, several studies showed that the result can struggle to efficiently learn with high-dimensional input vectors [42]–[44] Accordingly, SYRROCA is indeed designed to avoid the curse of dimensionality, while taking advantage of deep autoencoders: a set of parallel deep autoencoders is used to scale with very large and heterogeneous input vectors, using a dedicated autoencoder per group of resources. Those we use for the analyzed use-cases are CPU, memory, file system, and network resource groups.

The integration of autoencoders in a neural network, taking into consideration the temporal dynamics, is an additional design step. When analyzing multivariate time series, it is important that the autoencoder-based NN catches both the time dynamics of each variable, and the cross-dependencies among variables, to effectively grasp knowledge from the input data. In the literature, recurrent neural networks (RNN) are generally used to analyze time series [45]. Unlike Feed Forward (FF) neural networks, where each element is processed independently from the others, RNNs apply a recurrent relation at every time step to process a sequence in order to take into account past inputs, like a sort of memory. Nevertheless, many studies as [46] report that RNNs suffer from the vanish gradient problem, preventing long-term relations to be learned. To deal with long-term relations as well as shorter-term ones, authors in [47] propose the use of Long-Short-Term-Memory (LSTM) RNN that enforces constant error flow through the internal states of special units called 'memory cells'by employing multiplicative gates. Thanks to this complex internal structure, LSTMs can learn long term sequence correlations and model complex multivariate sequences [48].

In the context of network traffic and load forecasting, LSTMs demonstrated to outperform non-ML and other deep neural networks approaches [49]–[52]. In [53], authors propose a mechanism to scale 5G core resources by anticipating traffic load changes through LSTM and deep neural networks forecasting. They show that LSTM-based anomaly detection can be more accurate, thanks to its ability to store data

pattern without degradation over time. LSTM-based neural networks are used in [23] to adapt network baseline estimation to changes in cloud environments; the authors propose to create a network baseline through LSTM autoencoders that can be adapted when metrics trends change, showing that the proposed adaptation improves prediction accuracy by 22%.

C. Our contribution

We propose in this article an AI-empowered cross-layer framework to detect and characterize anomalies in softwarized networks, named SYRROCA (SYstem Radiography and ROot Cause Analysis) framework. We extend our previous work [1] with a refined modeling and anomaly analysis, as follows:

- by covering data collection over both physical layer and virtual layer metrics to allow spotting how anomalies in physical layer may propagate to virtual layer, and vice-versa;
- by proposing a technique to infer system state deviations through a state machine model, analyzing the detected anomalies looking at the subset of impacted resource groups and monitored features;
- by specifying how to leverage on the SYRROCA framework to recommend automated orchestration decisions based on the impact of the characterized anomalies.

III. SYRROCA (SYstem Radiography and ROot Cause Analysis) framework

In this section, we detail the SYRROCA framework architecture, presenting its core components. Our framework is designed to be modular, to support root cause analysis, at different levels of aggregation, so as to make it reusable for different use-case applications. We justify key design decision, in this and following section, on the reconstruction error used to spot anomalies, and on the autoencoder structure.

Fig. 2 shows a simplified diagram of the proposed framework, for an application example. The represented architecture is split by layer, and features groups are duplicated to analyze separately both physical and virtual layer metrics, in order to comprehensively analyze the whole stack.

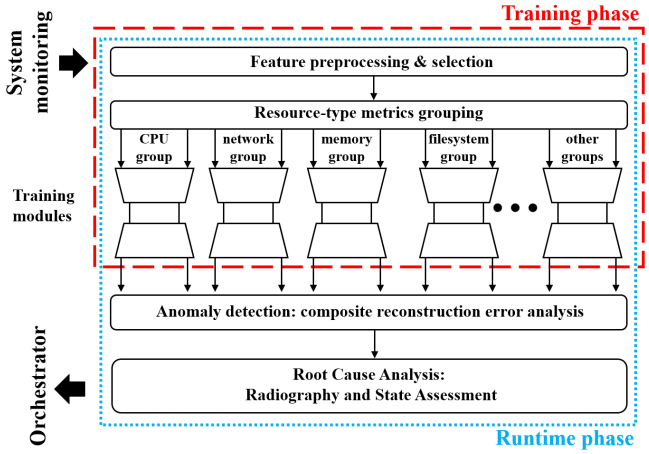


Fig. 2: Representation of the proposed SYRROCA framework

Table I summarizes the notation used throughout the article. [1]

TABLE I: Table of notations

N	Number of analyzed metrics
G	Set of resources groups
L	Set of layers
U	Set of considered computational units
$N^{l,g}$	Number of metrics referring to resources group g at layer l
τ	Dataset time length
$X^{l,g}(t) = [x^{l,g}(1), ..., x^{l,g}(\tau)]$	time-series of the j^{th} metric in input to the AE operating on resources group g's metrics
$\tilde{X}^{l,g}(t) = [\tilde{x}^{l,g}(1), ..., \tilde{x}^{l,g}(\tau)]$	time-series of the l^{th} metric produced by the AE operating on resources group g's metrics
$\mathcal{D}$	Set of detected deviations
d_t	deviation at time-step t
$T^{l,g}$	MSE threshold for resources group g
$SE_j^{l,g}(t) = [x_j^{l,g}(t) - \tilde{x}_j^{l,g}(t)]^2$	Feature-wise reconstruction squared error
$p^{l,g}(t)$	l^{th} feature MSE contribution rate
$F_t^{l,g}$	most deviated feature index set for deviation d_t of resources group g
$\Theta : F_t^{l,g} \mapsto \{(g_1, u_1), ..., (g_k, u_k)\}$	Function associating to $F_t^{l,g}$ the couples (g, u) corresponding to features in $F_t^{l,g}$

A. Data collection and pre-processing

In order to monitor a softwarized network platform to infer its state, we can collect heterogeneous data related to the various subsystems in stake. Metrics include categorical information in a text shape (alarms, logs) or in a numerical shape, encompassing metrics and KPIs (Key Performance Indicators). In this work, we focus our analysis on numerical metrics, leaving out other textual data that requires other types of pre-processing such as semantic text processing capabilities like in log analysis [54], [55].

The continuous collection of metrics results in a set of time-series, univariate or multivariate time-series, at different time intervals given by a scraping frequency. Depending on the metric source, time series may inherently encompass a trend; e.g., the number of sent/received packets is a cumulative counter with a monotonically increasing trend. On the contrary, other time series may arbitrarily describe increasing/decreasing metrics without an inherent trend; e.g., the frequency values taken by a computing processing unit or its temperature. On one hand, counters-based time series, if not properly pre-processed, are non-stationary, which tends to produce unreliable and spurious results leading to poor understanding and forecasting capabilities [56]; furthermore, counter

[1] Note that the terms *feature* and *metric* are used interchangeably throughout the article.

metrics evolution is rather characterized by its increments than the absolute cumulative value. On the other hand, gauges-like metrics do not exhibit an a-priori trend and are characterized by each instantaneous value. Thereby, to analyze a generic dataset containing both gauges and counters, we pre-process data as follows:

- we *de-trend* counters maintaining their increments, while keeping the raw values for gauges-like metrics.
- we *re-sample* metrics to a common fixed frequency taking into consideration how fast anomalies should be detected, and the training dataset length. Indeed, depending on the monitoring scraping frequency, the temporal resolution of each metric may be different, and in some cases so fine-grained that spurious outliers spikes could worsen data quality. Furthermore, high time-series resolution would make the training set huge, which proportionally increases training duration.
- we *re-scale* the input data into a uniform range, since metrics values may have widely different magnitudes. This is especially important for LSTMs, which are sensitive to the scale of the input data when the (default) $sigmoid$ or $tanh$ activation functions are used; it is in general true for whatever neural network trained with a gradient descend algorithm [57].

It is worth noting that two techniques are conventionally used to re-scale data: standardization and normalization. The former assumes that observations fit a Gaussian distribution (with a well behaved mean and standard deviation) and consists of shifting the distribution of each metric to have a mean of zero and a standard deviation of one (unit variance), while the latter consists of transforming the original metrics range so that all values fall within the $[0, 1]$ range. We detail next a way to choose a re-scaling technique for the generated dataset.

B. Training

During the training phase, SYRROCA learns the nominal conditions from the metrics characterizing a given layer and a resources group involved in a virtualized network service. Autoencoders are trained with a dataset composed of metrics collected during the normal working conditions, so that they can learn a compact representation of the nominal state. It is worth noting that both the quality and the extent of the data used for the training phase greatly affect the representation. Indeed, during the training phase, autoencoders are fed with anomaly-free samples during a sufficient period of time to learn the dynamics for each metrics group and layer to be characterized. SYRROCA uses dedicated deep autoencoders for each group of resources (CPU-related, memory-related, network-related, and file-disk-related) and layer (physical and virtual) in order to characterize anomalies occurring in a softwarized service in a fine-grained manner.

Consequently, for the addressed IMS use-case, we used a total of 8 deep autoencoders: 4 to analyze virtual layer metrics and 4 ones to handle physical layer metrics (e.g. a single autoencoder per resource group). It is worth noting that, for future possible applications of SYRROCA to other applications, additional and different metric sources and groups can be added with no restriction, for the sake of reusability.

As mentioned in the previous section, neural networks struggle to learn with high-dimensional inputs. Hence splitting the dataset per resource group streamlines learning. Additionally, it reduces training time. In fact, as described in [47] the LSTM cell epoch update complexity is $\mathcal{O}(W_i)$, where W_i is the number of cell's weights. For a standard implementation of an LSTM cell, $W_i = 4h_i \times (h_i + h_{i-1})$ where h_i is the number of hidden units and h_{i-1} is the dimension of the layer input, which is the previous layer's output. When using LSTMs in deep autoencoders, the number of hidden units depends on compression level $c_i \in \mathbb{R}$. Thus $W_i = 4Nc_i \times (Nc_i + Nc_{i-1})$ where N is the number of considered metrics. Subsequently, each cell update complexity is $\mathcal{O}(N^2)$. Assuming a deep autoencoder composed of M encoding and M decoding levels, the training complexity of the entire deep autoencoder is $\mathcal{O}(M \cdot N^2)$ for each training epoch. In contrast, if input metrics are split into n^g different groups (also referred to as resources groups in the remainder of the article), the deep autoencoder training complexity is reduced to $\mathcal{O}(M \cdot N^2/n^g)$ for each epoch.

C. Anomaly detection and characterization

In SYRROCA, an anomaly is characterized as a meaningful deviation from nominal conditions on a set of metrics. The framework is based on LSTM AEs wich allow to detecting anomalies when their reconstruction error exceeds a given fixed threshold. Indeed, when nominal conditions significantly deviate, the autoencoder fails in reconstructing those conditions and the reconstruction error increases.

Reconstruction error design: In general, depending on the problem, it is possible to choose among several types of reconstruction error. Let us discuss the most used ones at the state of the art of anomaly detection:

- *Mean Absolute Error (MAE)*. It is defined as the mean of the absolute error $\frac{1}{n}\sum ||x_t - \hat{x_t}||$, where x_t is the actual value and $\hat{x_t}$ is the forecasted one at time t. MAE is often used when outliers are not expected to be frequent and it is therefore not suitable for anomaly detection [58]. However, in our case, as we do not have negative values, the MAE is equal to the Mean Bias Error (MBE).
- *Mean Absolute Percentage Error (MAPE)*. It is defined as $MAPE = \frac{1}{n}\sum ||\frac{x_t - \hat{x_t}}{x_t}||$, and it is in general used to compare different models. However, when the data contains zeros MAPE gives indeterminate values [59], which is the case for our dataset.
- *Mean Squared Error (MSE)*, defined as $\frac{1}{n}\sum (x_t - \hat{x_t})^2$, and the *Root MSE (RMSE)*, defined as $\sqrt{MSE}$. Since the RMSE gives a relatively higher weight to large errors, it is generally preferred when larger errors are considered much worse than smaller. ones [60].

In SYRROCA, AE inputs are re-scaled to the $[0, 1]$ range, thus the training reconstruction error is always lower than 1. Thereby, as $\sqrt{x} > x \ \forall \text{x} \in [0, 1)$, the RMSE always produces greater error values than MSE. As the threshold is computed to be the 99.9% quantile of the reconstruction error distribution, the threshold also is always greater in the RMSE case. Accordingly, using the RMSE only produces a

re-scaled version of the same information we can compute with the MSE. Experimental tests we run to compare them in SYRROCA confirmed that we obtain with MSE or RMSE the identical anomaly detection outcome, with the same accuracy, recall, precision and F1 score for two sets of RMSE-based and MSE-based AEs.

Our choice for the reconstruction error is to use the MSE, mainly because (i) it is computationally less heavy than the RMSE, given that the former requires in addition to the MSE computation, also the computation of the squared root, while (ii) it produces identical results to the MSE. Moreover, as already mentioned, (iii) MAPE is not appropriate when data can have zero values, and (iv) MAE is not robust against outliers.

Anomaly characterization: An autoencoder is trained to reconstruct the nominal conditions with low error. However, when actual conditions significantly deviate, the autoencoder fails in reconstructing those conditions and the error increases.

For an AE trained with $N^{l,g}$ inputs and outputs the MSE is defined as:

$$MSE^{i,g}(t) = \frac{1}{N^{l,g}} \sum_{j=1}^{N^{l,g}} [\tilde{X}_j^{l,g}(t) - X_j^{l,g}(t)]^2 \qquad (2)$$

where $\tilde{X}_j^{l,g}(t) = [\tilde{x}_j^{l,g}(1), ..., \tilde{x}_j^{l,g}(T)] \in \mathbb{R}^T$ and $X_j^{l,g}(t) = [x_j^{l,g}(1), ..., x_j^{l,g}(T)] \in \mathbb{R}^\tau$ are respectively the output and the input vectors of the autoencoder working on group g, where τ is the size of the considered time-window.

Even though the MSE provides an efficient way to detect anomalies on a big set of metrics, because of the errors averaging it is not rich enough to characterize each anomaly. To compensate, we therefore design an approach to leverage the feature-wise reconstruction errors to determine the set of most deviated features, which in turn identifies the anomaly most severe symptoms. As described in the following, we can so use this information as an indication of the most impacted resources that can help in the system state assessment.

Let D and $d_t \in \mathcal{D}$ be the set of 'deviations' and a given deviation at time t, such that the MSE of a resources group exceeds a threshold value $T^{l,g}$. We use the 99.9% quantile as the threshold for each metrics group, i.e., 0.1% of the training samples are marked as raw anomalies by each autoencoder; note that, depending on the scales involved, this statistical threshold may be increased or decreased at the designer will.

To characterize anomalies using autoencoders we propose to compute the contribution of each feature to the MSE, computed as the feature-wise reconstruction squared error $SE_j^{l,g}(t) = [x_j^{l,g}(t) - \tilde{x}_j^{l,g}(t)]^2$ over the sum of the squared errors across all the features in a given resources group:

$$p^{l,g}(t) = \frac{SE_j^{l,g}(t)}{\sum_{j=1}^{N^{l,g}} SE_j^{l,g}(t)} \qquad (3)$$

The closer $p_j^{l,g}(t)$ gets to 1, the stronger is the contribution of feature j in group g to the MSE of that group ($MSE^{l,g}$), i.e. the feature j characterizes one of the most relevant symptoms of the detected anomaly.

Let us define $B^{l,g}(t) = \{b_1^{l,g}(t), b_2^{l,g}(t), ..., b_n^{l,g}(t)\}$ as the set of decreasingly ordered $p^{l,g}(t)$. Thus, we propose to take the first k values of $p^{l,g}(t) \in B$ that are reconstructed with the highest error and jointly contribute to at least the 90% of the MSE. This can expressed in mathematical terms as the set $b_1^{l,g}(t), ..., b_k^{l,g}(t) \in B$ with $k \leq N^{l,g}$ so that $\sum_{j=1}^{k} b_j^{l,g} \geq 0.9$ the features corresponding to these k values.

Consequently the set:

$$F_t^{l,g} = \{j : b_j^{l,g}(t) \in B^{l,g}(t), 1 \leq j \leq k\} \qquad (4)$$

contains the indices of the features that deviated the most from their nominal dynamics on layer l, group g. Therefore, the set $F_t^{l,g}$ pinpoints the most evident symptoms of the anomaly on the collected metrics. We only consider the feature that describes up to 90% of the MSE as our experiments demonstrated that this values brings a sufficient enough characterization of the anomalies while avoiding a too fine grained one[2].

D. Radiography characterization

Depending on the type of anomaly impacting the system, it may be contained in a single layer, or it may propagate to other layers. For instance, a process inside a container intensively using assigned CPU may be seen as an anomaly at the virtual/container level, while not affecting the quality of the delivered service. To account for this aspect, we adopt the *radiography* visualization proposed in [1], extended to handle both the physical and virtual layers, as a compact representation that shows anomalies impact across layers and on the delivered service.

Thanks to the usage of LSTM autoencoders, sudden yet recurrent variations of the state of a given resource, as the CPU for instance, will not be spotted as actual anomalies. Instead, non recurrent behaviors, such as uncommon spikes in CPU usages due to different events than those recurrent ones taken into account in the training period, will be appearing in the radiographies, and spotted by the anomaly detection logic.

We can have two types of radiographies:

- *Service cross-layer view:* going beyond virtual-layer specific approach, this view correlates the MSE of a given layer (virtual or physical) with a metrics characterizing the delivered service. Let $f(MSE^{l,g}, < service_metric >)$ be the bi-variate function joining the $MSE^{l,g}$ of a given group (virtual or physical layer specific) to values of the selected service metric (e.g., number of failed calls in the vIMS use-case).
- *Infrastructure cross-layer view:* this view correlates the MSE of the physical and virtual layers, for a given features group. Let $g(MSE^{l,phy}, MSE^{virt,g})$ be a bi-variate function describing how the MSE from the two layers of a given group are related to each other.

In practice, a radiography is a 2D density plot, computed through the Kernel Density Estimation (KDE), which is used to estimate the density of the f and g bi-variate functions.

[2]Note that k does not affect the anomaly detection, as a threshold set to the 99% quantile of the reconstruction error probability distribution is used to detect anomalies.

In particular, the density estimator is defined as $\hat{f}(x,H) = \frac{1}{n}\sum_{i=1}^{n} K_H(x - X_i)$, where $(X_1, ..., X_n)$ are the bi-variate function samples, K is the used Gaussian kernel and H is a bandwidth parameter, chosen with a well known rule of thumb [61].

A color/grey scale mapping density from high to low with colors from black to white, is used to visualize the computed KDE, obtaining the so-called radiography. Accordingly, the KDE-based radiography represents the density for each function for a chosen time window: the higher the density of a given point area is, the darker the color; darker regions represent most observed conditions. Density zones along the space bisector denote a direct propagation across variables/layers, while density zones near any of the two axes denote an impact affecting only one variable/layer hence with no propagation among the two. We later showcase radiographies for the vIMS use-case showing how to spot and characterize anomalies.

E. System State Inference

In cognitive network approaches [62]–[64] the sensing of the environment and learning the state is the preliminary step to inference. Optimal decision-making processes in orchestration concerns the capability to identify with low uncertainty the state of network resources to know where to apply those orchestration actions, so that they are effective enough to revert the deviated conditions towards normalcy.

The notion of system 'state' can be declined in different ways according to the addressed problem. In SYRROCA, we distinguish among three types of states:

- *Nominal State*: the system is in normal working condition when the MSE does not deviate for each group of resource metrics.
- *Training Degraded State*: the system is in an anomalous but known-in-advance working condition. In these cases, the MSE falls above the threshold for at least one metrics group in a given layer.
- *Test Degraded State*: the system is in an anomalous and unknown working condition during run-time test when the MSE falls above the threshold for at least one group of resource metrics in a given layer.

While the anomalous unknown states are meant to be detected in the testing phase, the nominal state and the known degraded states can be ex-ante characterized for a given use-case. Let a layer, whether physical or virtual, be fully characterized by two element sets:

- the set of resources groups, denoted as $G = \{g_1, ..., g_s\}$, where g_k denotes CPU, memory, network and file-system related metrics
- the set of computing units, denoted as $U = \{u_1, ..., u_v\}$, where each unit may represent a virtual machine, a container, or a server in the case of a physical layer.

Accordingly, we propose to characterize anomalous states in a given layer deriving the resource groups and the computational units that are the most impacted by the anomaly. Indeed, orchestration actions may not have a sufficient granularity to act on a single metric; rather, they can act on a specific resource and computational unit. Thus, the proposed characterization narrows down the choice of possible mitigation actions to those that act on the identified resources and computational units.

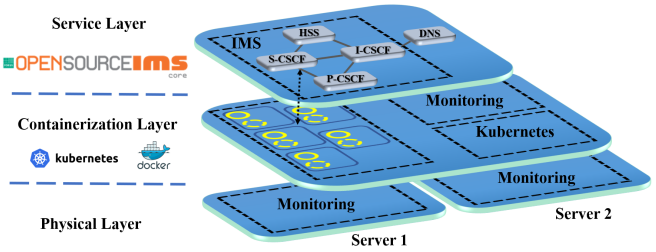


Fig. 3: Testbed settings

As defined before, the sets G and U are respectively the sets of resources groups and computational units impacted by a deviation. We propose then a function $\Theta : F_t^{l,g} \mapsto \{(g_1, u_1), ..., (g_k, u_k)\}$ to map the features indexes $F_t^{l,g}$ characterizing the deviation to the corresponding network resources and computation units. For example, the deviation characterized by the features $F_t^{l,g} = \{21 : container_cpu_load_average_10s\{dns\}, 33 : node_memory_MemFree_bytes\{server1\}\}$ is mapped through Θ to the impacted resource groups and computational units $\{(CPU, DNS), (memory, server1)\}$.

State deviations characterized by different features may be mapped onto the same couples (g,u).

Therefore, we define the system states as:

$$S_t = \begin{cases} Nominal & \text{if } MSE^{l,g}(t) < T^{l,g} \ \forall g \in G \\ \{(g_1, u_1), ..., (g_k, u_k)\} & \text{if } \exists\ g \in G \wedge \exists\ l \in L \mid \\ & MSE^{l,g}(t) \geq T^{l,g} \end{cases} \tag{5}$$

Non-nominal or degraded states are characterized through the most impacted computational unit(s) and the related resource(s) groups in physical and virtual layers. As SYRROCA can perform an analysis on a per resource group and per computing unit basis, state graphs can account for a comprehensive view of the entire softwarized platform.

IV. Experimental Results

We test the SYRROCA framework on a virtualized core IMS (IP Multimedia Subsystem) case study.

An IMS platform (IP multimedia subsystem) is composed of four main network functions: (i) the HSS (Home Subscriber Server), a database containing subscriber's profiles performing authentication and authorization; (ii) the P-CSCF (Proxy Call Session Control Function), the SIP proxy server, the first point of contact for the users; (iii) the S-CSCF (Serving-CSCF), it is the central node of the signaling plane for the SIP server and session controller functions; (iv) the I-CSCF (Interrogating-CSCF), the SIP function located at the edge of an administrative domain which assigns an S-CSCF to a user performing SIP registration.

We used the open source OpenIMSCore IMS [65] and modified its functions to be deployed as distinct containers. We exploited Kubernetes - an open-source container-orchestration

TABLE II: Number of features per layer and resource type

	CPU	Network	Memory	Disk	Total
Physical	370	290	40	260	960
Virtual	60	80	160	230	530

system that aims to provide a platform for automating deployment, scaling, and operations of application containers across clusters of hosts - to manage containers.

Fig. 3 depicts an example of our containerized vIMS composed of two physical servers, deployed at Cnam facilities.

Each physical server is equipped with an Intel(R) Xeon(R) CPU E5-2620 v4 @2.10GHz with 384 GB of RAM, connected to the same network through a 1-Gbps port physical switch. All the vIMS functions are deployed in dedicated Pods located in the server 1 (srv1), while Kubernetes core components are deployed in the server 2(srv2). Server 2 hosts the SIPp [66], a traffic simulator used to inject SIP and RTP traffic into the platform as two pods representing the caller and the callee. The whole platform is monitored through Prometheus node-exporters [67] for the physical level, while Pods and containers are monitored through a Kubernetes embedded CAdvisor [68] agent. Both exporters are compliant with Prometheus data model and architecture so that feature metrics can be exported through GET requests at a specific polling frequency.

A. Dataset

Metrics derived from server and container logs and traffic are directly collected every 5 s, forming time-series of 17280 values per feature and per day during 21 days. We downsampled the series to a 30 s frequency (2880 samples/day) to keep a good trade-off between time complexity and training error. Collected features are explicitly typed as counters or gauges, to ease their pre-processing.

Table II details the number of features per layer and resource group, showing the important magnitude in the number of features (hence our choice of using deep autoencoders to effectively compress high dimensional inputs vectors).

To simulate realistic traffic, we used real call traffic profiles extracted from a given LAC (Location Area Code) from the 3G Orange network. We injected 21 days (March 16 to April 05, 2020) of this traffic distribution onto the vIMS containerized platform. We set the average call duration to 3 min according to [69]. Moreover, the vIMS containerized platform is tailored to correctly process this traffic load. Fig. 4 reports mean call distribution for the three nominal simulated weeks as well as a LAC distribution used for testing purposes. The traffic profile was injected towards the vIMS through the SIPp tool by emulating continuous calls between the simulated users. Both RTP data traffic and SIP signaling traffic are transported over UDP. The employed data time-series and call distribution datasets are available at [9], and the algorithms and data extraction scripts are made available at [70].

It is worth highlighting that virtualized network services, differently than legacy transport network services, are tailored to a particular traffic, which can be isolated first, and then chained through dedicated functions thanks to the possibility to program virtual switches along the network path. This is the reason why we focus on a particular traffic and related virtualized network service architecture, as we believe this is a more realistic application than an application applied to an aggregate with undifferentiated traffic. In particular, we focus on VoIP traffic because it still today represents an important portion of ISPs revenues and as it requires QoS guarantees also in terms of availability.

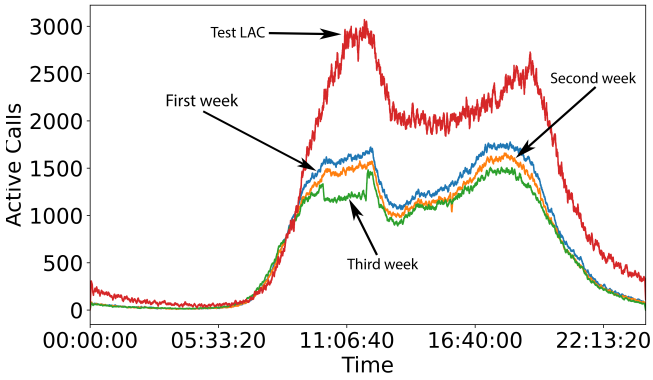


Fig. 4: VoIP call distributions emulated in the experiments

B. Features re-scaling

As discussed in Section III-A, it is of paramount importance for a neural network to evenly learn on all inputs. To decide which feature re-scaling approach to use we need a prior metrics distribution analysis. We use the chi-squared Pearson's cumulative test [71], [72] to characterize goodness of fit of different statistical distributions [3].

TABLE III: Best-fit distributions per resource group

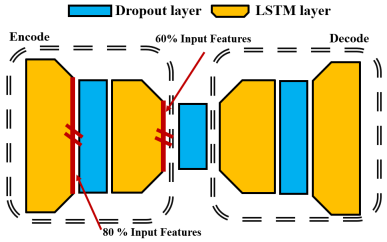
CPU	Network	Memory	Disk
Levy (15%)	PowerLaw (25%)	Alpha (14 %)	Betaprime (34 %)
Net (15 %)	Alpha (20 %)	Net (12 %)	Jhonsonsu (26 %)

Table III represents the distributions obtained for the four resource groups after the application of the χ^2 test. We can observe that none of the analyzed metrics fit a Gaussian distribution; according to what explained in Sect. III-A, we used therefore normalization.

C. LSTM-based deep autoeconder implementation

Autoencoder setting and hyper-parameter tuning are key steps needed to run an effective AI model. It is important that the chosen architecture is carefully adapted to the dataset characteristics. In this section we report how we calibrated the autoencoders architecture and hyper-parameters through several experiments performed on different combinations. We used the Keras python library with TensorFlow backend to easily compose deep autoencoders through pre-built layers.

[3]The chi-squared statistic, χ^2, is a normalized sum of squared deviations between observed and theoretical frequencies. The χ^2 bins data into n bins based on percentiles so that each bin contains approximately an equal number of values; for each fitted distribution the expected count of values in each bin is predicted from the distribution. The chi-squared value is the sum of the relative squared error for each bin.



Hyper-parameter	Value
Batch size	1Day
Epochs	2000
LSTM activation function	Elu
LSTM recurrent activation function	Sigmoid
Dropout rate	0,2

Fig. 5: SYRROCA deep autoencoder architecture

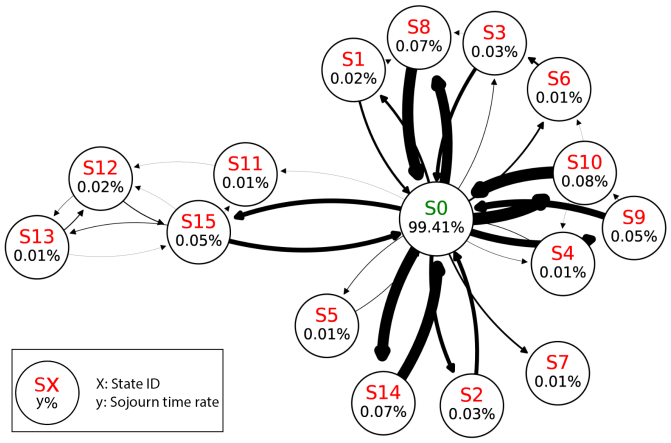


Fig. 6: State graph obtained during the training phase

Fig. 5 depicts the architecture used for each of the eight deep autoencoders; encoder and decoder are composed of two LSTM layers, and one dropout regularization layer to prevent over-fitting, which particularly affects DNNs [73]. According to the state of the art, over-fitting can be reduced by fitting all possible different neural networks architectures on the same dataset and then averaging the predictions from each model [74]. However, this is not feasible in practice. With dropout, during training, some layer outputs are randomly "dropped out"; therefore some layers look like one with a different number of nodes and links to the prior layer, mimicking different architectures.

It is worth noting that in the Keras implementation of LSTM neural networks the batch size hyper-parameter directly influences the amount of samples the internal state is built from. Indeed, by default Keras' LSTMs are stateless, which means that the internal state is reset after each training batch [75]. Consequently, as it was intended that the autoencoders to extract a compact representation of the nominal state across a whole day, the batch size was set equal to the size of a daily dataset. Furthermore, the training dataset represents three weeks of simulated calls, thus it is inherently characterized by a one-day period, therefore is out of scope to extend batch size to more than one day.

D. Training - known states characterization

The aim of the learning phase is to acquire a characterization of the nominal states as well as known degraded states with respect to the nominal conditions, so as to be able to detect future deviations from both nominal and known degraded states during the run-time usage for testing.

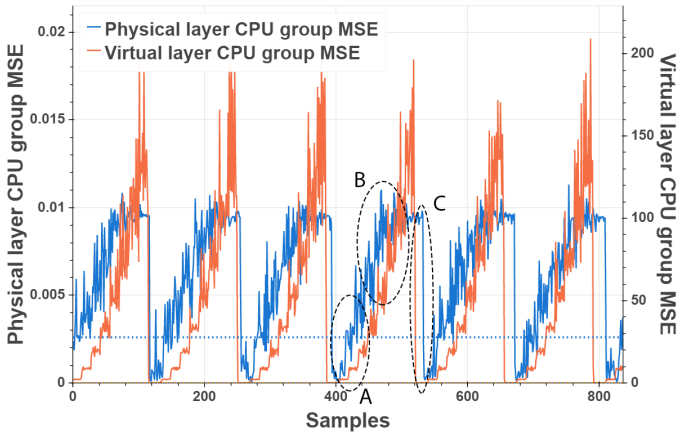
The nominal scenario is simulated through several SIP clients that first register to the vIMS core and then start a call. The vIMS containerized platform is tailored to correctly process the previously mentioned emulated VoIP traffic load.

Fig. 6 represents the states learned during the training phase; they are obtained as explained in Section III-E. States are connected within a directed graph [4], where an edge indicates any state transition occurred during the learning phase. The nominal state (or reference) is tagged as $S0$, while the degraded states are tagged as SX, where X is a unique identifier to unequivocally characterize each degraded state. Table IV summarizes the taxonomy of all degraded states detected across our tests. The thickness of the edge transition between states is proportional to the number of times it occurs. Under each state label it is quoted the percentage of time-steps the infrastructure sojourned in the given state during the simulation. In particular, given a time window T and the number t_{S_k} of time-steps the system state is S_k, the sojourn time is computed as $\frac{100 \cdot t_{S_k}}{T}$.

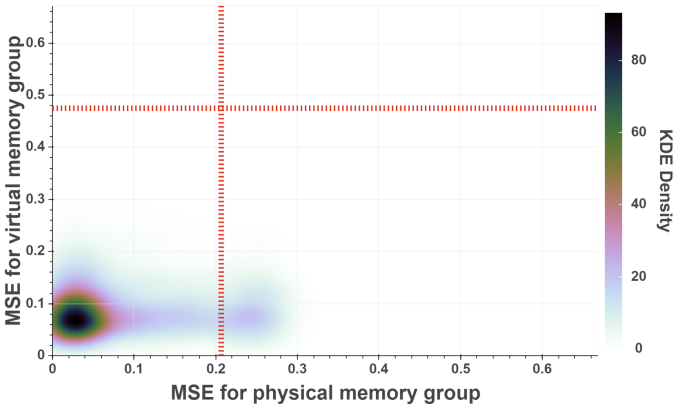
In order to report only on relevant transitions, we omit the states in which the system stays less than three time-steps (1 minute and 30 seconds). Some state transitions may not appear, such as outgoing edges from the state $S7$ to $S0$ in Fig. 6. Full states graphs are available at [9]. As each AE threshold is set to the 99.9 quantile of the training MSE distribution, each AE inherently detects 0.1% of the training samples as anomalies. Consequently, when each of the eighth AEs detects anomalies at different time-steps the maximum amount of detected anomalies is 8% over the training dataset.

As shown in Fig. 6, the most frequent transitions in the virtual layer are from/to the nominal state $S0$ to/from $S8$, that is a degraded file-system state related to anomalous DNS, HSS, ISCSCF and SCSCF features behavior. Similarly, the most frequent transitions in the physical layer are from/to the nominal state towards/from $S9$, $S10$, $S14$ and $S15$ which respectively represent CPU, network, memory and disk deviated feature states. Additionally, it is worth mentioning that in the physical layer states describing more than one type of degraded resource at the same time, i.e. $S3$, $S4$ and $S5$, only occur few times, while in the virtual layer none of the degraded states describe a simultaneous degradation for more than one resource type. This means that degraded states perceived in the training phase only represent occasional events that temporarily affect only one type of resource and that do not simultaneously impact the entire infrastructure (probably outliers). Nonetheless, a considerable amount of degraded states concern DNS and HSS features, meaning that AEs struggle in modeling associated metrics likely due to a barely predictable behavior. Finally, note that some states appear to describe an amplified deviation of other states, as it happens for $S13$ which adds a further deviation on (net,srv1) with respect to $S12$.

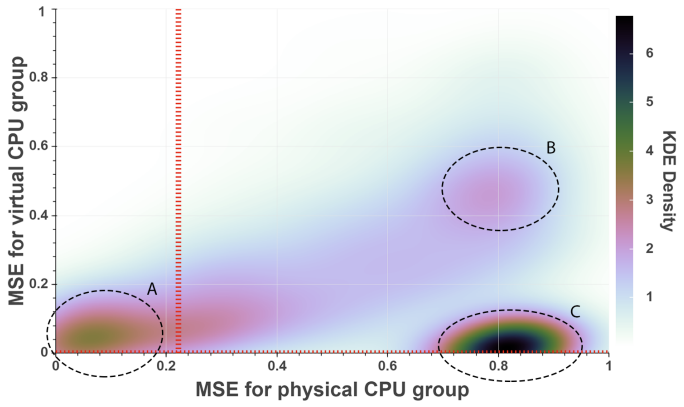
[4]the proposed state graph is only intended to be the foundation for a cognitive recovery control loop and it thus only serves as an indication on the type of remediation action the loop should perform.



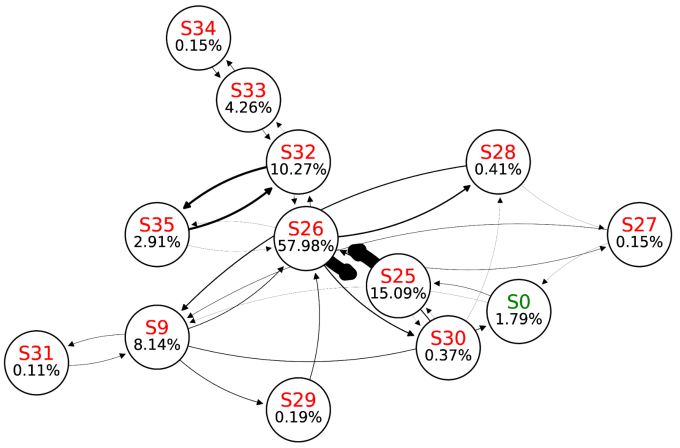
(a) autoencoder MSE for the CPU metrics groups in physical and virtual layers



(b) PCSCF memory stress radiography



(c) PCSCF CPU stress radiography



(d) State graph obtained during the CPU stress test case

Fig. 7: SYRROCA output visualizations for the CPU stress test case

E. Test on degraded conditions

We evaluate the SYRROCA ability to detect and characterize anomalies under three different anomaly scenarios. For each test case, SYRROCA provides a state graph and a set of different radiographies giving a per-layer, a per-resource group and an all-in one characterization of the anomaly impact. The obtained comprehensive view of the system compactly snapshots the system resiliency state, highlighting anomaly propagation across layers. Moreover, it allows spotting the computing units and resources on which remediation actions should act to bring the system back to the nominal state. Due to space constraints, for each test case we only show the most relevant representations (additional ones can be found in [9]).

1) CPU stress test: In the *first scenario* we tested how stressing the physical CPU in the PCSCF container is perceived by the autoencoders and how this stress propagates from the physical to the container layer. We injected a persistent physical CPU stress, which increases over time in evenly time distributed increments of 10% during one hour across 32 CPUs, starting from 10% up to 80% of single CPU capacity. Each stress cycle is repeated ten times.

As expected, the most frequently visited state is $S26$ characterized by a deviation on the PCSCF and physical CPU, as seen in Fig. 7d. Interestingly, SYRROCA detects other less frequent deviations on the CPU group of other containers as a consequence of that deviation, such as the states $S27$, $S28$ and $S29$. Indeed, those deviated states were not observed in the training phase, thus they do not refer to known occasional deviations. We can suppose then a non-perfect isolation of containers CPU lets the stress inside PCSCF occasionally get through other containers.

The third most frequent state is $S32$, which is the result of a stronger deviation from the state $S26$. This state is characterized by a deviation on the physical memory in addition to that of those deviated resources of state $S26$. This state is likely due to the high amount of memory used by the stressing test. Fig. 7b depicts the radiography showing how the deviations evolve through time across the 10 stress episodes. Red dotted horizontal and vertical lines represent the autoencoders thresholds for the memory group of virtual and physical layers, respectively. Accordingly, the bottom left rectangle represents the nominal region, where a dark high-density region. For instance, 7b shows that most of the time the system is in nominal conditions for both physical and virtual layers metrics. It can be observed that the memory deviation remains contained in the physical layer. This can be evidenced by the horizontal medium density region exceeding only the physical layer threshold but not exceeding the vertical threshold.

Fig. 7c instead depicts how the injected anomaly behaves for

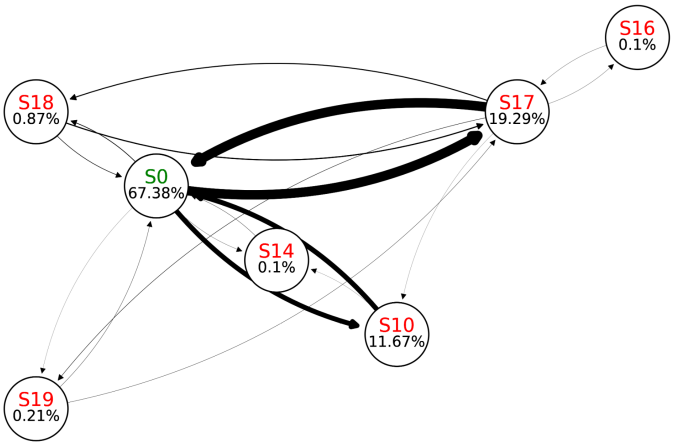


Fig. 8: State graph obtained during the packet loss test case

the CPU metrics group. While the region A represents a deviation affecting only the virtual layer (vertical axis threshold is not exceeded), regions B and C represent deviations perceived at both layers. Those deviations can be observed in Fig. 7a also, which depicts the stress pattern perceived through the MSE for the physical layer (in blue) and the virtual layer (in orange). At the beginning of each stress episode, the anomaly is first perceived only at the virtual layer (A). Then, as the stress increases, the deviation increases for both physical and virtual layer (B). Finally, when the stress episode terminates, the virtual layer deviation immediately decreases, while at the physical layer a residual deviation is still perceived (C). Note that across the 10 stress episodes time window, region C is the most frequent one: this is probably due to the effect of cumulative metrics which represent the physical CPU load across the last 5 and 10 minutes. Indeed, metrics representing the CPU load for the virtual layer are only instantaneous.

2) Packet-loss injection test: The *second scenario* consists in injecting packet loss to generate calls failures.SIPp allows simulating packet loss by simply blocking outgoing messages or discarding received messages. We alter the call distribution of March 16, 2020, blocking 50% of INVITE (SIP message) acknowledgments, causing at least 50% of calls to fail.

Looking at Fig. 8 system state graph, it is interesting to point out that almost all the degraded states only refer to network-related metrics, except $S16$ and $S14$, rarely visited. This means that the injected anomaly does not propagate across different resource types. In particular, the most recurrent deviated state is $S17$ which represents a degradation on the network metrics of the ICSCF and PCSCF containers, thus a virtual-layer only anomaly. Similarly, state $S10$ is only characterized by a deviation on the physical network metrics. As collected virtual layer metrics only belong from vIMS containers, a deviation like the one of state $S10$, which is only detected at physical layer, describes something related to Kubernetes pods or to any underlying Linux system-level process. Consequently, we can infer that deviated state $S10$ does not depend on the packet loss we imposed. Similarly, as no degraded state is characterized by metrics from both the physical and virtual layer, we can conclude that the detected anomalies never propagate through physical and virtual layers for any kind of resource.

TABLE IV: Couples (g,u) of resource group (g) and computing units (u) impacted in each degraded state

$S1$	{(cpu,dns),(cpu,icscf)}
$S2$	{(cpu,hss)}
$S3$	{(net,dns),(net,hss),(net,icscf)}
$S4$	{(net,dns),(net,hss),(net,icscf),(net,scscf)}
$S5$	{(net,pcscf),(net,scscf)}
$S6$	{(mem,dns),(mem,hss)}
$S7$	{(mem,hss),(mem,pcscf),(mem,pcscf),(mem,scscf)}
$S8$	{(disk,dns),(disk,hss),(disk,pcscf),(disk,scscf)}
$S9$	{(cpu,srv1)}
$S10$	{(net,srv1)}
$S11$	{(cpu,srv1),(net,srv1),(mem,srv1),(disk,srv1)}
$S12$	{(cpu,srv1),(disk,srv1)}
$S13$	{(cpu,srv1),(net,srv1),(disk,srv1)}
$S14$	{(mem,srv1)}
$S15$	{(disk,srv1)}
$S16$	{(cpu,scscf),(cpu,pcscf),(cpu,pcscf)}
$S17$	{(net,pcscf),(net)}
$S18$	{(net,pcscf),(net,pcscf),(net,scscf)}
$S19$	{(net,srv1),(net,pcscf),(net,pcscf)}
$S20$	{(net,pcscf)}
$S21$	{(cpu,scscf),(net,pcscf),(net,scscf)}
$S22$	{(net,scscf)}
$S23$	{(cpu,pcscf),(cpu,scscf),(net,pcscf),(net,scscf)}
$S24$	{(cpu,dns),(cpu,scscf),(net,pcscf),(net,pcscf)}
$S25$	{(cpu,pcscf)}
$S26$	{(cpu,pcscf),(cpu,srv1)}
$S27$	{(cpu,srv1),(cpu,dns),(cpu,hss),(cpu,icscf), (cpu,pcscf),(cpu,pcscf)}
$S28$	{(cpu,srv1),(cpu,hss),(cpu,pcscf)}
$S29$	{(cpu,srv1),(cpu,dns),(cpu,pcscf)}
$S30$	{(cpu,srv1),(cpu,pcscf),(net,srv1)}
$S31$	{(cpu,srv1),(net,srv1)}
$S32$	{(cpu,srv1),(cpu,pcscf),(mem,srv1)}
$S33$	{(cpu,srv1),(net,srv1)}
$S34$	{(cpu,srv1),(net,srv1),(mem,srv1)}
$S35$	{(cpu,pcscf),(mem,srv1)}

3) Call overload test: The *third scenario* consists in stressing the vIMS core with a call profile exceeding the resources available to the vIMS network functions. To do that, we chose to inject the call distribution of March 22, 2020 from a different LAC than the one used for training, serving more users (Fig. 4). Actually, even though in our deployment each pod can theoretically use as much memory as the physical server has (best-effort mode), the scripts used to launch IMS services impose a hard-coded memory limit. Nevertheless, we observed that although this script-level limit is not reached, it is possible to overload the vIMS core with a higher amount of traffic as in the selected test LAC. Unfortunately, the SIPp traffic simulation tool showed a limitation on the total number of the simultaneous emulated calls, therefore we could

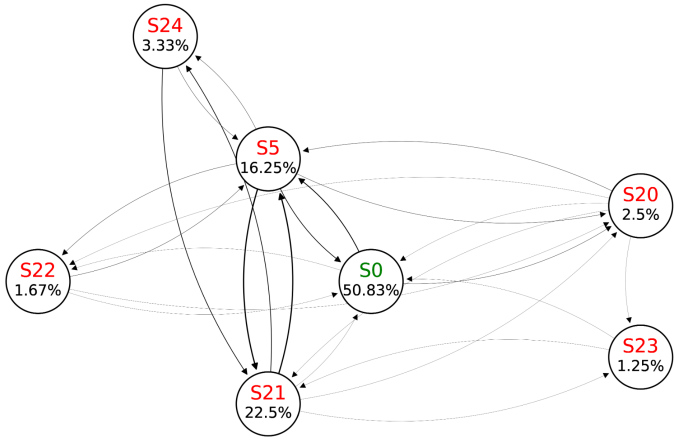


Fig. 9: State graph obtained during the overload test case

simulate only the first peak of the LAC depicted in Fig. 4, but in any case not invalidating the correctness of the test.

The state graph in Fig. 9 shows that the most frequent degraded state, $S21$, is characterized by deviations on SCSCF network and CPU-related metrics, and only network metrics of the PCSCF IMS virtual function. Similarly, the state $S5$, which is the second most frequent state, points out a deviation on the same resource/container, but on the SCSCF CPU. Furthermore, all the other most frequent states only point out deviations on CPU and/or networks metrics of different vIMS containers. Consequently, we can conclude that the simulated call overload generally produces a deviation on network-related metrics, and occasionally generates an additional load on the SCSCF CPU. Contrarily, no deviation is detected in any physical level resource, meaning that the anomaly does not propagate from the virtual to the physical layer.

Recurrent system states: Degraded states from the training phase, marked with labels from $S1$ to $S15$, refer to anomalous yet known-in-advance working conditions that can be found at the test phase as well. In our tests we observed that training degraded state $S5$ occurred also during call overload injection, state $S14$ during packet loss injection and state $S9$ in the CPU stress test case. As $S14$ and $S9$ are characterized by a deviation on server-wide metrics that are influenced by all the running processes, those states are quite likely to occur in a generic test case as autoencoders are not completely robust to outliers or state fluctuations. Furthermore, there could be several reasons for the CPU to deviate from the nominal learned state, which can make a degraded state characterized by only one deviated physical resource quite likely to appear. On the other hand, the state $S5$ characterizes a fine-grain deviation on PCSCF and SCSCF IMS components. The intuition could suggest that this state may be due to a sort of background noise, but this is not actually the case as $S5$ covers 16.25% of the total call overload test case time-stamps.

F. Detection performance assessment

As seen in section II-B, autoencoders proved to be effective in detecting and characterizing anomalies thanks to the inherent capability to encode and compress inputs. Likewise, our LSTM NN design demonstrated to be effective in learning long-term sequence correlations and to model complex multivariate sequences. In the following, we compare our SYRROCA LSTM-based autoencoder with both Isolation Forest (ISF) - a well-known unsupervised anomaly detection baseline - and RNN-based autoencoders - RNNs are also a good baseline, not storing long-term dependencies.

TABLE V: Comparison between RNN and LSTM autoencoders training epochs

	CPU		Network		Memory		Disk	
Model	**Virt**	**Phys**	**Virt**	**Phys**	**Virt**	**Phys**	**Virt**	**Phys**
RNN	140	249	160	233	492	359	103	359
LSTM	379	191	269	184	382	275	166	270

First, we compared a set of eight LSTM-based autoencoders with its equivalent RNN-based version: both autoencoder neuron architectures are identical, and the difference remains at unit-level (RNN unit and LSTM-unit). We trained both sets of autoencoders with the same dataset setting the maximum number of training epochs at 2000. Looking to use the framework in real environments, we stop the learning process if the MSE remains steady at least 10 epochs.

We can observe that LSTM autoencoders need fewer epochs than an RNN to reach the early stopping condition when analyzing physical layer metrics, as showed in Table V. On the contrary, results show that when working with less metrics such as in the virtual layer, RNNs reach the early stopping condition faster.

We compare the performance of LSTM-based autoencoders (AE) with RNN-based autoencoders and the ISF to detect deviations on the CPU group at the physical layer when analyzing the *CPU stress test* dataset. We used the baseline SKLEARN implementation of the ISF with all the parameters left to default values [76]. We labeled as 1 (positive samples) the anomalous samples while nominal ones are labeled with 0 (negative samples). The obtained results are summarized in Table VI. As we found the ISF not stable in SKLEARN, we repeated 500 times the classification, computing the mean for each performance metric. On the one hand, ISF improves the performance comparing to the RNN-based autoencoders, but it is less performing comparing to the LSTM-based autoencoders. In fact, a well-performing anomaly detection algorithm should be able to catch all the anomalies; in our experiments we labeled anomalies with 1, thus a properly working anomaly detection algorithm should have a Recall value close to 1. In our experiments, we found out that the LSTM-based autoencoders improve the recall by 30% comparing to ISF.

To account for each approach ability to not label as an anomaly (positive) a sample which is nominal (negative), we computed the F-2 score [77]. Based on the F-beta score that is the weighted harmonic average of precision and recall, the F-2 score lowers the importance of precision and doubles the importance of recall. As depicted in Table VI, RNN-based autoencoders give a low F2-score, while the LSTM-based autoencoders confirm an improvement of 30% over the ISF.

To understand why the RNN-based solution provides very poor performance, we analyze the RNN and LSTM autoencoders output reconstruction errors for the first four cycles of the CPU stress test case, depicted in Fig. 10. While the LSTM-

TABLE VI: Anomaly detection evaluation metrics

	Accuracy	Recall	Precision	F1-Score	F2-Score
ISF	0.472	0.515	0.802	0.626	0.554
RNN AE	0.166	0.053	0.714	0.098	0.065
LSTM AE	**0.826**	**0.866**	**0.927**	**0.895**	**0.877**

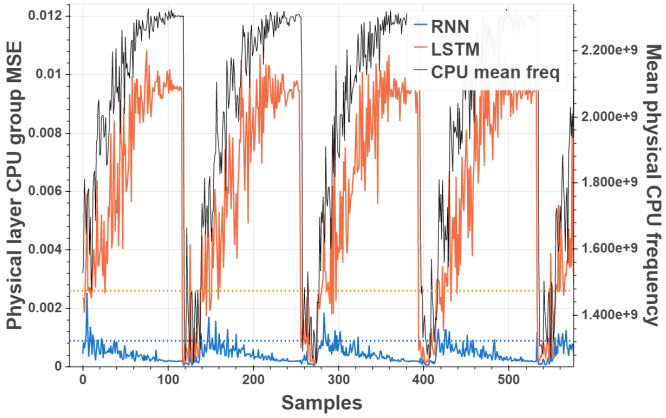


Fig. 10: RNN and LSTM based autoencoders reconstruction error compared to mean physical CPU frequency

based autoencoders produce an MSE that closely mimics the average CPU frequency, the RNN-based autoencoders can only detect the deviation at the very first start, but it fails in tracking the CPU stress during the rest of the cycle as its reconstruction error systematically tends to plummet below the threshold in contrast to LSTM-based approach. We advocate this behavior to the internal limited memory of standard RNN which limits the learning capabilities of the cells. In conclusion, even though the ISF provides good performance, the SYRROCA LSTM-based solution provides an improvement of at least 30% on the evaluation metrics. Therefore, the proposed LSTM-based solution is substantially more appropriate as it provides high performance for a large set of features, while it can tracks the anomaly evolution over time.

G. Leveraging SYRROCA for closed-loop automation

A cognitive loop allows to cover from the sensing of the network state to the act phase, that is the application of a set of actions (policies) to recover from a malfunctioning. Those policies are dynamically learned, with the aim to achieve an abstract goal dictated by the business or user technical requirements such as maintaining a certain Quality of Service (QoS) to fulfill a Service Level Agreement (SLA). The framework presented in this article covers the sense phase, by detecting and characterizing anomalies across physical and virtual layers. Nevertheless, this phase involves learning the nominal state conditions to later identify and characterize which network resources (resource group, computing unit) and to what extent are deviated. Thanks to this, further work is to build a cognitive loop to be able to compensate for the deviation by targeting those resources through the appropriate orchestration actions to bring the system back to the nominal state. For instance the degraded state $S26$ for the CPU stress test case, SYRROCA could recommend two orchestration actions related to scaling-out the PCSCF container or deploy that container on another server with more CPU instead of server srv1.

As per cognitive loop seminal ideas, the cognition should be implemented through an algorithm that 'improves its performance through experience gained over a period of time' [62]. Regardless of the implementation, not only the process needs to observe the result of the chosen action leading as a transition towards a new network state, but also to learn from that result. In the event of any degradation due to an incorrectly chosen orchestration action, the process should receive a negative feedback discouraging it to take the same action given the same conditions that led to that new state. Conversely, if the orchestration action correctly mitigates the detected anomaly, a positive feedback should reward the process. However, sometimes nominal system state must evolve to overcome an anomaly; e.g., when a physical server motherboard breaks down, the server will be replaced by a new one. To take into account the change, nominal system state representation should be periodically re-learned.

V. Conclusions

We presented an AI-empowered framework, named SYRROCA, to assess the running state of a softwarized network infrastructure. We proposed and experimentally evaluated a methodology to assess the running system state through the detection of deviations from a nominal state, learned from known history, for each type of resource and layer. We leveraged the MSE of a set of deep LSTM autoencoders to profile metrics on nominal working conditions and characterize anomalies with most deviated metrics. We described a state characterization methodology based on directed state graphs to visualize the evolution of the system across the nominal and the degraded states. The state assessment pinpoints the most deviated resource group(s) and computing unit(s) for a given anomaly. We also characterized the anomaly propagation across layers through the radiography visualizations. The obtained description of the system state and the detected anomalies allows a comprehensive assessment the resiliency state of the virtualized platform.

The state graph analysis methodology we proposed is meant to fuel a decision engine within a closed-loop automation system: as each degraded state is characterized by the set of most deviated metrics, remediation policies can be tailored to these deviations. Furthermore, we proposed a radiography visualization that, along with the state graph, allows SYRROCA to give a complete view of the impact of anomalies between each layer composing a virtualized network stack.

We tested SYRROCA for a vIMS use-case, with the goal to detect and characterize deviations and degradations on compute units and network resources in order to suggest the appropriate orchestration actions. Experimental results show that, as compared to alternative ML approaches, LSTM autoencoders are better able to model the nominal state of a virtualized platform by analyzing a multivariate time series composed of metrics collected by a monitoring system. We showed how to exploit the MSE of AEs to characterize system states that deviate from the nominal working conditions. In

comparison with a baseline RNN and an isolation forest (ISF) approaches, we found that RNN autoencoders improve both the recall and the F2-scores compared to the ISFs, which fail to extract any information on unlabeled data; moreover, LSTM further increases the autoencoder performance compared to RNN thanks to their ability to store long-term dependencies.

As future work, we aim at completing the closed-loop automation system working on a learning engine able to select connect-compute stack reconfiguration to recover from anomalies. We are also working to the distribution of the SYRROCA framework using federated learning for its online integration at edge network devices. An on-going work is the application of a recustomized SYRROCA framework to other use-cases, and in particular those of the H2020 AI@EDGE 5G-MEC connect-compute platform making use of OpenRAN technologies for the radio access components.

Acknowledgement

This work was partially supported by the ANR CANCAN project (ANR-18-CE25-0011) and the H2020 AI@EDGE project (grant nb. 101015922).

References

[1] A. Diamanti, J. M. S. Vilchez, and S. Secci, "Lstm-based radiography for anomaly detection in softwarized infrastructures," in *Proc. of the 32nd International Teletraffic Congress*, 2020, pp. 28–36.

[2] J. Rubio-Loyola *et al.*, "Scalable service deployment on software-defined networks," *IEEE Communications Magazine*, vol. 49, no. 12, pp. 84–93, 2011.

[3] M. Behringer *et al.*, "A reference model for autonomic networking," *IETF Internet Draft*, May 2018.

[4] A. a. Boubendir, "Network slice life-cycle management towards automation," in *Proc. of the IFIP/IEEE Symposium on Integrated Network and Service Management*, 2019, pp. 709–711.

[5] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, "5g e2e network slicing management with onap," in *Proc. of the 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops*, 2020, pp. 87–94.

[6] D. D. Clark *et al.*, "A knowledge plane for the internet," in *Proc. of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 3–10.

[7] Q. Mahmoud, *Cognitive networks: towards self-aware networks*. John Wiley & Sons, 2007.

[8] C. Fortuna and M. Mohorcic, "Trends in the development of communication networks: Cognitive networks," *Computer networks*, vol. 53, no. 9, pp. 1354–1376, 2009.

[9] A. Diamanti, J. M. Sanchez Vilchez, and S. Secci, "Syrroca github metrics repository," https://github.com/SYRROCA, 2021.

[10] P. C. et al., "A survey of resilience differentiation frameworks in communication networks," *IEEE Communications Surveys Tutorials*, vol. 9, no. 4, pp. 32–55, 2007.

[11] H. et al., "Software-defined networking (sdn): Layers and architecture terminology," in *RFC 7426*, 2015.

[12] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[13] T. Madi, H. A. Alameddine, M. Pourzandi, and A. Boukhtouta, "Nfv security survey in 5g networks: a three-dimensional threat taxonomy," *Computer Networks*, vol. 197, p. 108288, 2021.

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

[15] P. Vizarreta, K. Trivedi, B. Helvik, P. Heegaard, A. Blenk, W. Kellerer, and C. M. Machuca, "Assessing the maturity of sdn controllers with software reliability growth models," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1090–1104, 2018.

[16] S. Secci, A. Diamanti, J. Sanchez Vilchez, and et al., "Security and performance comparison of onos and odl controllers," Open Networking Foundation, Informational Report, pp. 4–7, September 2019.

[17] F. et al., "Resilience-based component importance measures for critical infrastructure network systems," *IEEE Transactions on Reliability*, vol. 65, no. 2, pp. 502–512, 2016.

[18] J. P. Sterbenz *et al.*, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[19] A. Jabbar, "A framework to quantify network resilience and survivability," Ph.D. dissertation, University of Kansas, 2010.

[20] D. Zhang and J. P. Sterbenz, "Measuring the resilience of mobile ad hoc networks with human walk patterns," in *2015 7th International Workshop on Reliable Networks Design and Modeling*. IEEE, 2015, pp. 161–168.

[21] W. Najjar and J.-L. Gaudiot, "Network resilience: a measure of network fault tolerance," *IEEE Transactions on Computers*, vol. 39, no. 2, pp. 174–181, 1990.

[22] M. Shaw, "Self-healing: softening precision to avoid brittleness: position paper for woss'02: workshop on self-healing systems," in *Proc. of the first workshop on Self-healing systems*, 2002, pp. 111–114.

[23] R. Mijumbi *et al.*, "Darn: Dynamic baselines for real-time network monitoring," in *Proc. of the 4th Conference on Network Softwarization and Workshops*, 2018, pp. 37–45.

[24] I. C. Paschalidis and Y. Chen, "Statistical anomaly detection with sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 7, no. 2, pp. 1–23, 2010.

[25] N. Ye and Q. Chen, "An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems," *Quality and reliability engineering international*, vol. 17, no. 2, pp. 105–112, 2001.

[26] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 105–136, 2002.

[27] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. of the eighth IEEE international conference on data mining*, 2008, pp. 413–422.

[28] W. Zhang, Q. Yang, and Y. Geng, "A survey of anomaly detection methods in networks," in *Proc. of the International Symposium on Computer Network and Multimedia Technology*, 2009, pp. 1–3.

[29] H. Ringberg *et al.*, "Sensitivity of pca for traffic anomaly detection," in *Proc. of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2007, pp. 109–120.

[30] R. Kumari, Sheetanshu, M. K. Singh, R. Jha, and N. Singh, "Anomaly detection in network traffic using k-mean clustering," in *Proc. of the 3rd International Conference on Recent Advances in Information Technology*, 2016, pp. 387–393.

[31] A. George and A. Vidyapeetham, "Anomaly detection based on machine learning: dimensionality reduction using pca and classification using svm," *International Journal of Computer Applications*, vol. 47, no. 21, pp. 5–8, 2012.

[32] I. Syarif, A. Prugel-Bennett, and G. Wills, "Unsupervised clustering approach for network anomaly detection," in *Proc. of the International conference on networked digital technologies*, 2012, pp. 135–145.

[33] D. Brauckhoff, K. Salamatian, and M. May, "Applying pca for traffic anomaly detection: Problems and solutions," in *Proc. of the International Conference on Computer Communications*, 2009, pp. 2866–2870.

[34] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[35] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[36] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proc. of the MLSDA 2nd workshop on machine learning for sensory data analysis*, 2014, pp. 4–11.

[37] D. Gong *et al.*, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1705–1714.

[38] M. A. Salahuddin, V. Pourahmadi, H. A. Alameddine, M. F. Bari, and R. Boutaba, "Chronos: Ddos attack detection using time-based autoencoder," *IEEE Transactions on Network and Service Management*, 2021.

[39] A. Gulenko *et al.*, "A system architecture for real-time anomaly detection in large-scale nfv systems," *Procedia Computer Science*, vol. 94, pp. 491–496, 2016.

[40] M. Kourtis *et al.*, "Statistical-based anomaly detection for nfv services," in *Proc. of the IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2016, pp. 161–166.

[41] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proc. of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 665–674.

[42] M. Verleysen and D. Francois, "The curse of dimensionality in data mining and time series prediction," in *Proc. of the International work-conference on artificial neural networks*. Springer, 2005, pp. 758–770.

[43] S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality," *Theory of computing*, vol. 8, no. 1, pp. 321–350, 2012.

[44] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE Trans. Inf. Theory*, vol. 14, pp. 55–63, 1968.

[45] R. J. Williams, "Learning representations by back-propagating errors," *Nature*, p. 533536, October 1986.

[46] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, pp. 1735–80, December 1997.

[48] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. o gf the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2015, pp. 89–94.

[49] Z. Cui *et al.*, "Deep bidirectional and unidirectional lstm recurrent neural network for network-wide traffic speed prediction," *arXiv preprint arXiv:1801.02143*, 2018.

[50] Z. Zhao *et al.*, "Lstm network: a deep learning approach for short-term traffic forecast," *IET Intelligent Transport Systems*, vol. 11, no. 2, pp. 68–75, 2017.

[51] X. Ma *et al.*, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 187–197, 2015.

[52] A. Dalgkitsis, M. Louta, and G. T. Karetsos, "Traffic forecasting in cellular networks using the lstm rnn," in *Proc. of PCI 2018*, 2018, pp. 28–33.

[53] I. Alawe *et al.*, "Improving traffic forecasting for 5g core network scalability: A machine learning approach," *IEEE Network*, vol. 32, no. 6, pp. 42–49, 2018.

[54] K. Fokianos and B. Kedem, "Regression theory for categorical time series," *Statistical science*, vol. 18, no. 3, pp. 357–376, 2003.

[55] G. M. Davis and K. B. Ensor, "Multivariate time-series analysis with categorical and continuous variables in an lstr model," *Journal of Time Series Analysis*, vol. 28, no. 6, pp. 867–885, 2007.

[56] W. W. Wei, *Time series analysis*. Pearson College Div, 2006.

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[58] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate research*, vol. 30, no. 1, pp. 79–82.

[59] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[60] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature," *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[61] N.-B. Heidenreich, A. Schindler, and S. Sperlich, "Bandwidth selection for kernel density estimation: a review of fully automatic selectors," *AStA Advances in Statistical Analysis*, vol. 97, no. 4, pp. 403–433, 2013.

[62] R. W. Thomas *et al.*, "Cognitive networks," pp. 17–41, 2007.

[63] C. Fortuna and M. Mohorcic, "Trends in the development of communication networks: Cognitive networks," *Computer Networks*, vol. 53, no. 9, pp. 1354–1376, 2009.

[64] L. Song, "Cognitive networks: standardizing the large scale wireless systems," in *Proc. of the 5th IEEE Consumer Communications and Networking Conference*, 2008, pp. 988–992.

[65] "Openimscore," http://openimscore.sourceforge.net/, accessed on 15/10/2021.

[66] "Sipp," http://sipp.sourceforge.net/, accessed on 15/10/2021.

[67] "Nodeexporter," https://github.com/prometheus/node_exporter, accessed on 15/10/2021.

[68] "Cadvisor," https://github.com/google/cadvisor, accessed on 15/10/2021.

[69] P. O. V. De Melo *et al.*, "Surprising patterns for the call duration distribution of mobile phone users," in *Proc. of the Machine Learning and Knowledge Discovery in Databases, European Conference*, 2010, pp. 20–24.

[70] "Syrroca github algorithms repository," https://github.com/Orange-OpenSource/Autoencoder-Based-Anomaly-Detection.

[71] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[72] R. L. Plackett, "Karl pearson and the chi-squared test," *International Statistical Review*, no. 1, pp. 59–72, 1983.

[73] G. E. Hinton *et al.*, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[74] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[75] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[76] "Sklearn isolation forest," https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html, accessed on 15/10/2021.

[77] H. a. Alaiz-Moreton, "Multiclass classification procedure for detecting attacks on mqtt-iot protocol," *Complexity*, vol. 1, 2019.

Alessio Diamanti is a PhD student since October 2018, working on virtualized network automation algorithms and platforms in collaboration with Orange Labs and Conservatoire National des Arts et Métiers (Cnam), Paris, France. After a master internship at LIP6 in 2017, he graduated in computer engineering from University of Bologna, Italy, and then worked as research engineer at LIP6 in 2018 before joining Orange and Cnam.

José Manuel Sánchez Vílchez (jose2.sanchez@orange.com) is a research engineer in Orange Labs. His main research interests are fault management and network resilience of programmable networks (SDN and NFV) trough novel machine learning approaches. He received the M.Sc. Degree in telecommunications engineering from Universidad Politécnica de Valencia in 2009 and the Ph.D Degree in Computer networks from UPMC and TélécomSud Paris in 2016.

Stefano Secci (stefano.secci@cnam.fr) is professor at Cnam, France, head of the Networks and IoT Systems team (https://roc.cnam.fr). He received a telecommunications engineering and a Ph.D. degree from Politecnico di Milano, and a dual Ph.D. degree from Telecom ParisTech. He was an associate professor at UPMC from 2010 to 2018, and previously worked for CNIT, Fastweb and Polytechnique Montral. His current interests cover network automation and cybersecurity.