This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

AI Anomaly Detection for Cloudified Mobile Core Architectures

Foivos Michelinakis[X†], Joan S. Pujol-Roig[X‡], Sara Malacarne[X*], Min Xie[X*], Thomas Dreibholz[X†], Sayantini Majumdar[§], Wint Yi Poe[§], Georgios Patounas[†], Carmen Guerrero[¶], Ahmed Elmokashfi[†], Vasileios Theodorou[‖]
[*]Telenor Research, Telenor, Norway
[¶]Universidad Carlos III de Madrid, Spain
[‖]Intracom Telecom, Greece
[†]Simula Metropolitan Centre for Digital Engineering, Norway
[‡]Samsung Electronics R&D Institute, United Kingdom
[§]Munich Research Center, Huawei Technologies, Germany

***Abstract*—IT systems monitoring is a crucial process for managing and orchestrating network resources, allowing network providers to rapidly detect and react to most impediment causing network degradation. However, the high growth in size and complexity of current operational networks (2022) demands new solutions to process huge amounts of data (including alarms) reliably and swiftly. Further, as the network becomes progressively more virtualized, the hosting of NFV on cloud environments adds a magnitude of possible bottlenecks outside the control of the service owners. In this paper, we propose two deep learning anomaly detection solutions that leverage service exposure and apply it to automate the detection of service degradation and root cause discovery in a cloudified mobile network that is orchestrated by ETSI OSM. A testbed is built to validate these AI models. The testbed collects monitoring data from the OSM monitoring module, which is then exposed to the external AI anomaly detection modules, tuned to identify the anomalies and the network services causing them. The deep learning solutions are tested using various artificially induced bottlenecks. The AI solutions are shown to correctly detect anomalies and identify the network components involved in the bottlenecks, with certain limitations in a particular type of bottlenecks. A discussion of the right monitoring tools to identify concrete bottlenecks is provided.**

***Index Terms*—Anomaly Detection, Autoencoders, Deep Learning, 5G, AI, Smart Networks, Mobile Networks**

I. Introduction

Identification and rectification of technical problems in mobile networks is a recurring issue faced by Telecom operators and vendors alike. Presently, with networks facing an architectural paradigm shift to 5G with diverse service demands, the need to automate the detection of bottlenecks in the network is even direr, as the failure to identify them on time may severely degrade Key Performance Indicators (KPIs) or violate Service Level Agreements (SLAs). Moreover, as manual detection and diagnostic procedures are error-prone and ineffective, automating the profiling and correction of anomalies is critical to handle the complexity of mobile network data. Early works such as [1], [2] have proposed effective anomaly detection and diagnostic frameworks to reduce human intervention in traditional 3G/4G cellular networks, but there is a need to extend this to the new multi-service, multi-tenant architecture.

[X]These authors contributed equally to this paper.

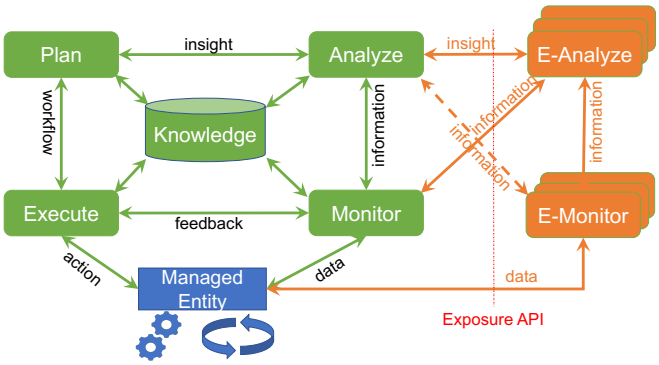


Figure 1: Exposed MAPE-K Model [3]

A group of companies focused on a specific niche, such as fish farms, eHealth, and smart cities are called vertical customers. To assure the SLAs that are delivered to vertical customers, it is critical to monitor, detect, identify, and resolve the issues causing service degradation. Although Service Assurance (SA) is inherited in network operations and management, it is also tightly related to the customer experience. Thus, Service Assurance should take into account the angles and insight of customers. In [3], we proposed an exposed Monitor-Analyze-Plan-Execute with Knowledge (MAPE-K) closed-loop model to automate Service Assurance, as depicted in Figure 1. In this model, network operators formulate an internal Closed Loop (CL) (green blocks) with their own monitoring, analyzing, planning, and executing capabilities. Then, based on the capability exposure model [4], network operators can expose certain monitoring services to external customers, who then combine with their external monitoring (E-Monitoring) tools to produce a more comprehensive view of the E2E services and provide for their own external analytics (E-Analyzing). The external insights can be fed back into the internal CL to enhance the internal Analyzing capabilities and produce more customer-driven insights. Both Mobile Network Operators (MNOs) and customers benefit from exposing the monitoring services: i) the customers get a view into network performance; ii) the MNOs get the customer's insight.

Furthermore, operators are experiencing significantly increasing pressure on assuring the service performance for multiple vertical customers concurrently. The two key ele-

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

ments of Service Assurance, monitoring and analyzing, are not adequately scalable in the 5G context with dynamic network slicing, as it is extremely complex and expensive to deploy a complete full-stack monitoring framework and support Service Assurance for all customers. Network slicing in 5G core networks relies on virtualization technologies including Network Function Virtualization (NFV) and Software-Defined Networking (SDN). The monitoring framework needs to capture the status and behaviour of different components and services, both physical and virtual. Moreover, it should also separate the monitoring data to differentiate between slices (network Quality-of-Service (QoS)), between vertical customers consuming the slicing services (vertical QoS), between the application services of an individual customer (application QoS), and even between the end-users running the application services (QoE monitoring). To make monitoring scalable, *differentiation* is desired. For example, monitoring granularity can be differentiated based on the Service Assurance task and corresponding Artificial Intelligence (AI) solutions (*e.g.*, low-granular monitoring for anomaly detection vs. higher-granular monitoring for root cause analysis), which often leads to different level of QoS guarantees.

Although the use of AI is a promising approach to deal with network-related tasks like network management and operations, resource allocation, service orchestration, and assurance in 5G networks , it remains an open question where and how these AI solutions should be integrated within the 5G networks. Vertical-specific AI (e.g., user-experience or QoE related, special data processing and security) is often supplied by 3rd parties or the verticals themselves. Thus, not directly related to network operations and not contained in the AI framework of 5G. Instead, such AI mechanisms can be attached to the network management systems as an enhancement to the network AI framework. These types of AI models collaborate with the 5G network in the following way: the 5G network exposes network data to the vertical AI, whereas the vertical AI provides customer/vertical insight back into the network to improve network operations and management.

In this work, we leverage the exposed capabilities of cloudified infrastructure to deploy external analytics tools to enhance the network Service Assurance focusing on the components of the core. Cloudification is a hallmark of 5G, through the use of NFV and SDN technologies. We use the testbed of [3], which is based on a cloudified mobile core architecture, and focus on how we can leverage AI for anomaly detection. To this end, we propose and evaluate two deep learning anomaly detection solutions in an expanded set of bottleneck scenarios and data sources proposed in [3]. We start with a primer on anomaly detection on network data. Then, we discuss how the MAPE-K model of service exposure can be adapted to a mobile network, which allows external customers to access the monitoring information of their Virtual Network Functions (VNFs). Next, we build a mobile network testbed, which uses a virtualized mobile core orchestrated by Open Source MANO (OSM) [5]. The testbed models an LTE network with an EPC core, which is typically the mobile core used in contemporary 5G architectures, i.e. 5G Non-Standalone (5G NSA). Since its components are packaged as VNFs, our solutions can be applied to any network that utilizes a cloudified mobile core, such as 5G Standalone (5G SA). OSM, as an open-source MANO platform, supports VNFs, physical network function (PNFs), and hybrid network functions (HNF) like Kubernetes network function (KNF). Its monitoring relies on both, the built-in OSM MON module and external monitoring tools. The obtained monitoring data is then exposed to an external 3rd party AI module, which adopts two deep learning anomaly detection solutions to ensure a big subset of Service Assurance requirements is respected. Namely, we identify network faults and isolate bottleneck locations. The output of the external AI could be used to send recommendations back to OSM, resulting in closed-loop automation. Finally, we test the correctness of our anomaly detection approach by inducing distinct artificial bottlenecks into the testbed. The results show how the exposed monitoring data can be used by the AI module to accurately detect anomalies, not only by identifying the faulty network components but also by listing the network metrics causing such anomalies. Thus, providing an architectural framework and external analytics for root cause anomaly detection systems.

The remainder of this paper is organized as follows: The state-of-the-art is presented in Section II. In Section III, we provide a detailed explanation of the AI anomaly detection solutions. Section IV presents our testbed and describes the distinct monitoring data sources and the type of bottlenecks that are examined to stress the testbed. Section V details the two AI anomaly detection implementations and gathers the results obtained using the distinct bottlenecks. A discussion on the strengths and limitations of our solution is provided in Section VI and future research directions are given in Section VII. Finally, conclusions are drawn in Section VIII.

II. Related Work and Background

A. Monitoring and bottleneck identification in 5G networks

Monitoring and identification of bottlenecks is a main theme in any system that serves users with an expectation of QoS. There is a significant body of work tackling monitoring and bottleneck characterization for individual components of 5G networks, such as traditional mobile networks (e.g. [6]–[8]) and cloud computing infrastructures (e.g. [9]–[14]). Further, there is work on bottleneck characterization and localization in virtualized mobile networks focusing solely on the core network, (e.g. [15]–[17]) or covering only single layers of the architecture (e.g. optical fault localization in the 5G vRAN [18]).

Equally important to detecting a bottleneck is the ability to do so in an efficient manner with advanced monitoring solutions, aiming to adapt and balance accuracy with monitoring overhead costs [19]–[21]. The authors of propose the monitoring components to be instantiated within each VNF, making their solution scalable [22]. [23] proposes a dynamic flow monitoring solution, which adapts the monitoring interval based on link utilization.

The aforementioned approaches only cover fragments of what has now become the 5G landscape which brings together mobile networks, virtualized infrastructure, and cloud computing. It is becoming clear that a holistic approach to bottleneck identification and localization is necessary. To

that end, the authors in [24] instrument a cloudified mobile network testbed end-to-end and on multiple layers, attempting to provide a comprehensive approach to bottleneck characterization. As a result, they provide a limited mapping of various data sources to the examined bottlenecks and a basic classification technique based on unsupervised methods. In [25], authors propose a general-purpose anomaly detection framework targeting micro-services architectures, such as the typical Service-oriented 5G architectures. They process service execution logs and spatial service query traces to identify outliers.

Finally, authors in [26] introduce a blockchain-based approach for the assurance of SLA between service providers and consumers in multi-domain network slicing environments. They employ a MAPE-K closed-loop architecture using cloud native operational data lakes for the continuous monitoring of the health and status of exchanged services, combined with smart contract-based intelligent orchestration mechanisms for the prediction of SLA violations (i.e. analysis) and their mitigation (i.e., planning and execution).

B. The role of AI in anomaly detection

The mounting pressure for networks to provide fast and reliable services has made IT system monitoring a crucial tool for network operators to manage and orchestrate the network infrastructure. The gathered data is continuously analyzed to detect any deviation from what is defined as normal or expected behavior so that network operators can react rapidly to minimize any network impact. Traditionally, the monitoring data analysis was performed by system monitoring experts who established threshold values based on handmade models of normal behavior for each evaluated event. If the measured value exceeds its associated threshold, the system is considered to not be performing as expected, and thus an exemption is raised so that the network administrator can take corresponding actions. Although the threshold-based method provides a simple and scalable solution to network operation management, threshold-based criteria is a static solution, i.e., thresholds cannot adapt to variable flow patterns. It also makes the management of dynamic traffic and network equipment difficult, as network traffic models must be elaborated beforehand and heuristics are used to determine the threshold values, which usually end up being sub-optimal.

The idea of distinguishing one set of normal measures from a set of outliers can be seen as a two-class classification problem with unbalanced data, i.e., one class (normal behavior) has more samples than the other (abnormal behavior). To process and monitor mobile network data, which is usually high dimensional, several traditional machine learning methods in the literature have already proven their efficacy. K-nearest neighbors, clustering techniques such as K-means, or support vector machines (SVMs) have been employed to classify data as abnormal [27]. However, typical drawbacks of SVMs are aligned with the usual disadvantages of classical techniques, i.e., inability to perform well for large and noisy datasets, not being too robust to outliers and performance sensitive to the choice of kernels. One way to cope with large amount of data and the high feature dimensionality of the samples that is deep neural networks.

Recent advances in AI models have shown that for complex high-dimensional, large and noisy datasets, autoencoders are a reliable choice for anomaly detection. An autoencoder is a generative unsupervised learning algorithm, usually implemented by a neural network, that consists of an encoder and a decoder network. The encoder accepts high-dimensional input features, producing a compressed representation of the input layer while the decoder reconstructs the compressed input to the output. The compressed layer encapsulates the latent features representative to the input dataset. The goal of the autoencoder is to minimize the reconstruction loss. It is able to detect anomalies in the data by analyzing the magnitude of the reconstruction loss, typically comparing the loss with a predefined threshold. To this end, [28] provides a solution for anomaly detection using variational autoencoders. In their work, an autoencoder is trained using the monitored data, and the reconstruction probability of error is used to identify the anomalies. Generative adversarial network (GAN) is another deep learning field that has found application for anomaly detection. In [29], the authors train a GAN architecture to generate samples following the distribution of the provided dataset. GANs are composed of one generative and one discriminative network, and once the training is finished, the generative network is discarded while the discriminator network is used to detect anomalies. Finally, the temporal correlation between samples has also been used to identify outliers in multivariate time series analysis. For example, [30] provides a solution that uses recurrent neural networks in combination with autoencoders to detect anomalies. Again, the reconstruction error of the autoencoders is used to flag a sequence of samples as abnormal.

More recently, these anomaly detection solutions have been extended into communications networks. In [31], an autoencoder is trained using Reference Signals Received Power (RSRP) and Reference Signals Received Quality (RSRQ) values to detect cell outages in Self Organizing Networks. The optimization objective, the reconstruction loss, is compared with an appropriate decision threshold to predict a potential cell outage. In [32], a novel anomaly detection solution for network data is provided, where authors proposed an autoencoder architecture leveraging two decoders, and use adversarial training to identify anomalies. Finally, the work of [33] provides a framework for anomaly detection aimed to the particularities of network data utilizing redefined threshold-based criteria.

In this work, a realistic 5G testbed is implemented, that performs Management and Orchestration (MANO) stack operations, using multi-layer measurements from across the network. These measurements are then exposed to external analytics functions, that provide advanced bottleneck detection methodologies based on AI to accurately identify performance bottlenecks. In particular, the proposed AI solutions leverage deep learning to detect anomalies and identify the root (network component) cause of them. More precisely, we provide two solutions that adapt to the particularities of network data and leverage a combination of the data exposed by the distinct monitoring tools: denoising autoencoders and convolutional-based autoencoders. The first approach leverages the generalization benefits of denoising autoencoders for

anomaly detection, while the second combines convolutional neural networks, to capture the temporal correlation between samples and to reduce training time, and autoencoders to detect anomalies. Once the faulty component is identified by the external AI module, the loop from monitoring to bottleneck detection can be closed by triggering countermeasures and bottleneck resolution. This work showcases a framework for a comprehensive fault resolution system that could be used in a real-world network deployment.

III. Our approach on A.I. anomaly detection in the context of NFV

A. Methodology

The methodology consists of the following parts/steps:

1) preprocessing (extract key messages from Section V.A)
2) application of anomaly detection (extract key messages from Section V.B)
3) thresholding (extract key messages from Section V.C)

Applying the exposed MAPE-K model to the OSM system leads to a joint CL with internal and external Monitoring as shown in Figure 2. The internal CL is composed of OSM components. OSM is an orchestration system capable of supporting auto-scaling in an CL way. Its *MON Collector* module collects infrastructure metrics from the Virtualized Infrastructure Manager (VIM) (e.g., OpenStack) like CPU and memory utilization and Virtual Machine (VM) behavior. The collected metrics are evaluated by the *MON Evaluator* module to produce alarms that trigger scaling. For example, based on a predefined scaling policy, an alarm is sent on the Kafka bus to notify other modules that the monitored Virtual Deployment Unit (VDU) metric(s) have crossed the predefined thresholds. Next, the *Policy Management* (POL) module, decides whether and what scaling action should be taken. The scaling action is executed by the *Life-Cycle Management* (LCM) module, e.g., instantiating a new VDU instance or terminating the underutilized VDU instance. In other words, the OSM components of *MON Collector*, *MON Evaluator*, *Policy Manager*, and *LCM/VCA* act as the four stages of the MAPE-K model of *Monitor*, *Analyze*, *Plan*, and *Execute*, respectively. In the sequel, we focus on the first two stages and leave the latter two as future work. Note that the scaling policies are defined in the VNF Description (VNFD), which can be regarded as the knowledge base where knowledge is shared by these MAPE stage modules. This internal MAPE-K CL is called *Internal-CL*.

Since OSM allows exposure of monitoring data to external parties, e.g., via Prometheus, an external CL can be created by extending the monitoring to external Monitoring modules, such as *Traffic Monitoring* and enhanced *Infrastructure Monitoring* (the blocks in orange of Figure 2). In addition, external AI modules are introduced as External Analyzer (E-Analyzer) to run customer-driven analytics by taking the input both from OSM and E-Monitor. The E-Analyzer results contain customer insight or interest, which are fed back into the OSM for further actions (e.g., revised VNFD or Day-2 reconfiguration commands). In this way, an external CL is built up on top of the internal CL as an enhancement. Specifically, the external CL reuses the OSM Policy Manager and LCM for *Plan* and *Execute*, respectively. E-Monitor is jointly realized by Traffic Monitoring and Infra Monitoring modules and E-Analyzer is offered by 3rd party suppliers and receives data from both OSM-internal MON Collector (via Prometheus) and E-Monitor.

Although both external and internal CL produce insights for further assurance actions, there is only one *Planning* stage, *i.e.*, the OSM Policy Manager. If the internal and external insights are contradictory, then it is up to the OSM Policy Manager that mediates and makes the final decision to be executed.

B. Anomaly detection in network data

This subsection formally introduces the problem the external analytics function aims to solve, and then describes the distinct AI solutions employed to solve it.

1) Problem formulation: The goal of the architecture described in this work is the collection of the network-related data obtained from the different monitoring sources of the testbed, which will be presented in detail in Section IV-B. These samples are then exposed to external analytics function using the close-loop automation architecture described in Section IV-A. The goal of the external analytics function leveraged in this work is to implement an AI solution that identifies whether a monitoring sample $\boldsymbol{x} \in \mathbb{R}^n$ belongs to a learned distribution $\mathcal{D}_{in}$. When a sample is found to belong to a learned distribution, it is referred to as "in-distribution" or normal sample, and otherwise it is said to be an anomaly or belonging to $\mathcal{D}_{out}$. Generally, it is difficult to know the distribution of outliers, $\mathcal{D}_{out}$, that one will encounter, thus, we assume that $\mathcal{D}_{out}$ is unknown. Given a binary variable $y \in \{0, 1\}$, the goal of the AI solution is to map each sample $\boldsymbol{x}^{(i)}$ to the label $y_i = f\left(\boldsymbol{x}^{(i)}\right)$, where $y = 1$ translates to anomaly and $y = 0$ to an in-distribution sample.

Furthermore, instead of assessing each sample individually, the anomaly detection solution can also exploit the temporal correlation between data samples. Thus, now instead of treating each sample individually, we can treat the problem as a multivariate time-series analysis. To model the dependence between each current sample and previous ones, we define the sequence of data points $\boldsymbol{w}_t$, as a sequence of points X_i for a time window of length K at given time t, as follows:

$$\boldsymbol{w}^t = \{\boldsymbol{x}^{t-K}, \boldsymbol{x}^{t-K+1}, \ldots, \boldsymbol{x}^t\}. \tag{1}$$

In a similar fashion as before, the goal of the AI solution is to map each $\boldsymbol{w}^t$ to a label y_t, i.e., learning the mapping function f that gives $y_t = f\left(\boldsymbol{w}^t\right)$.

2) Anomaly detection using autoencoders: An autoencoder is a type of neural network trained in an unsupervised way, that attempts to learn efficient data codings [34]. The autoencoder aims to learn a representation (latent distribution) from a set of data points, using dimensionality reduction. To this end, an autoencoder is composed of two parts:

- Encoder: This component aims to learn an encoding function f_θ to map the input data $\boldsymbol{x} \in \mathbb{R}^n$ to a latent representation of the input data $\boldsymbol{c} = f_\theta(\boldsymbol{x}) \in \mathbb{R}^m$, $m \leq n$.
- Decoder: takes as input the latent variable from the encoder $\boldsymbol{c}$, and aims to learn a reconstruction function

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

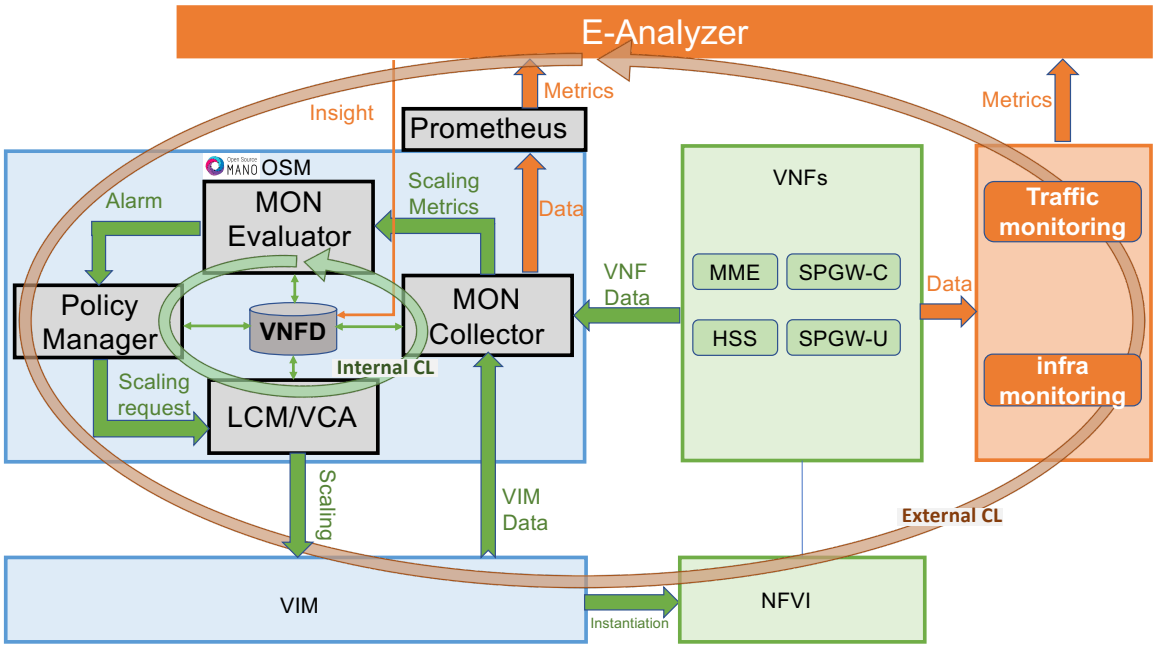


Figure 2: Architecture of joint internal and external CL based on OSM [3]

$g_{\theta'}$ to obtain a representation $\tilde{\boldsymbol{x}} = g_{\theta'}(\boldsymbol{c})$ as close as possible to the original input data $\boldsymbol{x}$ of the encoder.

In our case, both the encoder (θ) and decoder (θ') are parametrized using deep neural networks. The parameters of both networks can be optimized by minimizing the average reconstruction error in the training phase as follows:

$$\theta^*,\ \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\left(\boldsymbol{x}^{(i)} - g_{\theta'}\left(f_\theta\left(\boldsymbol{x}^{(i)}\right)\right)\right), \quad (2)$$

where $\mathcal{L}$ is a loss function such as means square error, L1-norm, cross-entropy, etc, and N is the batch size. As autoencoders are feedforward neural networks, stochastic gradient descent solutions can be used to obtain the network optimal parameters θ^* and θ'^*.

Finally, in order to label a sample $\boldsymbol{x}$ as anomaly, we use the reconstruction error and compare it to a predefined threshold, l_{th}, whose selection will be explained later. If the loss is below l_{th}, we assume that the sample belongs to the $\mathcal{D}_{in}$ distribution, otherwise we assume that it came from $\mathcal{D}_{out}$. The label y mapping is then given by the following pair-wise function:

$$y_i = \begin{cases} 0 \text{ if } \mathcal{L}\left(\boldsymbol{x}^{(i)} - g_{\theta'}\left(f_\theta\left(\boldsymbol{x}^{(i)}\right)\right)\right) \leq l_{th} \\ 1 \text{ otherwise.} \end{cases} \quad (3)$$

For some bottlenecks, it might be useful to have a further granularity of the loss, i.e., we want to know what features causes the overall loss function to increase. To this end, we define the per-feature loss as follows:

$$L_{x_j} = \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\left(x_j^{(i)} - g_{\theta'}\left(f_\theta\left(\boldsymbol{x}^{(i)}\right)\right)_j\right) \quad (4)$$

where notation $\boldsymbol{x}_j$ refers to feature j of vector $\boldsymbol{x}$.

3) Denoising Autoencoders (DAEs): The problem of optimizing Equation (2) as it is, i.e., without weight bounds, is that DNNs might have too much capacity so that they learn the task of copying data to inputs without extracting important information, this is also referred to as learning a null function. In order to not reduce our model capacity and restrict it to just a limited number of parameters, we can use regularization functions, or, in our case, Denoising Autoencoders (DAEs).

The idea behind DAEs is simple, the initial training data is partially corrupted by some form of noise added to the input vector in a stochastic manner. Then, the regular autoencoder model is trained to predict the original, uncorrupted data point as its output. The autoencoder is forced to undo this corruption rather than simply copying their input, avoiding thus, the overfitting problem. In our case, each data point is corrupted by white Gaussian noise as:

$$\boldsymbol{x'} = \boldsymbol{x} + z \sim \mathcal{N}\left(0, \sigma^2\right), \quad (5)$$

and then the parameters optimization becomes:

$$\theta^*,\ \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{N}\sum_{i=0}^{N} \mathcal{L}\left(\boldsymbol{x}^{(i)} - g_{\theta'}\left(f_\theta\left(\boldsymbol{x'}^{(i)}\right)\right)\right) \quad (6)$$

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

4) Convolutional Autoencoders (CAEs): Convolutional Neural Networks (CNNs) are a deep learning architecture inspired by the structure of the human visual cortex, [35], [36]. Two-dimensional CNNs (2D-CNNs) have gained major success in the field of Computer Vision, for their efficient way of performing feature learning, i.e. the reduction of images into a form which is easier to process, without losing features or degrading performance. Moreover, CNNs have been extensively adopted because of their scalability to large datasets since the number of parameters required is constant, contrasting with classical feed forward networks, where the number of parameters scale exponentially with the number of features. For 1D signals, that is, time series, 1D-CNNs have recently been proposed and immediately achieved the state-of-the-art performance levels in several applications,including anomaly/fault detection (see [37] for a survey). The advantage of adopting 1D-CNNs lies in the fact that the model training is highly efficient in terms of speed that allows real-time applications [38]–[41]. Similarly to 2D-CNNs, 1D-CNNs manage to capture important underlying patterns in the time series data by means of a 1-dimensional filter that slides along the temporal dimension. Throughout the paper we will simply refer to autoencoders equipped with one-dimensional convolutional layers as Convolutional Autoencoders (CAEs). For CAEs the loss function to be optimized is given by (Equation 2).

In summary, we provide two solutions that adapt to the particularities of network data and leverage a combination of the data exposed by the distinct monitoring tools: denoising autoencoders and convolutional-based autoencoders. The first approach leverages the generalization benefits of denoising autoencoders for anomaly detection, while the second combines convolutional neural networks, to capture the temporal correlation between samples and to reduce training time, and autoencoders to detect anomalies. Once the faulty component is identified by the external AI module, the loop from monitoring to bottleneck detection could be closed by triggering distinct predefined policies for bottleneck resolution.

IV. Experiments

A. Testbed

We apply the above concept to our testbed setup illustrated in Figure 3. The Enhanced Packet Core (EPC) of the testbed is realised as VNF by the SimulaMet OpenAirInterface (OAI) VNF [42]–[45]. It uses the EPC implementation of OpenAirInterface [46] implementing a cloudified mobile core architecture. The testbed uses LTE components, but our approach may be generalised to any network relying on a cloudified mobile core architecture, such as 5G SA. The VNF is instantiated as Network Service (NS) in OSM, the orchestration platform from ETSI. In our case, the VIM that manages and controls the hardware and virtualization resources in the Network Function Virtualization Infrastructure (NFVI) is OpenStack [47], [48]. It instantiates the VDUs as VMs. In this implementation, each VNF consists of exactly 1 VDU running in a dedicated VM. The EPC consists of 4 VDUs whose details are described in [43]:

1) The *Home Subscriber Server (HSS)* is the central database containing the information about users and their subscriptions. The HSS functionalities include mobility management, session establishment, user authentication and access authorisation. It provides its service to the MME via the S6a interface.
2) The *Mobility Management Entity (MME)* handles the procedures of attaching and detaching as well as service requests of *User Equipment (UE)* and Evolved Node Bs (eNodeBs). It communicates with eNodeBs over the S1-C interface, with SPGW-C over the S11 interface, and with HSS over the S6a interface.
3) The *Control Plane of the Serving and Packet Data Network Gateway (SPGW-C)* provides the control part of a Serving Gateway (SGW) and Packet Data Network Gateway (PGW). That is, OAI combines SGW and PGW, but uses *Control and User Plane Separation* (CUPS). The SPGW-C handles control requests from the MME via the S11 interface, and communication with the SPGW-U via the SXab interface.
4) The *User Plane of the Serving and Packet Data Network Gateway (SPGW-U)* handles the forwarding of user traffic between the *Public Data Network* (PDN) at the SGi interface (i.e. usually the public Internet) and the eNodeB over the S1-U interface. User traffic between eNodeB and SPGW-U is tunnelled via GPRS Tunnelling Protocol (GTP). The setup of user traffic tunnels is controlled by the SPGW-C over the SXab interface.

The UE is a regular PC, running Ubuntu 20.04 LTS "Focal Fossa", equipped with a Huawei E392 4G USB modem. NetPerfMeter [49, Section 6.3] [50], [51] is used for generating various traffic flows (TCP and/or UDP) between the UE and a peer server in the Internet running Ubuntu 18.04 LTS "Bionic Beaver". Another regular PC, running Ubuntu 18.04 LTS "Bionic Beaver", is used to emulate the eNodeB. It is running the eNodeB software from OAI, stable version 1.2.2. As Software-Defined Radio (SDR) board, an Ettus B210 connected via USB 3.1 is engaged. Various traffic flows, TCP and/or UDP, are generated between the UE PC and the server.

B. Monitoring Data Sources

We monitor the health of the system and traffic using a variety of tools placed at every component as shown in Figure 3. Table I provides a sample of the collected metrics. Our monitoring tools generate a few hundred metrics, but in the sequel, we deep dive only in the metrics that proved to be meaningful indicators of performance degradation and only in the components that are affected. It is possible a bottleneck introduced in one component to be invisible in its monitoring data, but manifest in the data of another component. For example, packet loss at a transmitter is detected more easily at the recipient of the traffic. We now present our data sources in hierarchical ordered based on their granularity and resource consumption:

a) OSM metrics: The highest level of data sources is OSM-provided information. We use the default OSM metrics collected by the *MON Collector*, which are stored in a Prometheus Time-Series DB with a granularity of 1 s [52]. The DB is exposed through a REST API to external consumers. The exact variables tracked by the *MON Collector* vary,

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

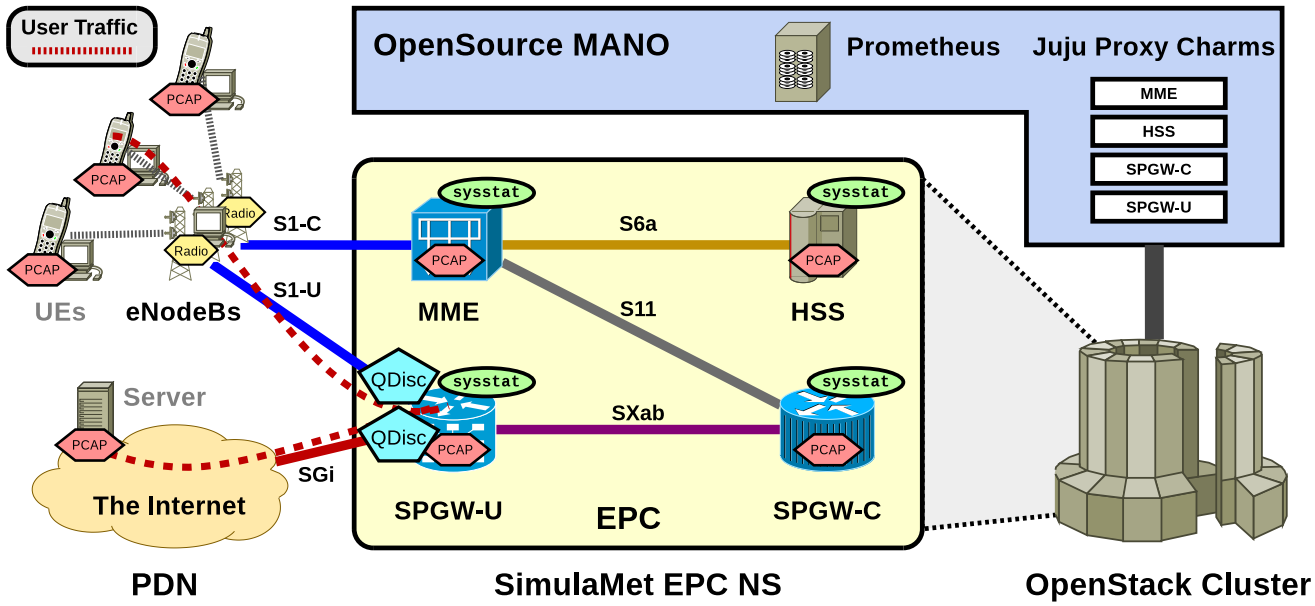


Figure 3: The Testbed Setup with EPC, OSM, NFVI, Packet Captures and QoS Rules

Table I: An example subset of the Monitored Metrics.

Type	Metric	Collection Place	Description
OSM Metric	Uptime	Every EPC VDU	Uptime of the VDU
OSM Metric	CPU Processes	Every EPC VDU	Number of processes
OSM Metric	CPU Load, current	Every EPC VDU	CPU load (%), current
OSM Metric	CPU Load, 1/5/15 min	Every EPC VDU	CPU load (%), over 1/5/15 min
OSM Metric	Memory Usage	Every EPC VDU	Memory usage (%)
OSM Metric	Swap Usage	Every EPC VDU	Swap usage (%)
OSM Metric	Disk Usage	Every EPC VDU	Disk usage for / partition (%)
OSM Metric	Network	Every EPC VDU	Bytes/packets in/out, per EPC interface
SysStat Metric	CPU	Every EPC VDU, eNodeB, UE, Server	Detailed CPU usage
SysStat Metric	Memory	Every EPC VDU, eNodeB, UE, Server	Detailed memory usage
SysStat Metric	Swap	Every EPC VDU, eNodeB, UE, Server	Detailed swap usage
SysStat Metric	Disk	Every EPC VDU, eNodeB, UE, Server	Detailed disk usage
Packet Capture Metric	Packets	Every EPC VDU, eNodeB, UE, Server	Packet capture for each interface
HiPerConTracer Metric	Ping	UE	RTT between UE and server

depending on the VDU and among others, including average CPU load, aggregated across all cores, over the past 1, 5, and 15 minutes and incoming and outgoing packets and bytes per interface, as exposed to OSM (e.g. S11). The data comes from two sources: NFVI metrics via OpenStack telemetry and customized performance metrics via Juju charms in OSM, set up in Kubernetes and running in a container of the corresponding machine hosting OSM.

b) SysStat *metrics:* Each VDU runs inside its own VM. We use the system performance monitoring tool SysStat [53] in each VM of the EPC core to collect information regarding its health. SysStat's sampling frequency is 1 s, as for OSM, but it can collect much more detailed statistics. For example, we are able to monitor CPU load per core and the load metrics are further divided by the process type, such as user and high-priority system processes and the network traffic statistics are provided on a per-interface level. In consequence, this leads to higher resource requirements for processing and data storage.

c) Packet capture (traffic monitoring): We perform packet capture through `tcpdump` on all the interfaces of the EPC, the client UE, and the server the UE communicates with. This is to a large extent the most demanding data source we have available, enabling us to monitor traffic at the packet level. We are able to track packets as they enter and exit a VDU, which can be used to estimate the packet processing delay and also check if a packet exiting a VDU arrives at the corresponding VDU of at the other side of the link, which can be used to estimate packet loss. In our experiments, we capture all the traffic, but in practice, the traffic would be filtered. Therefore, only a small selection of flows would have all their traffic monitored. In future iterations of the testbed we consider using a specialized tool, such as PPTmon [54], designed to monitor VNF traffic at a much lower cost compared to packet captures.

Finally, we can track end-to-end Application Layer statistics such as end-to-end throughput and delay through two tools running at the connection endpoints: the UEs and the server. 1) Our traffic generation tool NetPerfMeter generates data-

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

B0	Baseline, with no bottleneck. Data used to train ML models
B1	VM resource overload on SPGW-U
B2	0.1% packet loss in traffic crossing SPGW-U
	10% packet loss in traffic crossing SPGW-U
B3	2 Mbit/s, 250 ms delay in traffic crossing SPGW-U
B4	2 Mbit/s bandwidth limit in traffic crossing SPGW-U
	10 Mbit/s bandwidth limit in traffic crossing SPGW-U

Table II: Summary of the experimental scenarios.

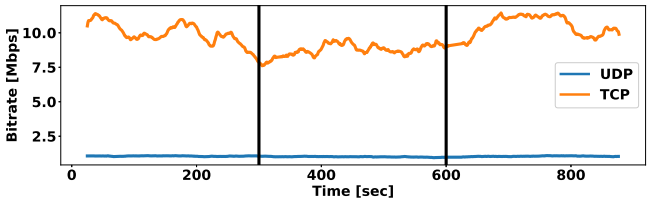


Figure 4: Bitrate of incoming traffic to the UE during a B1 scenario. The vertical lines indicate the time period when the bottleneck was active.

plane TCP/UDP traffic flows emanating from the peer server (on the Internet) towards the UEs. NETPERFMETER can tag the generated traffic, both TCP and UDP. Using the tags and packet captures at each hop, various metrics – like individual link/host delay – can be calculated. 2) We also perform "Ping" measurements through HIPERCONTRACER [55]: every 250 ms, the UE sends an ICMP Echo Request to the server, which is answered by an ICMP Echo Reply. It collects the Round-Trip Time (RTT), i.e. the time span between request and reply, to measure the network latency experienced by the user.

C. Bottlenecks

Before training the Machine Learning (ML) models, we need to have a clear picture of what normal behaviour is. To this end, we define a baseline scenario B0, meant to reflect the operation of the network under "normal" conditions. We generate several TCP and UDP flows between the client UE and its peer server on the Internet, which start and stop randomly. We do not introduce any artificial load or adjust network characteristics. Instead, the traffic flows in the setup without any disturbance for 2 h. The dataset of this scenario is the input of the models discussed in the sequel.

Next, we examine several different bottlenecks that may manifest in a mobile network, which are grouped into experiment scenarios. Experiment scenarios B1, B2, and B3 consist of three phases bottlenecks, each phase lasting 300 seconds. In the first phase ($t \in [0, 300]$ sec), the testbed runs in its normal configuration. During the subsequent phase ($t \in [300, 600]$sec) we artificially introduce the respective bottleneck. The normal configuration is restored in the final phase ($t \in [600, 900]$ sec) such that the network is allowed to recover. Table II summarizes all the experimental scenarios.

Unless otherwise noted, each experiment initiates two "user" flows between the UE and the server in the Internet:

1) TCP flow: A saturated flow of payload data from server to UE (i.e. a download scenario).
2) UDP flow: A bidirectional flow of payload data between server and UE, with an average of 1 Mbit/s of payload data in each direction (50 frames/s constant, average of 2500 Bytes/frame with negative exponential distribution).

Both flows are generated by NETPERFMETER.

For bottleneck scenario B1, we are interested in testing VM resource overload. This is a typical scenario in cloudified environments where a co-tenant VM hosted on the same physical server might monopolize a resource, such as CPU, bandwidth, memory, or disk I/O. For this purpose, we provision the VM running the SPGW-U with 4 CPU cores and 3 GiB of memory. We test this scenario by running a Linux stress tool, called STRESS [56], inside the VM hosting the SPGW-U and impose workload on its CPU, memory and disk I/O resources. More specifically, we spawn 4 workers doing heavy CPU computations, 4 workers doing file operations, and 4 workers each consuming 256 MiB of memory (i.e. 1 GiB in total) while continuously doing memory operations. We keep track of these metrics through PROMETHEUS and SYSSTAT. Further, heavy bottlenecks of this kind could be indirectly detected in the packet capture level. Figure 4 shows the effect of the B1 bottleneck as noticed by the user. The average bitrate of TCP drops by more than 10%, while the UDP traffic is not much affected. Other scenarios involving packet loss have more drastic effects on user traffic, especially TCP.

In bottleneck scenario B2, we explore two packet loss scenarios, where we simultaneously drop packets at the egress direction of the SPGW-U interfaces facing the SGi and S1-U links. These two links make up the user-plane traffic pathway from the Internet to the UE and are susceptible to packet loss due to the large and unpredictable volumes of data that they serve, as well as the complexities of mobile network handovers. To this end, the first scenario represents a faulty network interface that corrupts a big number of packets. To emulate loss, we utilize the NETEM [57] queuing discipline on the SPGW-U, where we set it to drop either 0.1% (cable insulation) or 10% (faulty interface) of the outgoing packets in each direction (i.e. towards the Internet as well as towards the eNodeB; see Figure 2). Another possible cause of packet loss is hardware problems. As a result, the second packet loss scenario represents faulty cable insulation, which due to interference corrupts a small number of packets. We expect packet loss to be captured by metrics gathered through PROMETHEUS and SYSSTAT as well as packet analysis.

Mobile backhaul, the transport network connecting the mobile core and the Radio Access Network (RAN), may lose connectivity over the default path. In such cases, the network switches to a backup path, which may be significantly under-provisioned to handle the typical mobile user traffic. In bottleneck scenario B3, we assume that the network switches its backhaul to a satellite-like link. Therefore, we account for the presence of low-capacity links such as those provided by satellite networks. We combine a Hierarchical Token Bucket filter (HTB) with NETEM in the SPGW-U to restrict the network capacity to 2 Mbit/s (by HTB) as well as to apply a one-way delay of 250 ms (by NETEM). As with B2, the applied QoS characteristics are symmetric, i.e. the same for both directions.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

Finally, in bottleneck scenario B4, we experiment with different link bandwidth limits, while saturating the network. B4 deviates from the rest of the bottleneck scenarios as the total duration of the experiments is 300 seconds and the bottleneck is always present. We use the same bidirectional 1 Mbit/s UDP flow, described above and in addition perform a TCP download, a TCP upload, or a bidirectional TCP traffic. The bandwidth limits are set on the outgoing traffic of SPGW-U and are 2 and 10 Mbit/s. It should be noted that due to the limitations of OAI, the maximum traffic that can be transmitted to the mobile user under normal conditions is approximately 12 Mbit/s.

V. Analytics

In this section we provide the results obtained from running the different anomaly detection autoencoder-based solutions explained in Subsection III-B for the bottlenecks described in Subsection IV-C. It must be noted that classical machine learning models, such as the ARMA solution of [3], support vector machines, principal component analysis, Gaussian mixture models, and isolation forest have also been tried, but the performance was found to be consistently below the deep architectures' and thus, these results have not been included in the analysis.

A. Data Preprocessing

An anomaly detection model is trained on each network component independently, i.e. MME, HSS, SPGW-U and SPGW-C (see Subsection IV-A for details), for both SysStat and OSM monitoring datasets. The models performing anomaly detection are self-supervised, that is, they are trained on data not containing abnormal samples (baseline scenario B0) and are tested on each experiment with bottleneck. The B0 baseline experiment data is split into two, training and validation, consisting of 80% and 20% of the total data, respectively. Models reporting the lowest loss in the validation dataset are stored and used to obtain the results provided in this subsection. All data is normalized between 0 and 1 using MinMax scaler before being fed into the model. While the OSM dataset contains only 14 features, SysStat data has more than 100, and thus, requires feature selection which is based on variance, i.e., measures that are constant throughout the whole experiment are discarded. Both the models outlined below are trained using the mean square error (MSE) as loss function, and ADAM optimizer for stochastic gradient descent.

B. Models

1) DAE: During the training phase, each sample is corrupted with additive white Gaussian noise of $\mathcal{N}(0, 0.1)$ following Equation (5). The encoder inputs are processed by three fully connected layers consisting of 64, 32, and 16 neurons units, respectively. At the decoder, the dimensionality reduction process is reverted, so each encoded sample c is processed by three fully connected layers of size, 16, 32, and 64. Each fully connected layer is followed by a rectified linear unit (ReLU) activation function with a negative slope 10^{-2}, except for the last layer of decoder where a Sigmoid function is used. A learning rate of $l_r = 1e^{-3}$ and weight decay of $w_d = 1e^{-5}$ is deployed over a total of 250 episodes.

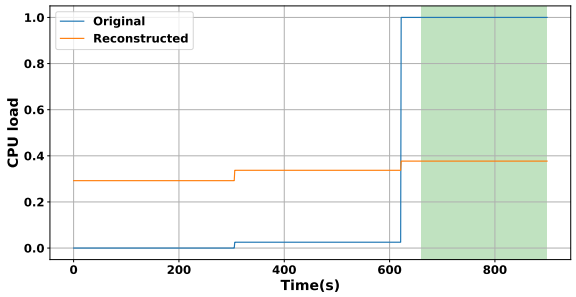


Figure 5: Anomaly detection using CAE for bottleneck scenario B1 at SPGW-U. Total CPU load metric, reported by OSM. Green indicates window anomalies.

2) CAE: The model is fed with data collected in batches of 60 consecutive samples each, and trained over a total of 50 epochs, after which it reaches convergence. The encoder's architecture for SysStat (OSM) data consists of three 1D-convolutional layers with filter size 3, stride 1 and padding 3, each followed by a rectified linear unit of slope 0.2, and dropout of 0.2, to avoid over-fitting. The last layer of the encoder is given by a normalization layer, using IterNorm [58]. The decoder is a mirrored version of the encoder, except for the absence of the normalization layer. A learning rate of $1e{-}3$ has been used.

C. Thresholding

One of the main challenges when tuning the hyper parameters associated with anomaly detection solutions is the choice of the threshold l_{th} that will trigger a sample to be classified as an anomaly. Thresholds are chosen based on percentiles of the validation loss distribution, for both the total and the per-feature loss (Equation 4). We will call these *total* and *per-feature thresholds*. The former approach allows to easily infer whether some network component is behaving abnormally, while the latter permits to look into each counter and map the anomaly to a specific metric. In order to dampen spikes in errors, to improve accuracy and reduce false positives, smoothing of the validation loss is commonly adopted, before looking at the distribution and setting the threshold, [59]. Thresholds obtained from the smoothed validation loss (total or per-feature) will be called *smoothed thresholds*. For both approaches, once the error distribution for the validation dataset is obtained, the error corresponding to the 95^{th} percentile is chosen to be the threshold l_{th} parameter. Once the threshold is set, samples that show a reconstruction error above l_{th} are labelled as anomalies. Furthermore, depending on whether models classify each sample individually or in a time-frame manner, we will refer to the detected anomalies as *timestamp anomalies* and *window anomalies* respectively.

D. Experimental results

Results for anomaly detection on the OSM dataset for both the DAE and the CAE models on all bottlenecks were not satisfying since for this data source the models are very sensitive to hyperparameter tuning. This is probably due to

that most of the reported measures of OSM monitoring source follow a time series "step-function". The models struggle to learn and they do not converge. Thus, OSM monitoring data was not used for anomaly detection. Figure 5 illustrates how the CAE model attempts to reconstruct a counter from OSM dataset for bottleneck B1, failing to individuate the correct anomaly region as described in the corresponding bottleneck scenario.

1) Bottleneck B1: Both the DAE and the CAE can detect anomalies in SYSSTAT data for bottleneck B1 happening at SPGW-U between $t \in [300, 600]s$, while no anomalies are reported in any other timestamp outside of that time frame, or in any other network component. Results for the DAE are shown in Figure 6. These plots illustrate how, based on the *total threshold*, the DAE solution detects *timestamp anomalies* happening at SPGW-U between $t \in [300, 600]s$, while no anomalies are reported in any other time outside that time frame, or in any other network component. The fact that the DAE solution works on a per-sample basis, i.e., mapping each sample individually to either 1 (anomaly) or 0 (normal), explains the spurious signal seen in Figure 6a. Furthermore, Figure 6b also shows the reconstructed signal of one of the metrics reported by the SYSSTAT monitoring tool compared to the original. In this case, it is a metric tracking the load of one of the CPU cores: CPU-0. As it can be seen, the reconstructed metric almost matches the original, showing that, this particular metric follows the latent distribution learned by the DAE model. In contrast, Figure 6c shows how the reconstructed measure of another CPU core: CPU-2 matches the original signal from time 0 to 300. At 300 a bottleneck is introduced in the system, and thus the reconstructed and original signal diverge significantly, causing the model to identify an anomaly, and being able to identify the root cause of the anomaly, which is in this case, the CPU-2 measure. Faced with this anomaly, the network operators could react by scaling up CPU-2, as the original measure is higher than the reconstructed. For example they could provision the VNF with more CPU cores or launch more instances of the VNF to attempt load balancing. Thus, the DAE not only allow us to detect an anomaly but also to identify the component causing it so that infrastructure managers can react accordingly.

Results for CAE are shown in Figure 7, where a *per-feature smoothed* threshold has been obtained after smoothing the per-feature loss by taking the average loss per batch. Anomalies have been detected for counters connected to the machine overload, in line with the description for the B1 scenario reported in Table II. In Figure 7 three counters reported anomalies. A thorough description of the counters is found in the documentation file [60]. We conclude that SYSSTAT level metrics are enough to detect this type of bottleneck and thus more expensive packet capture metrics are not needed.

2) Bottleneck B2: For experiment B2 no anomaly is detected using the DAE model while anomalies in very few SYSSTAT metrics are detected by the CAE model at the eNodeB. Given that the anomalies are not present in most metrics, a per-feature smoothed threshold was the only successful thresholding technique achieved by CAE. Figure 8 reports the only metric having an abnormal behaviour throughout the entire window $[300, 600]s$ where the 10% packet loss occurred. This metric is "total number of kilobytes transmitted per second" from the physical interface of the eNodeB facing the EPC core. It includes traffic from both S1-U and S1-C links. Even though we can not differentiate the traffic of these two links at this level, traffic from S1-C is minimal, thus any anomaly should be attributed to S1-U, the link connecting the eNodeB and SPGW-U. The bottleneck is inserted at the IP layer and, given that OSM places its focus on the monitoring of the hardware of the underlying infrastructure, no OSM monitored metric presents an abnormal behavior for this concrete bottleneck. This type of bottleneck exemplifies that complex monitoring solutions have to be implemented in order to understand the current state of the network.

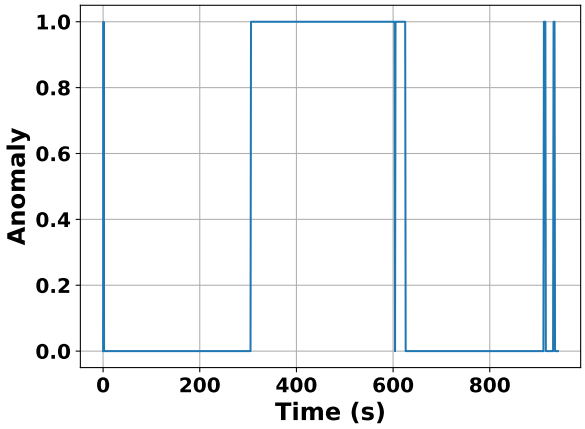
In this particular case, this bottleneck is more easily detected via the Packet Capture (PCAP) files [61]. For our analysis, we capture the full packet trace, in order to examine the performance implications on different protocol flows. This is clearly time- and space-consuming. A real-world operator deployment would of course only capture the minimal required information, e.g. certain header fields of (a representative subset of) flows. In performance-critical environments, this could even be realised by custom high-performance hardware, e.g. in a Field Programmable Gate Array (FPGA) based implementation. However, such details are out of scope for this article. In the 0.1% packet loss case the effect in UDP traffic is unnoticeable, but if we monitor the control flow of TCP we can notice an increase in duplicate ACK packets signaling a problem and the subsequent drop of bitrate from the TCP flow. For the 10% packet loss case the effect is noticeable in both UDP and TCP. The TCP bitrate drops to almost 0 due to the excessive packet loss. We are also able to notice the loss if we compare the number of incoming packets per flow towards the SPGW-U and eNodeB. In case of fragmentation between these two nodes, counting packets is not sufficient, and we have to resort to measuring application payload (in case of UDP) or sequence numbers (in case of TCP). Thus, in this bottleneck scenario, sampling flows is a costly, but more trustworthy approach to detect the bottleneck.

3) Bottleneck B3: The results of DAE for bottleneck B3 are shown in Figure 9a, where a small set of anomalies are detected in the component MME around $t = 400$. Total loss thresholding makes the DAE approach fail to detect the B3 type of anomalies unless they are very steadfast, as is happening around time 400s, explaining why anomalies for the entire window $[300, 600]$ were not found. The same reasoning extend to the CAE model, and the results in Figure 10 show that, only small subset of few samples deviate from the expected outcome.

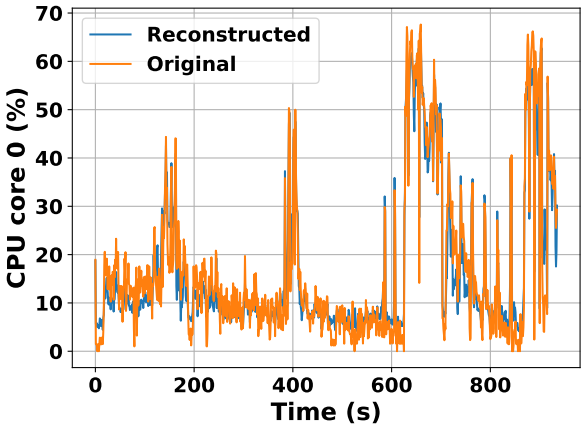
As above, packet captures can reveal the problem at the expense of higher cost. We may monitor the delay between a packet arriving at the SPGW-U and eNodeB. A sudden spike in the delay can not be explained by increased congestion, so it should be attributed to a link problem, especially if it affects all the traffic crossing a link. One such example for the TCP flow is shown in Figure 11.

4) Bottleneck B4: For experiment scenario B4, both the DAE and CAE solutions are able to detect anomalies happening at the eNodeB using SYSSTAT monitoring data, for almost all the duration of the experiment, as described in

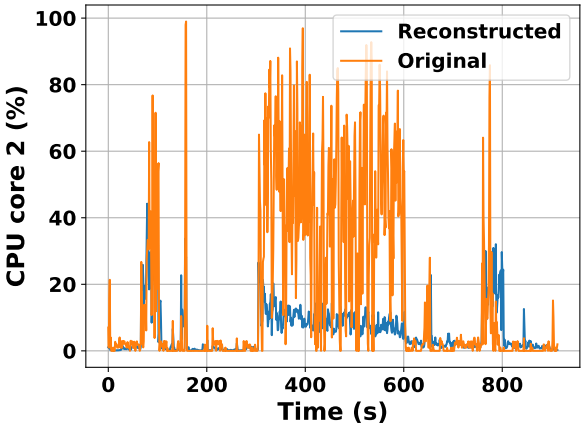
This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246



(a) Anomaly detection at the SPGW-U.

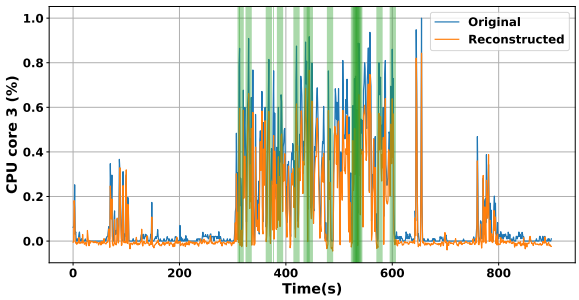


(b) Reconstruction of the CPU-0 metric.

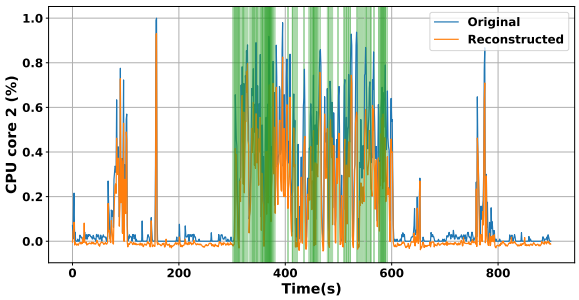


(c) Reconstruction of the CPU-2 metric.

Figure 6: Anomaly detection using DAE for bottleneck scenario B1 at different network components



(a) CPU-3 utilization metric.



(b) CPU-2 utilization metric.

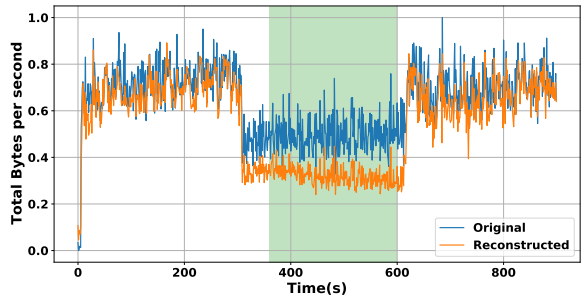
Figure 7: Anomaly detection using CAE for bottleneck scenario B1 at component SPGW-U for different features. Green indicates timestamp anomalies.

Section IV-C. Figure 9b shows how this component reports data samples that are very distinct from the ones learned using the baseline, resulting in most of the experiment samples being labelled as abnormal. Figure 12 shows two out of the several counters that have reported anomalies.

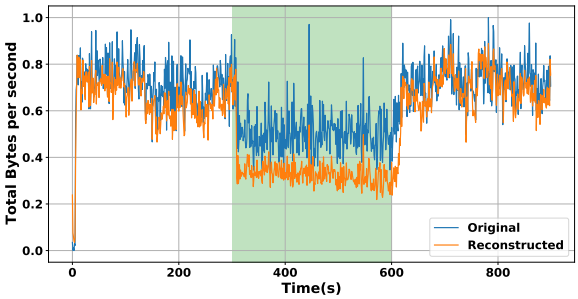
E. Time complexity

Both the models, DAE and CAE, used to perform anomaly detection are trained offline, that is, sets of historical data are collected and stored, and the models' parameters are optimised using the obtained data. Once the trained models are obtained, the anomaly detection for testing occurs offline, using data from the network where an artificial bottleneck is applied.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

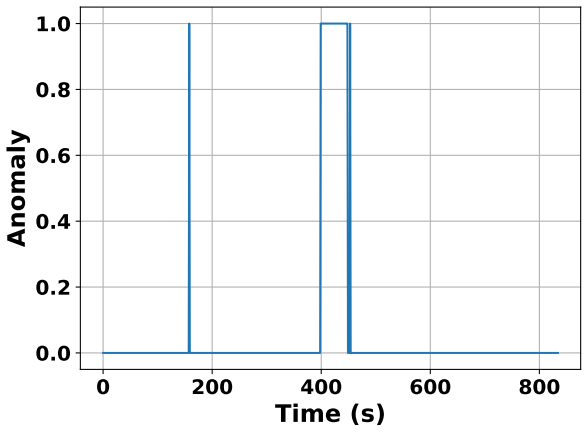


(a) Kilobytes transmitted per second metric.

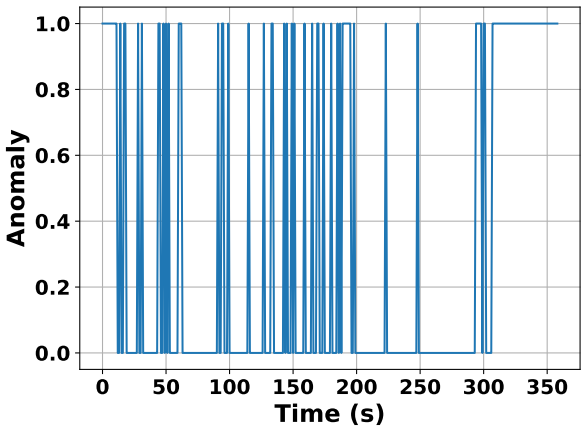


(b) Kilobytes transmitted per second metric.

Figure 8: CAE results for two different experiments in bottleneck scenario B2, for the physical eNodeB interface facing towards the EPC core. Green indicates window anomalies.

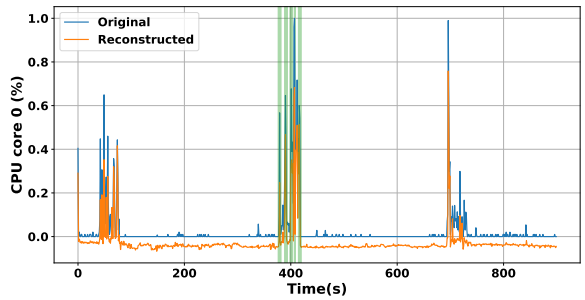


(a) Anomaly detection for bottleneck B3.

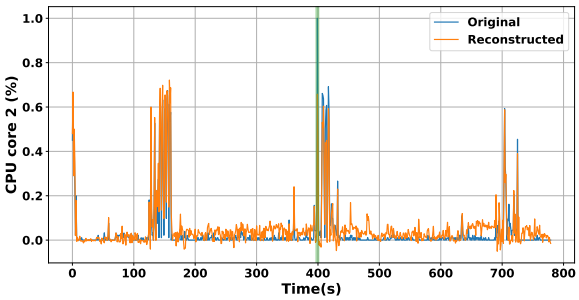


(b) Anomaly detection for bottleneck B4.

Figure 9: Results using DAE in bottleneck scenarios B3 and B4 at components MME and eNodeB.



(a) CPU-0 utilization measure.



(b) CPU-2 utilization measure.

Figure 10: Anomaly detection using CAE for bottleneck scenario B3 at MME component. Green indicates timestamp anomalies.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246

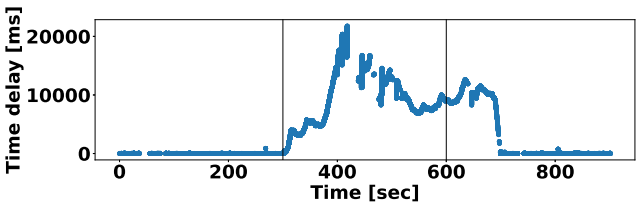


Figure 11: Time delay between packets of a TCP flow arriving at the SPGW-U and at the eNodeB. The vertical lines indicate the time period when the bottleneck is active.

Nevertheless, the anomaly detection could also happen online, as the combination of data preprocessing and inference takes around 15.78 seconds to label 300 data points from the SYSSTAT as normal/anomaly using an Intel(R) Core(TM) i9-9820X processor and a 2080TI NVIDIA graphic cards. The integration between the AI solution and the testbed in a live environment is one of the future research directions for the authors of this work.

VI. Discussion and Challenges

The results of the previous Section show that in order to get a full overview of the network situation, multiple monitoring sources have to be analyzed at once as no single source is able to cover all the complex variables involved in a network deployment. It must be noted that monitoring comes at a cost, as resources have to be allocated to gather and store the network metrics, which translates into increased operating costs. The nature of the bottleneck dictates at which level it may be detectable and, by extension, how costly the detection will be. VNF monitoring tools, designed to supervise the overall health of a VNF, perform well in detecting problems related to the cloud infrastructure and are in general, a cheap option. However, bottlenecks affecting traffic are often invisible to these tools, and therefore, an approach that monitors traffic at a lower level is necessary. Considering that network telemetry is very resource-intensive, hence costly, these solutions cannot be widely deployed in a network and should be aimed at a small number of network flows. Link problems concern most flows on the core network and thus, have a greater impact on TCP, making TCP flows a better sampling candidate. Problems affecting all flows should be visible in the subset of flows we sample. Solutions relying on a smart sampling of flows have been shown to achieve a good balance between resources and information gathering [62], [63] and as presented in Section II there is active research on developing solutions offering low-level network telemetry at reduced cost. Moreover, sampling random packets may reveal added delay and possibly packet loss.

The fact that multiple monitoring sources are needed dictates that various anomaly detection solutions must be used to correctly detect all possible types of anomalies. As we have shown above, a problem in the SPGW-U might manifest at the eNodeB level. Furthermore, some problems, such as high CPU usage, may affect the monitoring components themselves, impacting the capacity to detect and react to system faults. For example, a machine that is resource strained can miss or reorder packets during a packet capture, a problem that can hide the type of anomaly the network is facing. We can conclude that a combination of data sources with its corresponding anomaly detection solution, is the most reliable way to accurately identify the true nature and location of the bottlenecks.

In-band network telemetry [64], [65] allows the tracking of flows behaviour that traverses the network. For example, a P4 programmable switch can add extra headers to the forwarded packets, in order to indicate to another P4 switch to track the same flow, enabling the detection of packet loss or delay in the path between the two P4 switches [66]. In our experiments, we were able to detect these events in an offline manner, i.e., capturing the data generated at the different nodes and computing the delay manually afterward. In future works, we plan to deploy P4 switches, that will allow us to do part of the processing in real-time. Similarly, tools such as PPTmon [23], [54], may provide similar insights on the health of traffic flows for a significantly reduced processing cost compared to packet captures.

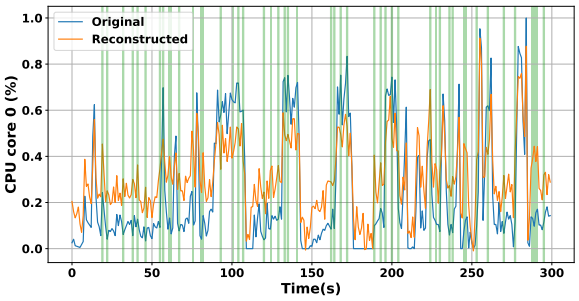
In this work, we focused on detecting a single bottleneck at a time, it is expected that a real network deployment might experience several simultaneous bottlenecks at different parts of the network [24]. Thus, increasing the complexity of anomaly detection as each monitoring node might report anomalies caused by both, its own local fault and the delivered effects of bottlenecks located at other network components. A further extension of this analysis consists of investigating how distributed monitoring and analytics can address these types of situations. Finally, we only studied bottlenecks located at the core network. In terms of QoS, bottlenecks could occur at different segments of the network, but RAN related problems are very different to what we study and are not easily detectable by our available tools. In addition, we would need access to, not easily accessible, vendor-specific logs. Therefore RAN bottlenecks are out of scope of this study.

VII. Future Work and Research Directions

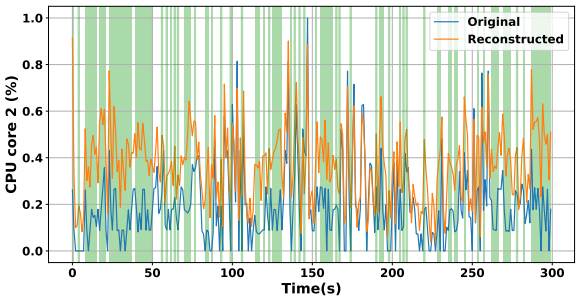
Identifying the bottleneck type and recognizing the bottleneck pattern is the first step of root cause analysis, which is the foundation of closed-loop automation. Once the bottleneck and potential root causes are identified, the E-Analyzer can send the information back to the internal Analyzer (see Figure 1), which then combines the internal analytical results to decide what actions should be taken to deal with the root causes and handle the bottleneck. In this way, an external closed loop is complete, with both internal MAPE stages and external MA (Monitoring-Analyzing) stages. In the testbed setup (Figure 2), using the bottleneck identification outcomes, the E-Analyzer will classify bottlenecks into two classes: *compute-resource-related* (e.g., VM overloading) and *network-resource-related* (e.g., bandwidth limitation, congestion). The former will be returned to OSM for resource-related handlings, such as scaling VNFs. The latter needs to be dealt with by other management systems, like an SDN controller for connectivity management.

In order to automate the closed-loop control, *integration* is necessary. In the future, the E-Monitoring and E-Analyzing modules will be integrated with OSM such that E-Analyzer outcomes can be fed back to OSM in real-time. OSM is open

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2022.3203246



(a) CPU-0 utilization measure.



(b) CPU-2 utilization measure.

Figure 12: Anomaly detection using CAE for bottleneck scenario B4 at the eNodeB for different features. Green indicates timestamp anomalies.

to integrate with external AI/ML framework and enable more advanced and predictive analytics from external parties, e.g., via AI agents [67].

Furthermore, as we move towards 6G, industry experts [68] believe that mobile networks will use highly distributed AI, where edge computing nodes will have their own AI resources rather than delegating the analytics tasks to a central cloud. Edge Intelligence (EI), as it is called, may relieve centralized components from management and monitoring overhead. It has the potential to significantly reduce the delay of the MAPE-K feedback loop, allowing for scalability and better adoption by industrial applications.

VIII. Conclusion

To assure the services SLA are delivered to customers, it is critical for Telecom operators to monitor, detect, identify, and resolve the issues causing service degradation. In this work, we leverage the exposed capabilities of 5G networks to deploy external analytic tools to enhance the network Service Assurance. To do so, a mobile network testbed is implemented, that uses a virtualized mobile core orchestrated by OSM. Its monitoring relies on both, the built-in OSM MON module and external monitoring tools. These measurements are then exposed to external analytics functions, using an adaptation of the MAPE-K model of service exposure to mobile networks. The external functions then provide advanced bottleneck detection methodologies based on AI to accurately identify performance bottlenecks. In particular, the proposed AI solutions leverage deep learning to detect anomalies and identify the network component causing them. The two solutions are based on state-of-the-art anomaly detection approaches adapted to the particularities of network data and leverage the data exposed by the distinct monitoring tools. The first approach leverages denoising autoencoders, where random noise is added to the training samples to increase generalization. On the other hand, the second solution combines convolutional neural networks, that capture the temporal correlation between samples, and autoencoders to detect anomalies. Once the faulty component is identified by the external AI module, the loop from monitoring to bottleneck detection can be closed by triggering countermeasures and bottleneck resolution. This work aims to showcase a framework for a comprehensive fault resolution system that could be fully integrated into a real-world network deployment. As future work, we plan to additionally exploit logging mechanisms from deployed softwarized network services and explore their effectiveness in identifying faster and more accurately the root cause of failures.

Acknowledgment

This work has been supported by the European Community through the 5G-VINNI project (grant no. 815279) within the H2020-ICT-17-2017 research and innovation program, and by the Norwegian Research Council through the "ML4ITS – Machine Learning for Irregular Time Series" project (grant no. 312062) within the IKTPLUSS Researcher Project – Transformative Research Project.

References

[1] S. Nováczki, "An Improved Anomaly Detection and Diagnosis Framework for Mobile Network Operators," in *Proceedings of the 9th IEEE International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2013, pp. 234–241.

[2] B. Gajic, S. Nováczki, and S. Mwanje, "An Improved Anomaly Detection in Mobile Networks by using Incremental Time-Aware Clustering," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 1286–1291.

[3] M. Xie, F. Michelinakis, T. Dreibholz, J. S. Pujol-Roig, S. Malacarne, S. Majumdar, W. Y. Poe, and A. M. Elmokashfi, "An Exposed Closed-Loop Model for Customer-Driven Service Assurance Automation," in *Proceedings of the 30th IEEE European Conference on Networks and Communications (EuCNC)*, 2021.

[4] 5G-VINNI, "Deliverable D3.1: Specification of Services Delivered by Each of the 5G-VINNI Facilities," 2019.

[5] A. Reid, A. González, A. E. Armengol, G. G. de Blas, M. Xie, P. Grønsund, P. Willis, P. Eardley, and F.-J. R. Salguero, "OSM Scope, Functionality, Operation and Integration Guidelines," ETSI, White Paper, Jun. 2019.

[6] P. Rengaraju, C.-H. Lung, F. R. Yu, and A. Srinivasan, "On QoE Monitoring and E2E Service Assurance in 4G Wireless Networks," *IEEE Wireless Communications*, vol. 19, no. 4, 2012.

[7] N. Baranasuriya, V. Navda, V. N. Padmanabhan, and S. Gilbert, "QProbe: Locating the Bottleneck in Cellular Communication," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015, p. 33.

[8] A. P. Iyer, L. E. Li, and I. Stoica, "Automating Diagnosis of Cellular Radio Access Network Problems," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 2017, pp. 79–87.

[9] J. M. A. Calero and J. G. Aguado, "MonPaaS: An Adaptive Monitoring Platform as a Service for Cloud Computing Infrastructures and Services," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 65–78, 2015.

[10] Y. Li *et al.*, "FlowRadar: A Better NetFlow for Data Centers." in *NSDI*, 2016, pp. 311–324.

[11] M. Moshref *et al.*, "Trumpet: Timely and Precise Triggers in Data Centers," in *Proceedings of the ACM SIGCOMM Conference*. ACM, 2016, pp. 129–143.

[12] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "NFV-VITAL: A Framework for Characterizing the Performance of Virtual Network Functions," in *Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 2015, pp. 93–99.

[13] P. Naik, D. K. Shaw, and M. Vutukuru, "NFVPerf: Online Performance Monitoring and Bottleneck Detection for NFV," in *Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. IEEE, 2016, pp. 154–160.

[14] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed Diagnosis in Enterprise Networks," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 243–254.

[15] A. S. Rajan, S. Gobriel, C. Maciocco, K. B. Ramia, S. Kapury, A. Singhy, J. Ermanz, V. Gopalakrishnanz, and R. Janaz, "Understanding the Bottlenecks in Virtualizing Cellular Core Network Functions," in *nternational Workshop on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2015, pp. 1–6.

[16] J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. Andres-Maldonado, and J. M. Lopez-Soler, "Modeling and Dimensioning of a Virtualized MME for 5G Mobile Networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 4383–4395, 2017.

[17] M. T. Raza, D. Kim, K.-H. Kim, S. Lu, and M. Gerla, "Rethinking LTE Network Functions Virtualization," in *25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–10.

[18] A. Giorgetti, K. Kondepu, A. Sgambelluri, D. Melkamu, N. Sambo, M. Capitani, G. Landi, and L. Valcarenghi, "Demonstration of fault localisation and recovery of optical connectivity supporting 5G vRAN," *45th European Conference on Optical Communication (ECOC 2019)*, 2019.

[19] V. Sciancalepore *et al.*, "z-TORCH: An Automated NFV Orchestration and Monitoring Solution," *IEEE Transactions on Network and Service Management*, 2018.

[20] A. P. Iyer, L. E. Li, M. Chowdhury, and I. Stoica, "Mitigating the Latency-Accuracy Trade-off in Mobile Data Analytics Systems," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: ACM, 2018, pp. 513–528.

[21] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi, and G. Pavlou, "Self-Adaptive Decentralized Monitoring in Software-Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1277–1291, 2018.

[22] M. Mekki, S. Arora, and A. Ksentini, "A Scalable Monitoring Framework for Network Slicing in 5G and Beyond Mobile Networks," *IEEE Transactions on Network and Service Management*, 2021.

[23] B.-H. Oh, S. Vural, and N. Wang, "A Lightweight Scheme of Active-Port-Aware Monitoring in Software-Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2888–2901, 2021.

[24] G. Patounas, X. Foukas, A. Elmokashfi, and M. K. Marina, "Characterization and Identification of Cloudified Mobile Network Performance Bottlenecks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2567–2583, 2020.

[25] Y. Zuo, Y. Wu, G. Min, C. Huang, and K. Pei, "An Intelligent Anomaly Detection Scheme for Micro-Services Architectures With Temporal and Spatial Data Analysis," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 548–561, 2020.

[26] V. Theodorou, A. Lekidis, T. Bozios, K. Meth, A. Fernandez-Fernandez, J. Taylor, P. Diogo, P. Martins, and R. Behravesh, "Blockchain-based Zero Touch Service Assurance in Cross-domain Network Slicing," in *Proceedings of the 30th IEEE European Conference on Networks and Communications (EuCNC)*, Porto, Portugal, Jun. 2021.

[27] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier Detection for Temporal Data: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2013.

[28] 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.

[29] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially Learned Anomaly Detection," in *2018 IEEE International conference on data mining (ICDM)*. IEEE, 2018, pp. 727–736.

[30] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based Encoder-Decoder for Multi-Sensor Anomaly Detection," *arXiv preprint arXiv:1607.00148*, 2016.

[31] P.-C. Lin, "Large-Scale and High-Dimensional Cell Outage Detection in 5G Self-Organizing Networks," in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2019, pp. 8–12.

[32] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: UnSupervised Anomaly Detection on Multivariate Time Series," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3395–3404.

[33] H. Lee, J. Cha, D. Kwon, M. Jeong, and I. Park, "Hosting AI/ML Workflows on O-RAN RIC Platform," in *2020 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2020, pp. 1–6.

[34] M. A. Kramer, "Nonlinear Principal Component Analysis using Autoassociative Neural Networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[35] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," *Advances in Neural Information Processing Systems*, vol. 2, 1989.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[37] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D Convolutional Neural Networks and Applications: A Survey," *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, 2021.

[38] S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1-D convolutional neural networks," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 3, pp. 664–675, 2015.

[39] O. Avci, O. Abdeljaber, S. Kiranyaz, and D. Inman, "Structural damage detection in real time: implementation of 1D convolutional neural networks for SHM applications," in *Structural Health Monitoring & Damage Detection*. Springer, 2017, vol. 7, pp. 49–54.

[40] S. Kiranyaz, T. Ince, O. Abdeljaber, O. Avci, and M. Gabbouj, "1-D Convolutional Neural Networks for Signal Processing Applications," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 8360–8364.

[41] I. Mitiche, A. Nesbitt, S. Conner, P. Boreham, and G. Morison, "1D-CNN-Based Real-Time Fault Detection System for Power Asset Diagnostics," *IET Generation, Transmission & Distribution*, vol. 14, no. 24, pp. 5766–5773, 2020.

[42] T. Dreibholz and A. F. Ocampo, "Managing Tailor-Made Enhanced Packet Cores for 4G/5G Testbeds in OSM with the SimulaMet OpenAirInterface VNF," OSM Hackfest, Dec. 2020.

[43] T. Dreibholz, "Flexible 4G/5G Testbed Setup for Mobile Edge Computing using OpenAirInterface and Open Source MANO," in *Proceedings of the 2nd International Workshop on Recent Advances for Multi-Clouds and Mobile Edge Computing (M2EC) in conjunction with the 34th International Conference on Advanced Information Networking and Applications (AINA)*, Caserta, Campania/Italy, Apr. 2020, pp. 1143–1153.

[44] ——, "A 4G/5G Packet Core as VNF with Open Source MANO and OpenAirInterface," in *Proceedings of the 28th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Hvar, Dalmacija/Croatia, Sep. 2020.

[45] A. F. Ocampo, T. Dreibholz, M.-R. Fida, A. M. Elmokashfi, and H. Bryhni, "Integrating Cloud-RAN with Packet Core as VNF Using Open Source MANO and OpenAirInterface," in *Proceedings of the 45th IEEE Conference on Local Computer Networks (LCN)*, Sydney, New South Wales/Australia, Nov. 2020, Demo presentation.

[46] N. Nikaein *et al.*, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM CCR*, vol. 44, no. 5, pp. 33–38, 2014.

[47] OpenStack, "OpenStack Architecture Design Guide," https://docs.openstack.org/arch-design, 2018.

[48] ——, "OpenStack Operations Guide," https://docs.openstack.org/operations-guide/, 2019.

[49] T. Dreibholz, "Evaluation and Optimisation of Multi-Path Transport using the Stream Control Transmission Protocol," Habilitation Treatise, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, Mar. 2012.

[50] ——, "NetPerfMeter: A Network Performance Metering Tool," http://blog.multipath-tcp.org/blog/html/2015/09/07/netperfmeter.html, 2015.

[51] T. Dreibholz, M. Becke, H. Adhari, and E. P. Rathgeb, "Evaluation of A New Multipath Congestion Control Scheme using the NetPerfMeter Tool-Chain," in *Proceedings of the 19th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Hvar, Dalmacija/Croatia, Sep. 2011, pp. 1–6.

[52] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Testing Virtual Network Functions Auto-Scaling using Open-Source Management and Orchestration," in *2021 Telecoms Conference (ConfTELE)*, 2021, pp. 1–6.

[53] S. Godard, "Performance Monitoring Tools for Linux," https://github.com/sysstat/sysstat, 2018.
[54] N. V. Tu, J.-H. Yoo, and J. W.-K. Hong, "PPTMon: Real-time and Fine-grained Packet Processing Time Monitoring in Virtual Network Functions," *IEEE Transactions on Network and Service Management*, 2021.
[55] T. Dreibholz, "HiPerConTracer - A Versatile Tool for IP Connectivity Tracing in Multi-Path Setups," in *Proceedings of the 28th IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Hvar, Dalmacija/Croatia, Sep. 2020.
[56] A. Waterland, "stress - tool to impose load on and stress test a computer system," https://github.com/resurrecting-open-source-projects/stress, 2021.
[57] Linux Network Developers , "Netem," https://wiki.linuxfoundation.org/networking/netem, 2018.
[58] L. Huang, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Iterative Normalization: Beyond Standardization Towards Efficient Whitening," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4874–4883.
[59] D. T. Shipmon, J. M. Gurevitch, P. M. Piselli, and S. T. Edwards, "Time Series Anomaly Detection: Detection of Anomalous Drops with Limited Features and Sparse Examples in Noisy Highly Periodic Data," *arXiv preprint arXiv:1708.03665*, 2017.
[60] S. Godard, "Sar Manual Page," Aug. 2020.
[61] U. Lamping, R. Sharpe, E. Warnicke, and G. Combs, "Wireshark User's Guide," Apr. 2020.
[62] Cisco, "Cisco IOS NetFlow," 2021.
[63] K. Bartos, M. Rehak, and V. Krmicek, "Optimizing flow sampling for network anomaly detection," in *2011 7th International Wireless Communications and Mobile Computing Conference*, 2011, pp. 1304–1309.
[64] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-Band Network Telemetry: A Survey," *Computer Networks*, vol. 186, p. 107763, 2021.
[65] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic Un-Band N etwork Telemetry," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2020, pp. 662–680.
[66] F. Paolucci, F. Cugini, P. Castoldi, and T. Osiński, "Enhancing 5G SDN/NFV Edge with P4 Data Plane Programmability," *IEEE Network*, vol. 35, no. 3, pp. 154–160, 2021.
[67] Francisco Javier Ramon and Subhankar Pal, "The Relation of ENI Policy Management to Network Intelligence: Relation to Open Source MANO (OSM)," https://docbox.etsi.org/ISG/ENI/Open/20200925_Workshop_ENI_policy, Sep. 2020.
[68] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Casti neira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, "6G White Paper on Edge Intelligence," *arXiv preprint arXiv:2004.14850*, 2020.