Bottleneck Identification in Cloudified Mobile Networks based on Distributed Telemetry

Mah-Rukh Fida*‡, Azza H. Ahmed*†, Thomas Dreibholz*,
Andrés F. Ocampo*†, Ahmed Elmokashfi* and Foivos I. Michelinakis*
*Simula Metropolitan Center for Digital Engineering, Oslo, Norway
†Oslo Metropolitan University, Oslo, Norway
‡ School of Computing and Engineering, University of Gloucestershire, United Kingdom

Abstract—Cloudified mobile networks are expected to deliver a multitude of services with reduced capital and operating expenses. A characteristic example is 5G networks serving several slices in parallel. Such mobile networks, therefore, need to ensure that the SLAs of customised end-to-end sliced services are met. This requires monitoring the resource usage and characteristics of data flows at the virtualised network core, as well as tracking the performance of the radio interfaces and UEs. A centralised monitoring architecture can not scale to support millions of UEs though. This paper, proposes a 2-stage distributed telemetry framework in which UEs act as early warning sensors. After UEs flag an anomaly, a ML model is activated, at network controller, to attribute the cause of the anomaly. The framework achieves 85% F1-score in detecting anomalies caused by different bottlenecks, and an overall 89% F1-score in attributing these bottlenecks. This accuracy of our distributed framework is similar to that of a centralised monitoring system, but with no overhead of transmitting UE-based telemetry data to the centralised controller. The study also finds that passive in-band network telemetry has the potential to replace active monitoring and can further reduce the overhead of a network monitoring system.

Index Terms—Bottleneck, Congestion, Mobile Cloud Network, Telemetry, Anomaly, Classification.

1 Introduction

A cloudified mobile network represents a network architecture in which the mobile network functions and services are virtualised, enabling them to be run on cloud-based platforms. This approach leverages the scalability, flexibility, and cost-effectiveness of cloud computing technologies to provide a multi-service infrastructure with service assurance and simplified network management [1]. In the near future, these networks are expected to carry, beside today's best effort traffic, a multitude of use cases with stringent requirements e.g., IoT, industrial automation and highly interactive multiverse traffic [2].

To ensure that the different tenants of a cloudified mobile network are accommodated, it needs to quickly detect and remediate performance degradation in the end services. Detecting performance degradation necessitates the timely collection of representative telemetry, the automation of flagging any performance degradation, root cause attribution and finally the design of an effective control system that re-configures the affected network elements.

This paper proposes a network monitoring framework to timely detect and trigger the attribution of a performance issue in a cloudified mobile network. Unlike data centre networks, the telemetry architecture in mobile networks have received little attention. Differences between the two types of networks, especially the challenging radio interface, make adopting data centre approaches insufficient. We tackle this by employing a 2-stage distributed telemetry framework, in which User Equipments (UEs) act as early warning sensors at stage-1 of the telemetry framework, and triggers the stage-2 by flagging the occurrence of an anomaly to the mobile network controller. An anomaly is flagged as a result of performance degradation in an end-service. The mobile network controller, then uses a supervised Machine Learning (ML) algorithm to identify the root cause of the anomaly. The ML model is built up on periodic active, passive and in-band telemetry monitoring of different links and components of the cloudified mobile network, where periodicity can be tuned after striking right balance between the overhead of telemetry collection, transmission and the non-deterministic performance-related state of the mobile network. A classical shortcoming of a supervised ML approach is the fact that training may not be comprehensive enough i.e., it does not cover emerging or unknown anomalies. Other than minimizing the monitoring overhead, a by-product of our distributed telemetry framework is that it signals out new types of anomalies and uses them to further improve the system.

To assess our 2-stage distributed framework, we have built a mobile network testbed based on the cloudified architecture. We imitate performance degradation in the network by introducing a set of bottleneck profiles, both in the Radio Access Network (RAN) and the mobile network core. These profiles encompass bandwidth congestion and transmission delays on network links, data loss at relay components, strained network resources within the network cloud, and radio interference in the final mile.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. 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/TMC.2023.3312051

For troubleshooting mobile networks, its operators collect key performance indicators (KPIs), usually every hour, at various network elements like basestations and core servers [3]. These KPIs are monitored for deviations from the norm using simple thresholds, but the approach is slow, imprecise, and reactive [4]. To address these challenges, ML methods are increasing considered [4], [5] to automate network troubleshooting. Deep learning, a subset of machine learning, is particularly suitable for such networking issues because it can handle large amounts of complex and high-dimensional data. Additionally, extracting relevant features or characteristics from the data is often necessary in troubleshooting network problems. Deep learning models have the capability to automatically extract these relevant features during the learning process [6]. We, therefore, use simple deep learning models both for bottleneck detection, at stage-1, and for its attribution, at stage-2, of our distributed telemetry framework.

In this study, we make the following contributions:

1) Proposing bottleneck detection for mobile networks, to be leveraged from end devices attached to the network edge, i.e., UEs. At this stage (i.e., at stage-1), we use Variational Autoencoder and achieve an impressive performance of 85% F1-score in detecting bottlenecks. This accuracy is not only 2% higher but also includes detection of bottlenecks at RAN that are missed by a recent related work [7].
2) Presenting a viable solution to the problem of using supervised ML for classifying problems with potentially, yet, unlearned classes in the context of mobile networks. At this stage (i.e., at stage-2) we use Multi-Layer Perceptron (MLP) that attributes bottlenecks with 89% F1-score. Th accuracy is 18% higher than that of a similar state-of-the-art study [4], conducted on cloudified mobile network.
3) Empirically evaluating our approach in the lab in comparison with a widely-used centralised telemetry framework. Our proposed distributed framework shows comparable accuracy to that of a centralised approach, but with no overhead of transmitting telemetry information from UEs to the centralised controller.
4) Investigating the potential of in-band network telemetry in comparison with the traditional active and passive monitoring methods, in a cloudified mobile network.

The paper comprises of eight more sections. After discussing the background and motivation of the study in Section 2, we explain our system architecture in Section 3. In Section 4, we present our distributed framework for bottleneck identification. Section 5 illustrates the types of bottlenecks that we analyze along with machine learning models for their detection and attribution. Section 6 evaluates the 2-stage distributed telemetry framework in comparison with a centralised system and with two state-of-the art works. Furthermore, this section investigates the potential and overhead of different monitoring approaches used by the framework. We then point out the challenges in the distributed framework in Section 7, followed by an overview of related work in Section 8. The paper is finally concluded, in Section 9, with key takeaways from the study.

2 Background and Motivation

We term the events causing performance degradation to end-users, in the mobile network, as bottlenecks. Bottlenecks may appear due to issues in the wireless link or network congestion, with an observable impact on end-users, such as depicted in Figure 1. The figure shows bit rate (Mbit/s), RTT and percentage of lost datagrams with IPERF and UDP ping commands from a UE to a test server in our experimental cloudified mobile testbed (mentioned in Section 3), without any bottleneck on end-to-end path and with different bottleneck profiles (illustrated in Table 2). We observe that the bottlenecks indeed impact the performance parameters at the end-devices, which in-turn effects Quality-of-Experience (QoE). For example, studies like [8], [9], [10] show that poor throughput, latency and packet loss result in slow and unresponsive applications such as poor audio or video quality during calls, slow loading times for apps and websites, and increased buffering during media streaming.

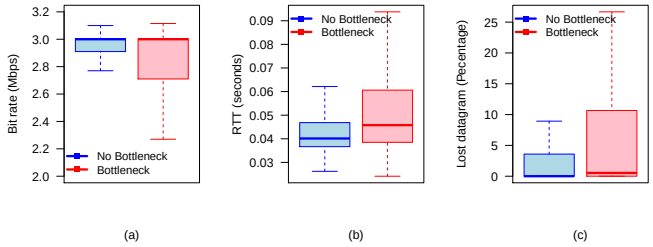

Fig. 1: The impact of network bottleneck on performance degradation at UE.

Identifying bottlenecks helps network operators to optimise their infrastructure. By pinpointing sources of bottlenecks, operators can allocate resources more effectively, such as adding to capacity where needed or dedicating more resources to the affected network function instances. Operators can also assess the demand patterns, predict future growth, and plan network expansions accordingly. Compared to other networks, the factors such as dynamic topology, heterogeneous devices, varying user behavior, mobility-induced fluctuations, load on the wireless links, fluctuations of signal level and resource constraints on different network components, add complexity to accurately identifying and mitigating bottlenecks in mobile networks [11], [12].

Existing literature shows that ML can be leveraged to automate detection of bottlenecks at run time [13]. To identify the root cause of performance degradation in a mobile network, studies like [14], [15] employed a supervised learning method, but these methods are dependent upon labeled training samples and fail when an unlearned issue arises. Furthermore, for accurate detection and attribution of performance issues, it is essential to base the ML model upon a holistic view of the network system. In other words, the monitoring should reflect the state across different components that inter-connect the end-to-end paths. A holistic

view includes monitoring of both the services at the UEs and the rest of the network infrastructure.

Monitoring the performance at UEs, can be done with crowdsourcing [16], [17], where UEs tracks and reports on Quality-of-Service (QoS) (e.g., network coverage quality, handovers and throughput) and QoE features (such as page load time in web-browsing applications or jitter in streaming videos). Crowdsourcing, however, is expensive in terms of data caps and overhead associated with the periodic reporting of the monitored features to a central monitoring entity [16]. Moreover, with of the ever increasing density of UEs, mobile networks may be burdened by the high volume of measurement data. There is a need to avoid this overhead, without compromising the in-time and accurate detection of a network performance degradation.

As far as the network infrastructure is concerned, passive measurement strategy best suits where the components are operated/controlled by the monitoring entity. It involves recording and analyzing the user traffic to understand network usage trends. In situations where it is not possible to select capture points freely, active probing is used. This method injects test traffic into the network to find faults or issues. Active probes are controllable in terms of when and what network features are to be measured. It, however, burdens the probed devices and links with additional data. Passive measurements do not inject additional data, but monitoring all the traffic flows can be expensive in terms of memory and processing resources. Additionally, network administrators and operators typically utilise a component-to-controller [18] monitoring framework, where, even passive measurements when uploaded to the centralised controller, incur communication overhead.

Unlike traditional passive monitoring strategies [19] and tools such as `tcpdump` [20] and `wireshark` [21], one can apply the Inband Network Telemetry (INT) [22] method used in datacenters [23], [24]. INT is a type of passive monitoring system implemented with Programming Protocol-independent Packet Processors (P4) [25], [26]. Being programmable, it allows a centralised network controller not only to configure the measurement frequency and to change the monitored features on the fly, but also to adjust the monitoring granularity to per-user, per-link, per-flow down to a packet-level.

In the light of the above discussed potentials and challenges, we aim to leverage a 2-stage distributed telemetry architecture with following features:

- Local (or semi-local) learning with minimal monitoring overhead. Unlike crowdsourcing the UE does not send the telemetry data to the central monitoring entity i.e., to the mobile network controller. Instead, anomaly detection on the monitored QoS/QoE metrics of end-services, local to the UE, is performed either on the UE itself or at an edge compute element. Anomaly detection flags both previously seen and unseen performance issues.
- After an anomaly is detected, the data analytics at the central monitoring entity is triggered. The central monitoring entity runs a supervised bottleneck classification model, being built upon periodic measurements from different components and links of the cloudified mobile network. For an unidentified bottleneck instance, the features are logged into a file, to be labelled and used in model retraining.
- The framework involves real time attribution of network issues, that are impacting a significant number of UEs. We therefore aim for a simple ML model that is not only scalable, has quicker response time but can also help to understand the impact of different monitoring methods and parameters on the model's accuracy.

3 System Infrastructure

The hardware and software infrastructure of our mobile network testbed is illustrated in Figure 2.

3.1 Cloudified Mobile Network

The main part of the network [27] is an Enhanced Packet Core (EPC), containing four basic components:

1) Home Subscriber Server (HSS), for managing the network subscriber accounts;
2) Mobility Management Entity (MME) for managing the attachment of Evolved Node Bs (eNodeBs), i.e. base stations, and UEs, e.g. smartphones or modems;
3) Control Plane of the Serving and Packet Data Network Gateway (SPGW-C), for managing access to a Public Data Network (PDN), i.e. the Internet;
4) User Plane of the Serving and Packet Data Network Gateway (SPGW-U), for forwarding user traffic between UEs and the PDN.

The HSS, MME, SPGW-C and SPGW-U are using the open source implementation from OpenAirInterface (OAI) [28]. In addition to the four EPC components, we also deploy FlexRAN [29]. In particular, a FlexRAN Controller to manage the eNodeB parameters and provide fine-granular metrics from the eNodeBs.

The eNodeB is deployed using the open source implementation from OpenAirInterface. In addition, the Software Defined Radio (SDR) Ettus USRP B210 provides both the antennas and the Radio Unit (RU), which converts radio waves into digital waveforms.

Clearly, managing the components of a complex setup manually is not straightforward. Therefore, we deploy Open Source MANO (OSM) [30] as the orchestration platform for Network Function Virtualization (NFV). Basically, OSM performs [31, Chapter 1]:

- Composition of Virtual Network Functions (VNFs) into Network Services (NSs);
- Instantiation of NSs and their VNFs in an underlying Network Function Virtualization Infrastructure (NFVI) as so-called Virtual Deployment Units (VDUs), which are virtual machines and/or containers;
- Run-time configuration (e.g. initial installation, run-time change of parameters, reconfiguration) of the VDUs;
- Monitoring of the VDUs (details in Subsection 3.2);

This article has been accepted for publication in IEEE Transactions on Mobile Computing. 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/TMC.2023.3312051

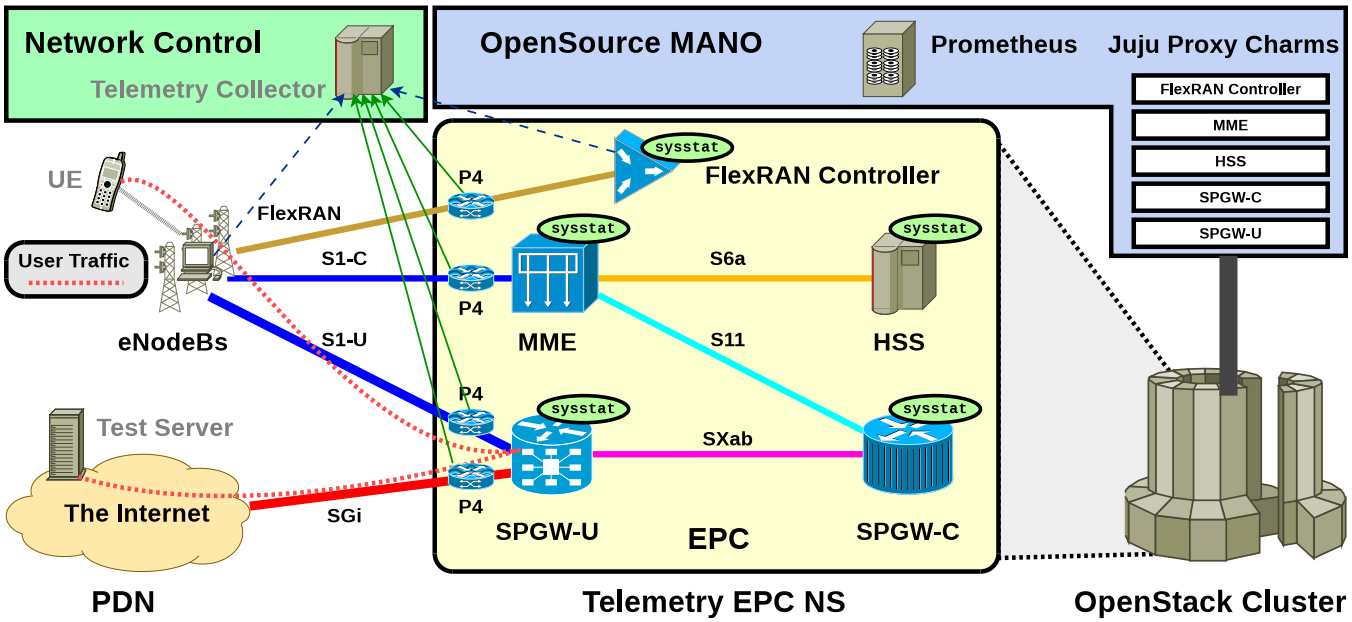

Fig. 2: The testbed Infrastructure of the Cloudified Mobile Network.

- Scaling (i.e. increasing/decreasing the number of instances) and removal of VDUs.

Currently, our setup is running OSM "Release EIGHT" on UBUNTU 18.04 "Bionic Beaver". OSM uses JUJU [32] for managing the VDUs. That is, for each VDU, JUJU maintains a separate container controlling it. Each container runs the JUJU Charm of the corresponding component, which is a custom Python program to implement the component-specific control functionalities. In our setup, we use OPENSTACK [33] "Stein" on UBUNTU 19.04 "Disco Dingo" as NFVI to host the VDUs, which are instantiated as virtual machines in two OPENSTACK compute nodes. The VDUs of MME, HSS, SPGW-C and SPGW-U run UBUNTU 18.04 "Bionic Beaver", while the VDUs of FLEXRAN Controller and P4 switches run UBUNTU 20.04 "Focal Fossa".

3.2 Telemetry Components

As part of the components orchestration, OSM already provides two ways of monitoring the deployed NSs:

1) By using features of the NFVI (i.e. by CEILOMETER and GNOCCHI in OPENSTACK [34]);
2) By JUJU Charms, that run customised monitoring code as part of the configuration service managing the VDUs.

However, this monitoring only covers coarse metrics [35], [36] – like CPU utilisation, per-interface packet and byte counters, etc. – and does not represent the quality of services features of user data traffic. Particularly, there is no information about user flows (e.g. TCP connections, etc.) of users. Packet and byte counters only represent the aggregation of *all* users and their flows. We aim at a vendor-independent, "standardised" solution for passive monitoring of the per-flow user data traffic with P4 version 16 [26] based software switches. P4 provides a standardised language for programming packet processors, i.e. switches, which can be compiled for different target devices. Currently, we deploy P4 software switches, using the Behavioral Model Version 2 (BMv2) Simple Switch software implementation [37]. However, once available, it would be straightforward to just replace them by more powerful, off-the-shelf P4 hardware switches. In our testbed, as shown in Figure 2, we have P4 switches for the four important interfaces (actually: internal networks):

1) S1-C, between eNodeB and MME (network control traffic);
2) S1-U, between eNodeB and SPGW-U (encapsulated user traffic);
3) SGi, between SPGW-U and PDN (decapsulated user traffic);
4) FlexRAN, between eNodeB and the FLEXRAN Controller (only FLEXRAN control traffic).

Particularly, the user traffic is handled on the S1-U interface, where it is tunnelled via GPRS Tunnelling Protocol (GTP), and on the SGi interface, where it is "normal" traffic without encapsulation. It should also be noted that SGi traffic uses the public IP address of an SPGW-U. An SPGW-U performs Network/Port Address Translation (NAT/PAT) between an internal address, used by a UE, and the public SPGW-U address. Inside the tunnel, traffic therefore uses the internal address of a UE.

We programmed the P4 switches, at S1-U and SGi, to attach custom telemetry data to packets running over them. For example, the S1-U switch attaches INT information to a user packet (in the GTP tunnel), and forwards the modified version of the packet to the Telemetry Collector (i.e., a part of the mobile network collector), while the un-modified to its destination (that is test server in Figure 3). The SGi switch does the same before forwarding the packet into the Internet. The Telemetry Collector correlates the two packet snippets (on the flow identifiers that come with the INT fields), and generates performance metrics. Note that the

This article has been accepted for publication in IEEE Transactions on Mobile Computing. 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/TMC.2023.3312051

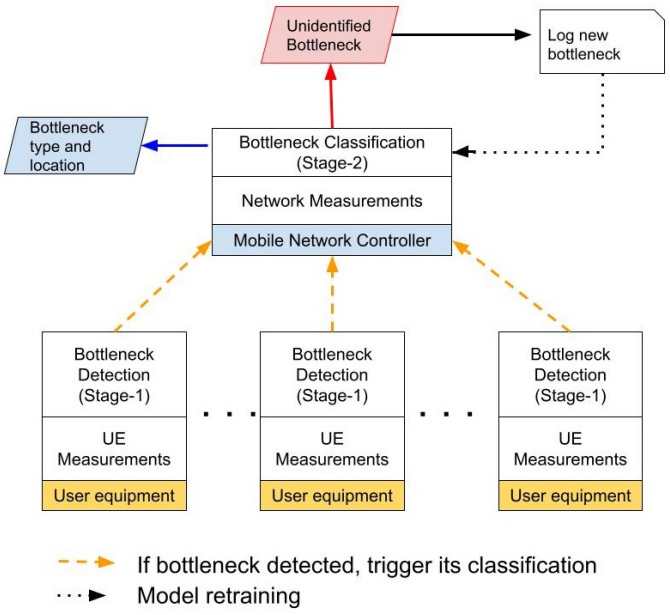

Fig. 3: Bottleneck identification in the 2-level distributed monitoring framework.

actual outgoing packet into the Internet does *not* contain telemetry data i.e., the privacy of the user is not getting compromised.

With the received INT information of the packets, the Telemetry Collector can track both the characteristics of user data traffic and the status of the P4 switches such as congestion at a port. Depending on the processing power of the P4 switches and the Telemetry Collector, such a system can be configured to only handle a subset of the packets or flows, for creating samples (e.g. only every n-th packet or flow or only flows of certain representative users, etc.). Moreover, based on the monitoring needs, INT can be performed on the data traffic with the granularity of per user, per service type, per flow or per packet.

4 Bottleneck Identification System

Figure 3 depicts a high level overview of the distributed framework of our proposed bottleneck identification system. It comprises of three stages:

1) System monitoring
2) Bottleneck detection
3) Bottleneck attribution (or classification)

Each of the three stages are elaborated in the following subsections.

4.1 System Monitoring

In our 2-stage distributed framework, we monitor the communication system both at the user side (i.e. at the UE) and at the rest of the mobile network.

Monitoring at UE: It records QoE-based features of the applications running on the end-device. The monitored features depend upon the specific application. This can be page load time and throughput for a web-browsing service, while for a streaming video it can be delay, jitter and throughput. For our test scenario, we generate downlink TCP traffic with iPerf from the test server as a user data session, the quality features of which, shown in Table 1, are monitored passively at the UE. This data transfer is performed at the maximum bandwidth of the end-to-end path. Other than monitoring performance of the service (i.e, iPerf TCP session) passively, the UE also tracks RTT to the test server by sending UDP ping messages every second. This active measurement monitors latency on the end-to-end path; an additional performance indicator of the end service. Lastly, the UE passively monitors the radio coverage quality via NetMonitor[2], every second. NetMonitor collects Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ) and Received Signal to Noise Ratio (RSSNR).

Monitoring status of eNodeB: The FlexRAN controller provides a northbound RESTful API for issuing control commands and for obtaining statistics and reports for the connected base stations using simple HTTP requests[3]. We run the `curl -X GET http://127.0.0.1:9999/stats/manager/all` command, every 5 seconds. This gets the RAN configuration and status for the current TTI for all eNodeBs connected to this controller. It reports on the configuration of eNodeB(s) and UE(s), and statistics about Medium Access Control (MAC), Radio Link Control (RLC) and Packet Data Convergence Protocol (PDCP) layers.

Monitoring network data links: To monitor data flows, network links and status of switching devices, we utilise the P4 switches of S1-U and SGi interfaces (see Figure 2). The switches create clones of the passing by packets and add an additional header of "IP options" [39, Subsection 3.1] with telemetry fields that comprise both of status of the P4 switches and flow characteristics. The metrics of INT can be programmed depending on the monitoring requirement. These cloned (mirrored) packets are then sent to the Telemetry Collector (shown in Figure 2), for further analysis. For our test scenario we apply INT, only on ping packets, to compute four parameters on each of the two switches. These are (1) *packet count of the data flow* using count-min sketch [40], (2) *hitter* [41] which is a Boolean metric that assesses if the packet is part of a bursty traffic. We consider switch queue size of 5000 or more bytes as an indicator of a bursty traffic, (3) *deq_timedelta* that measures in microseconds the amount of time a packet stays in P4 switch queue, and (4) *deq_qdepth* which indicates the length of the switch queue when the packet was dequeued, in number of packets. The last two parameters are derived from the struct *standard_metadata* of the P4 version 16 V1Model architecture[4]. Along with these key parameters the mirrored packet carries the flow identifier and the switch identifier to

1. Features collected by FlexRAN Controller: https://mosaic5g.io/apidocs/flexran/flexran_spec_v2.2.3.html

2. NetMonitor is an Android app from https://vavsoftware.ru.

3. FlexRAN northbound API: https://mosaic5g.io/apidocs/flexran/#api-Stats-GetStatsHumanReadable.

4. Standard Metadata: https://github.com/p4lang/behavioral-model/blob/main/docs/simple_switch.md

Type	Description	Examples of features
Monitoring at UE		
Active monitoring from UE	Ping to test server	Average RTT, Packet loss percentage
Passive monitoring of an application service	Downlink TCP IPERF from test server	Transfer (MiB), Bitrate (Mbit/s), Jitter(ms)
Passive monitoring of network coverage	NetMonitor App	RSRP, RSRQ, RSSNR
Monitoring status of eNodeB		
Control commands from northbound RESTful API of FlexRAN controller	To obtains configurations and statistics of eNB(s) and their UE(s)	We record features specified by `flex_cell_config`, `flex_ue_config` and `flex_ue_stats_report`[1].
Monitoring of network data links		
Active monitoring of S1-U and SGi links	Ping from eNodeB to SPGW-U	Average RTT, Packet loss percentage
	Ping from SPGW-U to the test server	Average RTT, Packet loss percentage
Passive monitoring by P4 switches at S1-U and SGi	INT on user data traffic and status of the switches	Packet count of a flow, $packet\_length$, deq_qdepth, $deq\_timedelta$, hitter
Monitoring of network resources		
Passive monitoring of resource usage at SPGW-U and FlexRAN controller	Using sar utility of $Sysstat$ to collect CPU and memory usage information, and track disk input output activities	CPU usage features (i.e. %user, %system, %iowait, %steal, %idle), Memory usage features (i.e. kbmemfree, kbavail, kbmemused, permemused, kbbuffers, kbcached, kbcommit, percommit, kbactive, kbinact, kbdirty) and I/O features (such as tps, rtps, wtps, dtps) [38]

TABLE 1: Network-wide monitoring and measurements.

the Telemetry collector.

We also leverage active probes in the network system. We have two such probes, one at the eNodeB and another at the SPGW-U. The first one measures delay and packet loss at the S1-U interface, and the other one at the SGi interface by injecting Ping messages from the eNodeB and the SPGW-U, destined to the SPGW-U and the test server, respectively.

Monitoring network resources: Lastly, we track the resource utilization in the mobile network, by monitoring load on the CPU, memory and I/O disk operations with SYSSTAT. We run SYSSTAT utility[5] [38] periodically, every 5 seconds. In the current architecture, we exploit SYSSTAT parameters collected at the SPGW-U and the FlexRAN controller. These two components reveal impact upon data and control flows, respectively, under stressed and non-stressed resources. The details of the monitored features are given in Table 1.

Other than UE measurements, rest of the network measurements are periodically transmitted to the Telemetry Collector part of the mobile network controller.

4.2 Bottleneck Detection

We formulate the bottleneck detection stage as an anomaly detection problem. We denote the measurements, used to identify bottlenecks, as multivariate time series $T = \{x_1, .., x_T\}$, $x(t) \in \mathbb{R}^m$ is an m-dimensional vector of samples at timestamp t. An anomaly detection method learns a model to label a binary variable $y_t \in \{0, 1\}$ at time t as 1 if anomaly is detected, where anomaly represents a rare or unseen observation $x_t^{'}$.

The unsupervised anomaly detection allows for a more holistic exploration of the data, enabling the identification of various types of bottlenecks, including complex patterns that may not be easily labeled by human operators. The anomaly detection process is assuming that T contains only normal samples and the model is trained to learn the distribution of normal data. An anomalous sample is one that differs significantly from T. The difference between the sample $x_t^{'}$ and the normal data T is measured by an anomaly score, which is then compared to a threshold. If the score is above the threshold, the sample is considered as anomalous.

5. SYSSTAT: https://bencane.com/2012/07/08/sar-sysstat-linux-performance-statistics-with-ease

4.3 Bottleneck Attribution

Once a bottleneck has been detected by the stage-1, we need to specify the type and location of this bottleneck. Note, in the wild, our framework will only trigger stage-2 or bottleneck attribution when multiple UEs report it. It is to avoid responding on quality degradation caused by a context specific to a single UE, e.g., its end-device defect. For this study we do not follow this restriction due to our simplistic experimental testbed.

We take bottleneck attribution as a classification problem and formulate it as a supervised learning model, which is trained with known class labels. More specifically, we define 10 classes of *single* bottlenecks (See Table 2). These are bottlenecks that have a single source of occurrence e.g., congestion only at S1-U. Besides the single bottlenecks, the classification model should also be able to classify the *composite* bottlenecks, that has more than one sources e.g., data congestion at S1-U and stress on network resources. Additionally, the model should identify any bottleneck that is not experienced before as *unidentified*, instead of misclassifying it. An *unidentified* bottleneck is registered in a log file (as depicted by Figure 3) along with its corresponding measurements features. If its occurrences increase, it can be labelled and used for retraining the classification model.

5 SYSTEM IMPLEMENTATION

5.1 Types of Bottlenecks

To represent occurrences of different types of performance issues in the mobile network, we follow the bottleneck profiles of [4]:

1) generate congestion on network data paths,

2) introduce packet loss at different intensities in the network,
3) overload network resources at different intensities, and
4) create interference at the radio access link.

Table 2 provides the complete list of bottlenecks that we test on our system architecture. To emulate **congestion**, we introduce an additional downlink TCP traffic flow at the maximum bandwidth of the network link(s), using the IPERF [42] tool. Next, for **packet loss** induction, we use the Linux traffic control feature NETEM [43], [44]. These latter experiments consist of either a high loss percentage of around 5% or a low loss percentage of 1% and adversely effect the passing by data flows. Thirdly, to **overload the network resources**, we stress out the CPU and memory resources and increase input/output disk operations with the *stress-ng* tool [45][6]. Lastly, to create **radio interference** we deploy a GNU RADIO[7] noise source on a separate system using a dedicated SDR ETTUS USRP B210. The noise source generates an additive white Gaussian noise (AWGN) signal which has central frequency similar to that of the eNodeB radio carrier.

The above bottlenecks are generated at different network links and components, making it up to fourteen different bottleneck profiles. In Table 2, the source link/component of each bottleneck is mentioned in *Location* column. In terms of ML, each bottleneck profile represents a class label. Further to it, the bottleneck profiles are categorised into *single* and *composite* groups depending on their complexity.

5.2 Bottleneck Detection and Attribution Models

As discussed earlier, in our bottleneck identification framework, the detection stage comes first. Taking bottleneck as an anomaly, our investigation shows that some of the following widely used unsupervised anomaly detection techniques result in excellent performance.

1) Isolation Forest (iForest): iForest algorithm [46] is based on decision trees to separate outliers from the rest of the data. It is widely used for anomaly detection [47] [48]. Recursively, it partitions the data by randomly choosing a feature and then selecting a random split value, i.e. cut-off-point between the max and min values of that feature. The algorithm then determines if this isolates an anomalous measurement sample; if so, it stops; otherwise, it selects a different feature and a different cut-off point at random. The anomalous measurement samples are distinguished from the rest of the measurement data by this random splitting of features, which will result in shorter routes in trees. The measurement samples that travel deeper into the tree are less likely to be anomalies as they required more cut-off-points to isolate them.
2) Autoencoder (AE) [49]: is an unsupervised artificial neural network composed of an encoder and a decoder. The encoder (in Equation 1) takes the input x and maps it into latent variable z, whereas the decoder maps the latent variable z back into the input space as a reconstruction $\hat{x}$ (Equation 2). W and b are the weight and bias of the neural network and σ is the nonlinear transformation function.

$$z = \sigma(W_{xz}x + b_{xz}) \tag{1}$$

$$\hat{x} = \sigma(W_{zx}z + b_{zx}) \tag{2}$$

The difference between the original input vector x and the reconstruction $\hat{x}$ is the reconstruction error as in Equation 3. An autoencoder learns to minimize this reconstruction error (*loss*).

$$loss = ||x - \hat{x}|| \tag{3}$$

The *loss* is considered to be an anomaly score, which if above a predefined threshold, depicts an anomalous data input.
3) Variational Autoencoder (VAE) [50]: To avoid the over-fitting that may result from decoding the latent space z without any reconstruction loss, we use VAE i.e., an extended versions of AE. Instead of encoding an input as a single point, VAE encodes input as a distribution over z. A sample point from this distribution is then decoded and the reconstruction error can be computed. Thus the encoders and decoders of VAE are called probabilistic encoders and decoders. Besides the reconstruction error, the loss function of VAE has to regularise the latent variable z that can be done using Kulback-Leibler divergence (KL). The loss function can be expressed in Equation 4, where μ, λ are the mean and covariance of the distribution and N is the Gaussian distribution.

$$loss = ||x - \hat{x}|| + KL[N(\mu_x, \lambda_x), N(0, 1)] \tag{4}$$

After training, VAE reconstructs normal data very well, while failing to do so with anomalous data which the VAE has not encountered.VAE uses the reconstruction error as the anomaly score.
4) Denoising Autoencoder (DAE) [51]: Again an extension of AE, DAE receives a corrupted data point $\hat{x}$ as input by adding random noise to the original input x. The DAE is then trained to recover the original uncorrupted data point x as its output.

After detection of a bottleneck anomaly, we aim for the second stage i.e., a ML classification model to attribute the type and location of the bottleneck. We, therefore need our *bottleneck classification stage* to tackle three different types of classification problems. These include:

1) Multi-class, to predict one of the 10 *single* bottleneck classes mentioned in Table 2;
2) Multi-label classification, for predicting *composite* bottlenecks mentioned in Table 2;

6. For high stress we use: `stress-ng --cpu 4 -d 1 --hdd-bytes 1G -m 1 --vm-bytes 1G --iomix 1 --iomix-bytes 1G`.
For low stress, we use: `stress-ng --cpu 1 -d 1 --hdd-bytes 256M -m 1 --vm-bytes 256M --iomix 1 --iomix-bytes 256M`.
7. GNU RADIO: https://www.gnuradio.org.

Bottleneck Profile/Class Label		Bottleneck Cause	Complexity
Type	*Location*		
Congestion	S1-U	Downlink TCP iPERF at maximum available bandwidth	Single
Congestion	SGi	Downlink TCP iperf at maximum available bandwidth	
Congestion	S1-U and SGi	Downlink TCP iperf at maximum available bandwidth	
High stress on resources	SPGW-U	High CPU, Memory and Disk I/O stress	
High Stress on Resources	FlexRAN Controller	High CPU, Memory and Disk I/O stress	
Low packet loss	SPGW-U	1% packet loss	
High packet loss	SPGW-U	5% packet loss	
Radio interference	Radio access link	Radio frequency interference caused by a transmitter on the same frequency at which the test UE received data	
Low resource stress	SPGW-U	Low CPU, Memory and Disk I/O stress	
Low resource stress	FlexRAN Controller	Low CPU, Memory and Disk I/O stress	
High stress on resources	SPGW-U and FlexRAN Controller	High CPU, Memory and Disk I/O stress	Composite
Congestion, High resource stress	S1-U, FlexRAN Controller	Downlink TCP iPERF at maximum available bandwidth, High CPU, Memory and Disk I/O stress	
Congestion, High packet loss	S1-U, SPGW-U	Downlink TCP iPERF at maximum available bandwidth, 5% packet loss	
High packet loss, High resource stress	SPGW-U	5% packet loss, High CPU, Memory and Disk I/O stress	

TABLE 2: Bottleneck profiles analysed on the testbed.

3) *Unidentified* classes; to predict occurrences of new/unseen types of bottlenecks. These will include the bottlenecks profiles, upon which the ML model is not trained.

Neural Networks, as deep learning models, have shown promising results in different classification tasks [52]. Architectures such as MLP, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) can be used for multi-class and multi-label classification by modifying the output layer to accommodate multiple labels. Among these, CNN is best suited for image classification while RNN needs long temporal pattern as input. To be able to tackle the bottleneck classification problem, we therefore select a MLP model.

An MLP is a neural network with fully connected neurons among layers. It has the capabilities to approximate, through supervised learning, the function that relates the input with the output. It offers flexibility in terms of its architecture, for example by customizing its number of layers, the number of nodes in each layer, and the activation functions, MLP can both classify multi-label instances and isolate the unseen instances. Although MLP is considered a black-box model, its architecture allows for some level of interpretability. We can analyze the weights and activations in the hidden layers to gain insights into the learned representations and understand how the model makes decisions.

Our MLP classifier has an input layer that expects 76 inputs and an output layer that matches with the number of *single* bottleneck classes. Each node in the output layer has a sigmoid activation, which predicts a probability of class membership for the label, a value between 0 and 1. This means, the MLP classifier will predict 10 probabilities for each input sample. The output probability indicates the confidence of the classifier in its predictions. An activation threshold of 0.5 is then used to convert the probabilities generated by the MLP into binary predictions. If the predicted probability for a label is ≥ 0.5, it is considered as positive (1), and if it is below 0.5, it is considered as negative (0). Based on this we can identify all types of bottlenecks. For instance, an input sample with the output [0.1, **0.98**, 0.2, 0.4, 0.09, 0.08, 0.3, 0.1, 0.03, 0.12] most likely belongs to the second bottleneck class. For the composite bottleneck, more than one class will have high probability ≥ 0.5 threshold. This way, any sample that does not belong to any of the bottleneck classes can me marked as *unidentified*.

6 Evaluation

6.1 Dataset

To evaluate our proposed system, we exploit three different datasets based on the design choices described above.

6.1.1 UE-based Dataset

It comprises the measurements collected at the UE (see Table 1). In the UE dataset, 8 features are measured directly at the UE namely, *Average RTT*, *Packet loss percentage*, *Transfer (MiB)*, *Bitrate (Mbit/s)*, *Jitter (ms)*, *RSRP*, *RSRQ*, and *RSSNR*. *Inter packet gap (IPG)* is a metric computed from the monitored feature of *Average RTT*. Our monitoring frequency is every second, but for data analytics we take mean values of the features in each 5 seconds window. For each of these windows we also compute other statistical metrics including median, skewness and kurtosis of all the primary features given above, except of the radio quality indicators. The resulting UE dataset consists of 28 features that we use for bottleneck detection. Within this dataset 8,640 data samples were collected under normal network conditions, which form a *baseline* scenario in our monitoring. In the context of this study a data sample denotes set of features that were derived from a single 5 seconds window. In total, 40,320 data samples were collected when various bottlenecks were emulated in the network.

6.1.2 Mobile Network Dataset

The Mobile Network Dataset contains the measurements collected from the components of mobile network, namely, eNodeB, FlexRAN, SPGW-U and P4 switches. These measurements, shown in Table 1, are reported to the Telemetry Collector (Figure 2), which is a part of Network Controller (Figure 3). Pre-processing steps are applied to the

measurements before feeding into the bottleneck identification model. Initially, we filter out all the missing values and constant features. In particular, we remove all configuration and identifier-related variables such as *cell_config.init_nr*, *cell_config.phy_cell_id*, *ue_config.rnti* and *imsi* from the FlexRAN measurements. Secondly we retain only one of the correlated features from a single monitoring point, for example, retaining *kbmemused* and removing its correlated feature of *kbavail* from memory usage features reported by *Sysstat* at SPGW-U.

The switches at S1-U and SGi perform INT and mirrors the modified packets to the Telemetry Controller, which not only extracts the INT features defined in the Table 1, but also derives metrics i.e., percentage of lost packets and uplink & downlink jitter between corresponding INT packets recieved from the two P4 switches. All mobile network-based measurements make it to 54 different features. For the measurements that we collect frequently, that is every second, their statistical metrics including mean, median, kurtosis and skewness are computed for every 5 seconds window. This increases the number of features to 76. Just like the UE dataset, the final dataset from the mobile network has 40,320 data samples.

6.1.3 Network-wide Dataset

It combines all features from both the UE-based dataset and the network dataset. It is used to investigate the trade-off between a *centralised* and the *distributed* monitoring and analysis frameworks.

We divided the above datasets into two subsets of ratio $60 : 40$ for training and testing sets respectively. During training, 30% of the training data is held for validation.

6.2 Evaluation Metrics

To evaluate the performance of our bottleneck identification framework, we use prediction (P), recall (R) and F1-score (F1) [53]:

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}, \quad F1 = 2 * \frac{P * R}{P + R},$$

where TP is True Positives, FP is False Positives and FN is False Negatives.

We formulate the *composite* bottlenecks as multi-label classes, and use the following metrics to evaluate their classification accuracy:

- Hamming Loss: It is equal to the number of incorrect predicted labels (TNIP) of the individual classes divided by the total number of predictions (TNP). In hamming loss the smaller the result, the better is the model [54].

  $$Hamming\,Loss = \frac{TNIP}{TNP}$$

  In other words if a composite bottleneck comprises of c number of bottlenecks and there are n instances of the composite bottleneck, then $TNP = n \times c$. $TNIP$ denotes the number of single bottlenecks, within $n \times c$ bottlenecks being classified incorrectly.
- Exact Match Ratio (EMR): is the most strict metric, indicating the percentage of samples that have all their labels classified correctly. In our case, the two single bottlenecks that form the composite have to be classified correctly in order to be considered as TP. The disadvantage of this measure is that multi-class classification problems have a chance of being partially correct, but here we ignore those partially correct matches [55].

For bottleneck detection, we define the threshold for the anomaly score that provides the best F1-score.

6.3 Models Implementation

We implement the MLP model using three hidden layers with 64, 24, 16 neurons respectively (these are chosen with trial and error method). We use the rectified linear unit (ReLU) as an activation function in the hidden layers which converges very quickly during the training. The input data propagates through the MLP layers, where each layer performs a matrix multiplication followed by an activation function. The computational complexity of a MLP model depends on the number of layers, the number of neurons in each layer, and the dimensionality of the input and output [56]. For an MLP model with L layers, N_i neurons at layer i, input dimension N_{in}, and output dimension N_{out}, the computational complexity is roughly $O(N_{in} \times N_1 + N_1 \times N_2 + ... + N_L \times N_{out})$. So, the approximate computational complexity of our MLP model with $N_{in} = 76$, $N_{out} = 10$, three hidden layers and different numbers of neurons per layer (i.e., $N_1 = 64, N_2 = 24, N_3 = 16$) would be $O(N_{in}^2)$.

In our implementation for AE, VAE and DAE, the encoder and decoder both have two hidden layers with 28, 14 dimensions at the first and second hidden layer, respectively. Each layer has N = 64 neurons. The computational complexity of the encoder can be approximated as $O(N^2)$ for each layer. Since there are two hidden layers, the overall complexity of the encoder would be proportional to $O(2 \times N^2)$. Similarly, the computational complexity of the decoder is also proportional to $O(2 \times N^2)$. Both encoder and decoder approximate it to $O(N^2)$. In short, both the bottleneck detection and the classification ML models make it to a quadratic complexity.

Table 3 details the hyper-parameter setup used for each model. The hyper-parameters are estimated using cross-validation. Where a parameter is not specified, it indicates that it is set by its default value. The MLP, AE, VAE, and DAE models are implemented using Pytorch[8] whereas iForest is implemented with scikit-learn 1.2.2[9]. All these models are trained on NVIDIA V100 GPU 32GB.

6.4 Performance Analysis of the Bottleneck Identification Framework

For the *bottleneck detection stage*, of our proposed bottleneck identification system, we investigate the different methods listed in Subsection 5.2, using UE-based dataset. Table 4 shows that AE-based methods demonstrate superior precision compared with the iForest model. However, VAE outperforms all the other methods by up to 26% F1-score.

8. https://pytorch.org/
9. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

TABLE 3: Hyper-parameter settings of the different methods.

Method	Parameters	Use
iForest	The number of base estimators=600	Bottleneck Detection
AE	window_size=20, Epochs=100, Learning rate= 0.002	
DAE	window_size=20, Epochs=100, Learning rate= 0.001, Noise= Gaussian noise $\mathcal{N}(0, 0.1)$	
VAE	window_size=20, Epochs=100, Learning rate= 0.001	
MLP	Epochs=200, Optimizer= Adam Optimizer [57], Learning rate=0.001	Bottleneck Classification

The better performance of VAE is due to its learning the underlying distribution of normal data. In contrast, DAE and traditional AE focus more on identifying anomalies based on deviations from normal patterns without explicitly modeling the underlying distribution. iForest, on the other hand, do not explicitly learn a latent representation and rely on proximity measures to partition the data, limiting its ability to capture complex patterns. We, therefore, proceed with VAE in our further experimentation.

We train the VAE model with baseline measurements and test it using a mix of baseline and bottleneck measurements. The baseline measurements are captured when data traffic does not co-exist with a bottleneck.

TABLE 4: Performance comparison between different anomaly (i.e., bottleneck) detection methods using UE dataset.

Method	Precision	Recall	F1-Score
iForest	0.621	0.613	0.639
AE	0.821	0.51	0.63
DAE	0.85	0.82	0.83
VAE	0.90	0.81	0.85

Table 5 provides comparative bottleneck identification results both for the *distributed* and *centralised* monitoring and analysis frameworks. For *distributed*, it separately evaluates the worth of the two measurement sets i.e., one collected at UE and other within the mobile network. The table shows that by using only measurements collected at UE, VAE achieves an F1-score of 0.85 to detect different types of bottlenecks collectively. When compared to the F1-score of the model built upon mobile network-based measurements, it is 4% lower. Reason of which is the low intensity bottlenecks such as low packet loss and low stress (SPGW-U and FlexRAN) that have negligible impact on the end-users QoS and QoE features, as are depicted in Figure 4. The UE measurements of these two bottlenecks have similar distributions to that of the baseline, making them go undetected most of the time. Although the VAE model built upon only mobile network-based measurements succeeds in identifying the bottlenecks that directly affect the mobile network, it fails when the source of performance degradation lies on the last mile such as in case of radio interference and. In our experiments, the majority of the bottlenecks originate from the mobile network infrastructure which results in high bottleneck detection accuracy using the mobile network based measurements.

TABLE 5: Performance of our proposed bottleneck identification system using different types of measurements.

System Architecture	Dataset	Bottleneck detection			Bottleneck classification		
		P	R	F1	P	R	F1
Distributed	UE-based	0.90	0.81	0.85	0.62	0.54	0.13
	Mobile network-based	0.92	0.87	0.89	0.90	0.88	0.89
Centralised	Network-wide	0.94	0.89	0.91	0.95	0.91	0.93

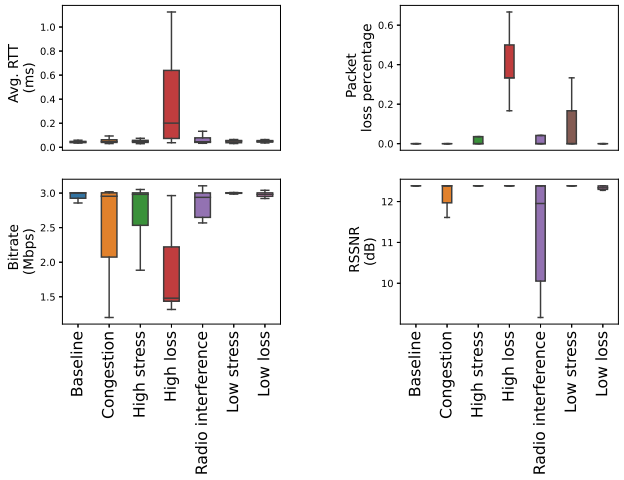

Fig. 4: Distribution of the most important UE features collected under different test scenarios.

Our distributed framework achieves 0.89 F1-score in the classification of the bottlenecks that are identified by the UE as anomalies. The results of our distributed architecture are highlighted by `cyan` color in Table 5. When compared to the *centralised* framework, ours lag behind by just 2% in detection and 4% in classification accuracy, respectively. This slight accuracy lag is out-weighted by the scalability and substantial reduction in overhead by the 2-stage *distributed* monitoring and analysis framework. In other words, the distributed framework is a feasible and better choice as:

- It relieves network from the overhead of transmitting monitored features from a UE to the Telemetry Collector, where in our experimental scenario, a single UE collects 13% of total network-wide features.
- It is scalable, since the the monitoring and analysis load of the mobile network does not increase with increase in number of UEs. Each UE performs bottleneck detection on either its own end or at its mobile edge.
- It triggers bottleneck classification at Telemetry Collector only when an anomaly is reported by a substantial number of connected UEs. This process relieves the network controller from the classification exercise until QoS/QoE of its end devices does not deteriorate.

Takeaways: Leveraging measurements at UE can help in detecting bottlenecks that arise from issues in the last mile as well as rest of end-to-end path. On one hand, UE based

TABLE 6: Performance evaluation of single bottleneck classification.

Single Bottlenecks			
Bottleneck class	**P**	**R**	**F1**
S1-U congestion	0.94	0.88	0.91
SGi congestion	0.96	0.90	0.88
S1-U and SGi cong.	0.90	0.86	0.85
High SPGW-U stress	0.97	0.88	0.93
Low SPGW-U stress	0.87	0.84	0.85
High FlexRAN stress	0.96	0.90	0.93
Low FlexRAN stress	0.85	0.82	0.83
Low SPGW-U loss	0.85	0.83	0.84
High SPGW-U loss	0.86	0.88	0.87
Radio interference	0.89	0.81	0.84
Unidentified	0.91	0.82	0.85

TABLE 7: Performance evaluation of composite bottleneck classification.

Composite Bottlenecks				
Bottleneck classes	**P_{EMR}**	**R_{EMR}**	**$F1_{EMR}$**	**Hamming loss**
SPGW-U stress, FlexRAN stress	0.84	0.83	0.83	0.27
S1-U congestion, FlexRAN stress	0.82	0.79	0.80	0.23
S1-U congestion, High SPGW-U loss	0.80	0.77	0.78	0.20
SPGW-U stress, High SPGW-U loss	0.77	0.71	0.73	0.25

triggering of an anomaly, relieves the mobile network from continuous data analytics and on the other hand it helps the system identify the bottlenecks that are out of cloudified mobile network domain such as radio interference. Low intensity bottlenecks at the mobile network are hardly detectable by the UE, as they have negligible impact on end-user services. We, however, argue that unless status of a network does not deteriorate user experience, it should not be considered a bottleneck.

6.5 Dissecting Classification Accuracy of Bottlenecks

Here, we dig deeper into understanding the performance of our system in classifying *single* and *composite* bottlenecks based on mobile network-based measurements. We also evaluate the effectiveness of our classification model in dealing with bottlenecks unknown to the model i.e., *unidentified* bottlenecks.

6.5.1 Single and Composite Bottlenecks

Table 6 presents the evaluation results of our classifier per bottleneck type for *single* bottlenecks. Our model is able to classify the defined bottlenecks with high accuracy; F1-score is above 0.84 and in some types even above 0.9. A closer look reveals that the model predicts a considerable number of the bottleneck events caused by low loss at SPGW-U as caused by high loss bottleneck events at SPGW-U and vice versa. The same applies for low and high stress at SPGW-U and FlexRAN. The good point, however, is that the reason of the bottleneck is correctly localised. Bottleneck generated by radio interference is mainly identified by UE measurements, more specifically *RSSNR* metric, as is shown by Figure 4. Features collected by FlexRAN Controller about eNodeB and its UE such as *pdcp_stats.pkt_tx_bytes* and *.wb_cqi* too have partial contribution in identifying bottlenecks caused by the interference.

Next we evaluate the performance of *composite* bottlenecks classification. These results are summarised in Table 7 using the EMR method and the hamming loss. The hamming loss shows that our model is able to detect high proportion of the single bottlenecks within composite bottlenecks. For example, in the case of <S1-U congestion, High SPGW-U loss> the hamming loss is 0.2 which means if there are 100 samples for composite bottlenecks, our model will predict correctly about 80% of the individual bottlenecks that form the composite bottleneck. Unlike hamming loss, EMR marks a prediction to be correct only when both classes within a composite bottleneck are labeled correctly. Based on EMR metric we can observe that the performance of the model in detecting composite bottleneck in the same location is marginally worse than the case of bottlenecks from different locations. For instance, <SPGW-U stress, High SPGW-U loss> exhibits lower $F1_{EMR}$ which means the model can hardly identify the composite bottleneck.

Although, our classifier shows good performance in attributing *single* bottlenecks, the performance degrades in *composite* bottleneck profiles. Reasons of which are:

- We assume that there is no correlation between the single bottlenecks that form a composite bottleneck; but in terms of cloudified mobile network stress generated at one virtualised component impacts the rest of the network, at different intensities. This phenomena leads to 10% drop in accuracy of the *composite* <SPGW-U stress, FlexRAN stress> bottleneck, compared to the corresponding *single* bottlenecks (see Table 6).
- Two single bottlenecks, emerging from same location such as in the case of <SPGW-U stress, High SPGW-U loss> increases attribution error. The model is classifying it as a single bottleneck with <SPGW-U stress> only. The reason of this miss-classification is that the set of features that are impacted by the resource-stress is super-set of the feature set affected by loss, at SPGW-U.

Takeaways: Our bottleneck classification model achieves high performance of above 0.83 up to 0.93 F1-score in the different types of single bottlenecks. The False negatives are mainly due to confusion between low and high intensity bottlenecks, in addition to bottlenecks caused by the interference. In case of the composite bottlenecks, the model can classify them fairly well if the individual bottlenecks are introduced in different network components, but when introduced at the same location, the model accuracy drops.

6.5.2 Unidentified Bottlenecks

To investigate the behaviour of our model with an unseen bottleneck type, we train our model using a dataset labelled with 9 types of *single* bottlenecks from Table 2. For evaluation we apply the trained model on the single bottleneck type that is absent in the training dataset. We refer to bottleneck type as *unidentified* when it does not exist in the training dataset. Table 6 shows the performance of

This article has been accepted for publication in IEEE Transactions on Mobile Computing. 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/TMC.2023.3312051

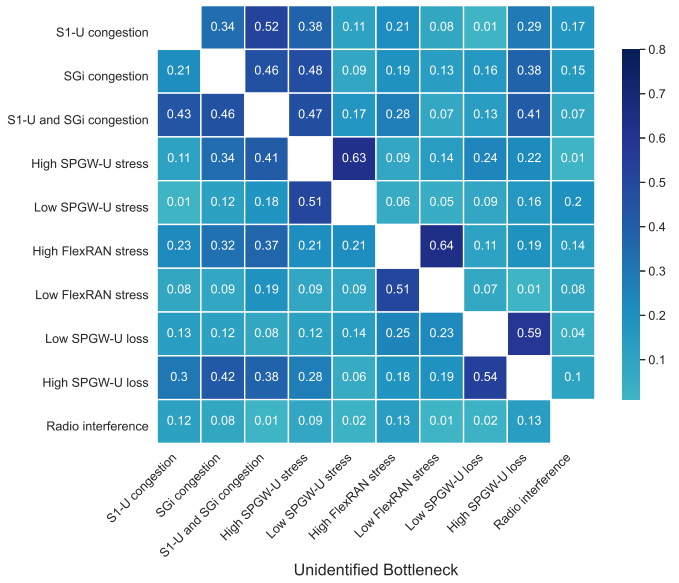

Fig. 5: Average probability of each bottleneck classes computed by the classification model in case of *unidentified* bottlenecks.

our classification model in case it experiences an *unidentified* bottleneck. The resulted performance metrics are averaged from 10 experiments where we test 10 different *single* bottlenecks, one at a time, as being *unidentified*. Interestingly, the F1-score is similar to the *identified* bottleneck types i.e., 0.85. It is 2% higher than F1-score of one of the *single* bottleneck types (see Table 6).

Figure 5 depicts the average probabilities of classifying *unidentified* bottlenecks per labelled bottlenecks. For example, if the *unidentified* bottleneck is *S1-U congestion*, our model predicts it as *SGi congestion* with a probability of 0.21 which is rounded to 0. In case of all probabilities below 0.5, the bottleneck is marked as *unidentified*. Bottlenecks such as *S1-U congestion*, *SGi congestion* and *radio interference* are classified as *unidentified* bottlenecks with high accuracy ($\approx 98\%$) if they are not introduced into our model during training. On the other hand, the model can identify the type and the location of the bottleneck if it experiences the same bottleneck during training however with different severity. For example, if the model is trained with high stress at SPGW-U and tested against low stress at SPGW-U, our model classifies it as high stress with a confidence of 63%. This also applies to other bottlenecks with different severity such as stress at FlexRAN and loss at SPGW-U.

Takeaways: When the mobile network experiences new type of bottleneck that is not known before, our model can correctly classify such bottlenecks as *unidentified* with an F1-score of 85% (see Table 6). The miss-classification by the model is due to its lack in differentiation between the bottlenecks that have same cause but with varying intensity.

6.6 Comparison with state-of-the-art methods

In Table 8, we compare our proposed framework against two recent works from state-of-the-art [7], [4]. We choose these methods because they show some similarities to our work. The work presented in [7] leverages DAE and Convolutional Autoencoder (CAE) for bottleneck detection in a cloudified mobile core testbed. The authors did only active and passive monitoring and did not collect measurements from UEs. In our work, with UE-based measurements and VAE, we can detect bottlenecks both at RAN and at core. The accuracy is slightly better than that of DAE (see Table 4), that is a preferred detection model of [7]. Other than using DAE, [7] examines only four types of high severity bottlenecks at the core network and does not consider bottlenecks occurring at RAN. Also, they focused on detecting a single bottleneck, whereas in real network deployment there might be several simultaneous bottlenecks at different parts of the networks.

G. Patouna et al.[4] on the other hand worked on bottleneck attribution. They preferred an unsupervised machine learning model for identifying bottlenecks across different locations of the network and layers of the system architecture. Similar to our work, they defined different bottleneck profiles, both *single* and *composite*. Hierarchical clustering with number of clusters k equal to number of bottleneck profiles was used by [4] to identify *single* bottlenecks.

By using mobile network dataset, the hierarchical clustering (with k=10) shows a performance of 74.8% F1-score, which is 17% lower than our MLP classifier for *single* bottlenecks. Furthermore, the hierarchical clustering doesn't work well in the case of *composite* bottlenecks, therefore, the authors of [4] leveraged fuzzy clustering model to identify *composite* bottlenecks. Fuzzy clustering achieves F1_EMR of 66.2% that is 18.3% lower than that of MLP, in the case of the composite bottlenecks. The advantage of our work extends to have a single model that works fairly well for both *single* and *composite* bottlenecks. Moreover, our model has the ability to identify unknown bottlenecks.

TABLE 8: Comparison between our and two most related works from literature.

Work	Bottleneck		ML-based method	Performance	
	D	A		F1-Score	F1 EMR
[7]	✓	✗	DAE, CAE	83%	-
[4]	✗	✓	Hierarchical clustering	74.8%(S)	-
			Fuzzy clustering	-	66.2%(C)
Our work	✓	✓	VAE	85%	-
			MLP	90%(S)	81%(C)

[A] Attribution.
[D] Detection.
(S) Attribution of *single* bottlenecks.
(C) Attribution of *composite* bottlenecks.

6.7 Potential of P4-based INT Monitoring

In our telemetry framework, the impact of issues on data links i.e., S1-U and SGi interfaces are monitored in two ways that is by active monitoring and by P4 based INT monitoring. Since active monitoring injects additional load on the network links by interfering with the user data, we investigate if P4 based telemetry can replace active monitoring in capturing bottlenecks on the data links. Figure 6 depicts separate contribution of these two types of measurements along with other measurements, from the mobile network, in classifying the bottlenecks that emit

This article has been accepted for publication in IEEE Transactions on Mobile Computing. 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/TMC.2023.3312051

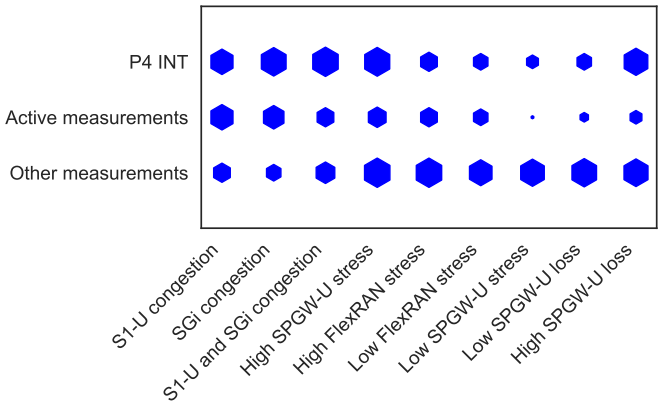

Fig. 6: Contribution of different types of mobile network measurements in identifying issues at this domain. Measurements contribution is calculated based on the features importance using SHAP framework [58]. A higher contribution of measurements is represented by a larger hexagon.

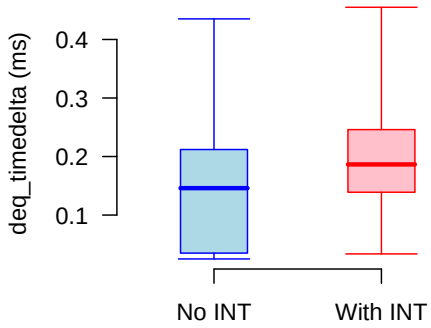

Fig. 7: Impact on queuing delay of data packets when a virtual P4 switch performs in-band network telemetry in a mobile network.

from the mobile network domain. 'Other' measurements here represent the telemetry features collected by resource monitoring and FlexRAN controller.

The figure shows that P4-based INT plays significant role in classifying bottlenecks caused by congestion as well as by high packet loss and stress at SPGW-U. The features from active measurements too, help in attributing these bottlenecks but with much less impact. As expected, for the rest of the bottlenecks including stress at resources, 'other' measurements have higher classification power. We argue, that the P4-switch lying on the eNodeB to FlexRAN controller link (see Figure 3) can catch the high stress in FlexRAN controller, if it is programmed for telemetry collection.

Takeaways: The above comparative analysis depicts that P4 based INT monitoring has the potential to replace active monitoring within mobile network. It assists in classifying the bottlenecks that are introduced by stress, congestion or packet loss at connected components and data links, respectively.

6.8 System Overhead

Telemetry comes with processing, memory and bandwidth cost of different intensities. Passive monitoring induces bandwidth cost, only, when measurements are sent to a central entity for data analytics. Compared to passive monitoring, active monitoring places additional overhead on network links by interfering with the actual user traffic. In our tested scenarios, active monitoring increases the overhead on network data links by 6.2%. As for passive monitoring of the resource utilization and its reporting towards the Telemetry Collector of our cloudified mobile network testbed, it has negligible impact on the VMs memory and interfaces. Same is the case with measurement features of eNodeB and its UE configuration and other characteristics at FlexRAN controller. The reason of negligible impact of our passive monitoring is that we do not apply it for tracking data flows, in the mobile network. According to [59], when passive monitoring is performed on data flows, it not only incurs significant disk I/O operations but also reduces end-to-end throughput by 22%.

We track status of data flows and switches with P4-based INT. We notice that INT-based monitoring increases the processing overhead of the software switches by 48%, which is inline to the findings of [59]. In Figure 7, we illustrate the impact of the P4-based INT on the queuing delay of user data packets i.e., $deq\_timedelta$. When the switches are programmed to monitor packet count of each incoming flow and identify if a packet is part of a bursty traffic, the median $deq\_timedelta$ of packets increases by 27%. A BMv2 software switch suffers from performance inconsistencies. In Figure 7 we, therefore, present $<0.05, 0.25, 0.5, 0.75, 0.95>$ percentiles of the observed *deq_timedelta* for both the `No INT` and `With INT` P4-programs. Other than inconsistency, latency with a BMv2 is significantly higher than that of a production-grade software switch like Open vSwitch[10] or hardware P4 switch, that we intend to replace BMv2 switch with.

To reduce the processing burden on the P4 switches and on the Telemetry Collector, we programmed our P4 switches to create cloned packets with INT information only on the end-to-end Ping messages. Furthermore, one can reduce the processing overhead and delay incurred by programmable switches by only adding needed INT to the packets and at a minimum acceptable frequency.

As for the overhead of data analytics methods, the training time for the bottleneck detection model can be measured by the average time taken per epoch on UE dataset. Off-line training for VAE requires 3 minutes per epoch to converge in 100 epochs. MLP training for bottlenecks classification requires 2.3 minutes per epoch (we train it for 200 epochs). Once trained both can perform inference in less than 2 ms.

10. https://www.openvswitch.org/

7 Discussion

In this study we have presented a monitoring framework for cloudified mobile networks. Our testbed is simplistic, as our aim is to investigate the proposed framework's feasibility, accuracy and overhead in comparison with a traditional centralised monitoring system. There are certain aspects of this study that have nuances which require further exploration. These include:

1) How will the distributed monitoring framework behave in-case some of the UEs either do not participate or have no compute resources to perform anomaly detection on QoS/QoE features of their end-service(s)?
2) Once an anomaly is detected, how will a UE report it to the central telemetry collector/analyser?
3) What will be the monitoring frequency, before and after an anomaly is reported to the mobile network?

As a solution to the first question, if a set of UEs lack the ability to detect an anomaly, the network coverage measurements and monitoring features of user applications can be off-loaded to an edge computing device for data analytics. In case a set of UEs do not participate at all, then the framework will depend upon the UEs that do contribute. Since the bottleneck attribution by the mobile network only works for the issues that affect multiple UEs, non-involvement of a subset of UEs will not hinder the functionality of the framework. Other than an anomaly that is affecting only a single UE or very few UEs on their last mile, the network wide bottlenecks will not go undetected. We argue that like crowdsourced solutions such as [60], monetary or gaming incentives such as in [61] can be exploited to encourage end-users to participate in reporting anomalies, in the performance of their end-services.

For the second challenge, we recommend a P4-INT like solution for the UEs to report about the occurrence of an anomaly. For example, any currently un-used or rarely used field of the protocol header(s) in uplink data transfer can be exploited to flag the detection of an anomaly. It can be as simple as a flip of a single bit; once the data packet arrives at a P4 switch, with P4-INT based telemetry the switch can notify the Telemetry Collector, which can then proceed to perform bottleneck classification.

Lastly, the monitoring frequency in mobile networks should be dynamic that can be adjusted to how frequently different issues arise in the mobile network. For example it should be reduced in case the mobile network runs smoothly, most of the time. Same should be the case with the frequency of re-training the bottleneck classification model. In case an anomaly is reported, the Telemetry Collector can increase the monitoring frequency until no more anomalies are reported by UEs. To avoid acting upon isolated cases of anomalies being reported by one or very few UEs, the monitoring and analysis system of the mobile network should only activate if a threshold percentage of UEs per base station, report an anomaly at same time.

8 Related Work

In the following subsections, we explore the related work in the areas of bottleneck identification, INT based telemetry and anomaly detection in the mobile networks.

8.1 Telemetry for Identifying Bottlenecks

There is a big interest in developing telemetry solutions for softwarised networks [62], [63], [64]. Advanced monitoring solutions have been proposed to identify bottlenecks, while attempting to balance high detection rates with minimal monitoring overhead costs.

Some of the earlier research studies used a single telemetry approach such as passive monitoring [65], [66] to detect if a capacity bottleneck is inflicting the network. As most transmissions comprise of TCP communications, these early works monitored the status of individual TCP transmissions just outside of the 3G network on a link to the server in the Internet [66], or inside the core network [65] to see if there is any congestion in the mobile network. These studies provide a coarse grained indication of presence or absence of a bottleneck in the network but do not identify the segment of the mobile network having the issue. With both active and passive measurements, an other study [7] preferred deep-learning method to detect bottleneck in the core of a cloudified mobile network.

A somewhat granular approach is proposed by QProbe [67], in which a train of small UDP probe packets are transmitted by a server, outside the mobile network, to a UE investigating if the end-to-end medium is congested. Using Decision-Tree ML model, it exploits the inter-packet arrival delay between the first and last probe packets of the probe train, as well as the inter-packet gap between each two adjacent probe packets arriving at the UE to locate, with above 80% of accuracy, if the congestion has sprawled from the last mile or rest of the end-to-end path. With Random-Forest as an ML model, the study in Q-TSLP [5], reduces the granularity of characterizing congestion related bottlenecks into radio access link, RAN, core of the mobile network, and rest of end-to-end path towards a server. It however, uses only active monitoring, similar to QProbe, and TSLP [68] a scheme originally designed to measure congestion on inter-domain links in Internet.

A recent experimental study conducted by G. Patounas et al. [4] used a hierarchical clustering method with Soergel/Tanimoto distance to separate different single bottleneck profiles from the baseline performance and from each other. They preferred Fuzzy clustering for identifying composite bottleneck profiles. The study was conducted on a testbed that deployed Orion RAN slice and virtualised EPC components based on OAI 4G LTE. Using passive monitoring, measurements were logged at all components and VNFs of the mobile system as well as at UE and server. The monitored measurements were divided into 3 groups i.e., service, network and infrastructure layers. The study performed a centralised analysis of all the collected measurements and concluded that infrastructure layer, followed by network layer measurement features are the most predictive in bottleneck identification.

8.2 Use of P4-INT

Other than traditional monitoring approaches of passive and active probing, the third type of telemetry based on P4 programming was initiated by data centers. To track status of switches and the data flows between its end-hosts, the data centers used both INT and out-of-band monitoring.

For example R. Joshi et al. proposed BurstRadar [23], an out-of-band method, to monitor microbursts at the egress pipeline of each switch in the network. To make good use of the computing, memory and bandwidth resources of the end-hosts, switches and the controller in a data center, OmniMon [24] splits the telemetry tasks between the end-hosts and switches and merges the results for analysis at the controller. To reduce bandwidth overhead, PINT [69] proposes the use of constant bit-budget i.e., 16 bits per packet to carry queue occupancy, switch utilization statistics, path tracing and tail latency of flows. Use of P4 based telemetry has gradually being employed in other networks such as Software Defined Networks (SDN). N.V. Tu et al. [70] present an INT based monitoring system for Open Network Operating System (ONOS), a widely used SDN controller for SDN data plane. It employed P4-INT to be aware of real time traffic, real time latency and link changes to detect anomalies (e.g., traffic spikes) or failures (e.g., terminated links) and hence to take right actions quickly.

In the context of mobile core networks, P4 switches have been used for real-time attack detection and mitigation [71], identification of malicious data [72], enhancing the User Plane Function (UPF) [73] and ensuring QoS at the slice level [74]. A recent study [59] investigated impact of P4-INT on network performance compared to the traditional active and passive monitoring approaches. [59] built a 4G network with OAI upon virtual box, with its different components as VMs. It deployed two switches, both in network core at SGi interface, with a dummy forwarder node in between the two switches. The authors found that processing and memory overhead of software switches is more than both the active (Ping) and passive (packet capture) monitoring on the switches. But that passive approach suffered high I/O disk activities and both the active and passive monitoring dropped the throughput substantially, while for P4-INT it was similar to the baseline.

8.3 Detecting Anomaly in Mobile Networks

Several machine learning-based methods have been presented focusing on detecting anomalies in network traffic. For example, Hadj-Kacem et al. [75] proposed an anomaly detection model that captured the correlation between the different KPIs in a mobile network using functional principal component analysis (FPCA). They used the logistic regression classifier for the functional data to predict anomalies. The logistic regression model achieved accuracy and F1-score of 71% and 70%, respectively. Also, [76] leveraged supervised classification models, namely logistic regression, random forest, LightGBM and an ensemble classifier of these earlier three models to detect high latency in mobile broadband networks.

Other than supervised ML, various unsupervised methods have been proposed for anomaly detection in mobile networks. [77] presented an autoencoder-based unsupervised model to detect cell outages in mobile networks leveraging measurements from the UEs namely, RSRP and RSRQ values of the serving cell and the neighboring cells, and the radio link failure (RLF). Also, [78] proposed a framework based on LSTM-autoencoder and One-class SVM (OC-SVM) to detect abnormal traffic data. Recently, [79] proposed a distributed anomaly detection framework for network data forwarding latency in an unsupervised fashion. The study used the hierarchical temporal memory (HTM) algorithm for the online detection of anomalies.

Among the above-mentioned studies [7], [4], [59] and [77] are similar in certain aspects to ours. [7] worked on bottleneck detection in a cloudified mobile network as we do in stage-1. We aim to characterise different bottleneck profiles as in [4]. We employ a combination of monitoring approaches including active, passive and P4-INT based measurements such as in [59]. For anomaly detection at UE, we run an an autoencoder-based unsupervised method like that of [77]. As for dissimilarities, [7] is limited to bottlenecks sourced from network core only. [4], aimed to identify the monitoring layer whose parameters played primary role in distinguishing among different bottleneck profiles. [59] performed a limited study on network core, to understand the overhead and impact of the three monitoring mechanisms of active, passive and P4-INT on end-to-end performance. As for anomaly detection at UE, [77] worked only on cell outage detection.

9 Conclusion

In this paper, we present a 2-stage distributed telemetry framework to identify and attribute bottlenecks in a cloudified mobile network and its last mile. The system includes monitoring both at the mobile network and its UEs. Monitoring at UEs assists in triggering identification of bottleneck events that impact user's experience. Inclusion of a UE not only relieves the mobile network from continuous computation of data analytics but also helps the monitoring system to catch a bottleneck that is beyond the internal scope of the mobile network, such as radio interference. Mild bottlenecks at the mobile network, however may go un-noticed by a UE when they do not degrade its application performance.

By leveraging measurements at a UE, our VAE based model accurately detects different types of bottlenecks with 0.85 F1-score. To attribute the cause and location of the bottlenecks, our classification model achieves 0.89 F1-score. Overall, the bottleneck identification accuracy of our distributed framework is comparable to that of a centralized approach, making it a better choice due to the non-feasibility of a centralized system.

Working with a combination of monitoring approaches, our study further reveals that in-band network telemetry can be the potential future alternate for active monitoring of mobile network links.

In this study, we have provided a proof-of-concept of our distributed telemetry framework using generic application traffic at the UEs, virtual P4 switches and a cloudified mobile core based on 4G VNFs. In the future, we plan to focus on real applications of our framework. We will upgrade our testbed with 5G VNFs realizing a 5G standalone core and hardware P4 switches. This will allow us to evaluate our framework on use cases where the UE is using novel applications with very high requirements, such as live broadcasting and networked music over 5G networks. Finally, we will explore the feasibility of applying the framework online.

REFERENCES

[1] Y. Zhang, M. Chen, and R. X. Lai, "Cloudified and software defined 5G networks: Architecture, solutions, and emerging applications," *Mobile Networks and Applications*, vol. 21, pp. 727–728, 2016.

[2] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[3] 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*, 2017, pp. 79–87.

[4] 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.

[5] M.-R. Fida, A. F. Ocampo, and A. Elmokashfi, "Measuring and Localising Congestion in Mobile Broadband Networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 366–380, 2021.

[6] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[7] F. Michelinakis, J. S. Pujol-Roig, S. Malacarne, M. Xie, T. Dreibholz, S. Majumdar, W. Y. Poe, G. Patounas, C. Guerrero, A. Elmokashfi *et al.*, "AI Anomaly Detection for Cloudified Mobile Core Architectures," *IEEE Transactions on Network and Service Management*, 2022.

[8] B. Daroczy, P. Vaderna, and A. Benczur, "Machine learning based session drop prediction in lte networks and its son aspects," in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, 2015, pp. 1–5.

[9] M. Rajiullah, A. Lutu, A. S. Khatouni, M.-R. Fida, M. Mellia, A. Brunstrom, O. Alay, S. Alfredsson, and V. Mancuso, "Web Experience in Mobile Networks: Lessons from Two Million Page Visits," in *The World Wide Web Conference*. San Francisco, CA, USA: ACM, 2019, pp. 1532–1543.

[10] G.-M. Su, X. Su, Y. Bai, M. Wang, A. V. Vasilakos, and H. Wang, "Qoe in video streaming over wireless networks: perspectives and research challenges," *Wireless networks*, vol. 22, pp. 1571–1593, 2016.

[11] A. Nediyanchath, C. Singh, H. J. Singh, H. Mangla, K. Mangla, M. K. Sakhala, S. Balasubramanian, S. Pareek, and Shwetha, "Anomaly detection in mobile networks," *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1–5, 2020.

[12] T. Prabhakar and M. N. Veena, "Review on anomaly detection in mobile networks using traditional learning, machine learning and deep learning," *Journal of Computational and Theoretical Nanoscience*, vol. 17, pp. 4789–4796, 2020.

[13] M. A. Ridwan, N. A. M. Radzi, F. Abdullah, and Y. E. Jalil, "Applications of Machine Learning in Networking: A Survey of Current Issues and Future Challenges," *IEEE Access*, vol. 9, pp. 52 523–52 556, 2021.

[14] M.-R. Fida, A. F. Ocampo, and A. Elmokashfi, "Measuring and Localising Congestion in Mobile Broadband Networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 366–380, 2022.

[15] K.-M. Chen, T.-H. Chang, K.-C. Wang, and T.-S. Lee, "Machine Learning Based Automatic Diagnosis in Mobile Communication Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 10, pp. 10 081–10 093, 2019.

[16] M. Hirth, T. Hossfeld, M. Mellia, C. Schwartz, and F. Lehrieder, "Crowdsourced network measurements: Benefits and best practices," *Computer Networks*, vol. 90, pp. 85–98, Jul. 2015.

[17] D. Baumann, "Minimization of drive tests (MDT) in mobile communication networks," in *Proceeding zum Seminar Future Internet (FI) und Innovative Internet Technologien und Mobilkommunikation (IITM)*, vol. 9, 2014, p. 7.

[18] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network Monitoring in Software-Defined Networking: A Review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, 2018.

[19] D. Harrington, R. Presuhn, and B. Wijnen, "An Architecture for describing Simple Network Management Protocol (SNMP) Management Frameworks," Tech. Rep., 2002.

[20] L. MartinGarcia, "TCPDUMP and LiBPCAP," Jun. 2023. [Online]. Available: https://www.tcpdump.org/

[21] Wireshark, "Wireshark: The World's Most Popular Network Protocol Analyzer," Jun. 2023. [Online]. Available: https://www.wireshark.org/

[22] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-Band Network Telemetry via Programmable Dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[23] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems, APSys 2018, Jeju Island, Republic of Korea, August 27-28, 2018*. ACM, 2018, pp. 8:1–8:8.

[24] Q. Huang, H. Sun, P. P. C. Lee, W. Bai, F. Zhu, and Y. Bao, "OmniMon: Re-architecting Network Telemetry with Resource Efficiency and Full Accuracy," in *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, H. Schulzrinne and V. Misra, Eds. Virtual: ACM, Aug. 2020, pp. 404–421.

[25] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[26] M. Budiu and C. Dodd, "The P4-16 Programming Language," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.

[27] 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. [Online]. Available: https://www.simula.no/file/m2ec2020pdf/download

[28] OpenAirInterface Software Alliance, "OpenAirInterface," Jun. 2023. [Online]. Available: https://openairinterface.org

[29] Mosaic5g, "FlexRAN," Jun. 2023. [Online]. Available: http://mosaic5g.io/flexran

[30] ETSI, "Open Source MANO," Jun. 2023. [Online]. Available: https://osm.etsi.org

[31] 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. [Online]. Available: https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf

[32] Canonical, "Juju," Jun. 2023. [Online]. Available: https://jaas.ai

[33] OpenStack, "OpenStack Zed Installation Guides," Jun. 2023. [Online]. Available: https://docs.openstack.org/zed/install/index.html

[34] OpenStack, *OpenStack Installation Guide*, Jun. 2023. [Online]. Available: https://docs.openstack.org//install-guide/InstallGuide.pdf

[35] M. Xie, T. Dreibholz, F. I. Michelinakis, J. Pujol-Roig, W. Y. Poe, A. M. Elmokashfi, S. Majumdar, and S. Malacarne, "An Exposed Closed-Loop Model for Customer-Driven Service Assurance Automation," in *Proceedings of the 30th IEEE European Conference on Networks and Communications (EuCNC)*, Porto/Portugal, Jun. 2021, pp. 419–424.

[36] M. Xie, J. S. Pujol-Roig, F. I. Michelinakis, T. Dreibholz, C. Guerrero, A. G. Sánchez, W. Y. Poe, Y. Wang, and A. M. Elmokashfi, "AI-Driven Closed-Loop Service Assurance with Service Exposures," in *Proceedings of the 29th IEEE European Conference on Networks and Communications (EuCNC)*, Dubrovnik, Dubrovnik-Neretva/Croatia, Jun. 2020, pp. 265–270.

[37] J. Nota, "BMv2 Simple Switch," Jun. 2022. [Online]. Available: https://github.com/nsg-ethz/p4-learning/wiki/BMv2-Simple-Switch

[38] S. Godard, "Sysstat Tutorial," Jun. 2023. [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/tutorial.html

[39] J. B. Postel, "Internet Protocol," no. 791, Sep. 1981. [Online]. Available: https://tools.ietf.org/rfc/rfc791.txt

[40] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and its Applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[41] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical Real-Time Microburst Monitoring for Datacenter Net-

works," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, 2018, pp. 1–8.

[42] V. Gueant, "iPerf – The TCP, UDP and SCTP Network Bandwidth Measurement Tool," Jun. 2023. [Online]. Available: https://iperf.fr

[43] Linux Foundation, "NetEm," Jun. 2023. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem

[44] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy, *Linux Advanced Routing & Traffic Control HOWTO*, May 2012. [Online]. Available: https://www.lartc.org/lartc.pdf

[45] C. I. King, "stress-ng," Jun. 2023. [Online]. Available: https://wiki.ubuntu.com/Kernel/Reference/stress-ng

[46] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *8th IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[47] ——, "Isolation-Based Anomaly Detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, pp. 1–39, 2012.

[48] Z. Ding and M. Fei, "An Anomaly Detection Approach based on Isolation Forest Algorithm for Streaming Data using Sliding Window," *IFAC Proceedings Volumes*, vol. 46, no. 20, pp. 12–17, 2013.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[50] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv preprint arXiv:1312.6114*, May 2014. [Online]. Available: https://arxiv.org/pdf/1312.6114

[51] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and Composing Robust Features with Denoising Autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 1096–1103.

[52] H. I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean, "InceptionTime: Finding AlexNet for Time Series Classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, 2020.

[53] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013, vol. 112.

[54] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," *Advances in neural information processing systems*, vol. 25, 2012.

[55] L. Tang, S. Rajan, and V. K. Narayanan, "Large scale multi-label classification via metalabeler," in *Proceedings of the 18th international conference on World wide web*, 2009, pp. 211–220.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[57] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[58] L. Antwarg, R. M. Miller, B. Shapira, and L. Rokach, "Explaining Anomalies Detected by Autoencoders using SHAP," *arXiv preprint arXiv:1903.02407*, 2019.

[59] V. T. Moen, "Virtualized 4G long term evolution testbed to investigate the performance impact of in-band network telemetry," Master's thesis, OsloMet-storbyuniversitetet, 2021.

[60] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz, "Quantification of YouTube QoE via Crowdsourcing," in *2011 IEEE International Symposium on Multimedia*. IEEE, 2011, pp. 494–499.

[61] T. Hoßfeld, M. Hirth, J. Redi, F. Mazza, P. Korshunov, B. Naderi, M. Seufert, B. Gardlo, S. Egger, and C. Keimel, "Best Practices and Recommendations for Crowdsourced QoE-Lessons learned from the Qualinet Task Force" Crowdsourcing"," 2014.

[62] G. Kakkavas, A. Stamou, V. Karyotis, and S. Papavassiliou, "Network Tomography for Efficient Monitoring in SDN-enabled 5G Networks and Beyond: Challenges and Opportunities," *IEEE Communications Magazine*, vol. 59, no. 3, pp. 70–76, 2021.

[63] 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.

[64] R. Boutaba, N. Shahriar, M. A. Salahuddin, S. R. Chowdhury, N. Saha, and A. James, "AI-driven Closed-loop Automation in 5G and beyond Mobile Networks," in *Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility*, 2021, pp. 1–6.

[65] F. Ricciato, F. Vacirca, and P. Svoboda, "Diagnosis of Capacity Bottlenecks via Passive Monitoring in 3G Networks: An Empirical Analysis," *Computer Networks*, vol. 51, no. 4, pp. 1205–1231, 2007.

[66] M. Schiavone, P. Romirer-Maierhofer, F. Ricciato, and A. Baiocchi, "Towards Bottleneck Identification in Cellular Networks via Passive TCP Monitoring," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2014, pp. 72–85.

[67] 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*, 2015, pp. 1–7.

[68] M. Luckie, A. Dhamdhere, D. Clark, B. Huffaker, and K. Claffy, "Challenges in Inferring Internet Interdomain Congestion," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014, pp. 15–22.

[69] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic In-band Network Telemetry," in *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, Virtual Event, USA, August 10-14, 2020*, H. Schulzrinne and V. Misra, Eds. ACM, 2020, pp. 662–680.

[70] N. V. Tu, J. Hyun, and J. W.-K. Hong, "Towards ONOS-based SDN Monitoring using In-band Network Telemetry," in *19th Asia-Pacific Network Operations and Management Symposium, APNOMS 2017, Seoul, Korea (South), September 27-29, 2017*. IEEE, 2017, pp. 76–81.

[71] M. Bonfim, M. Santos, K. Dias, and S. Fernandes, "A Real-Time Attack Defense Framework for 5G Network Slicing," *Software: Practice and Experience*, vol. 50, no. 7, pp. 1228–1257, 2020.

[72] O. A. Fernando, H. Xiao, and J. Spring, "Developing a Testbed with P4 to Generate Datasets for the Analysis of 5G-MEC Security," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2022, pp. 2256–2261.

[73] R. MacDavid, C. Cascone, P. Lin, B. Padmanabhan, A. Thakur, L. Peterson, J. Rexford, and O. Sunay, "A P4-based 5G User Plane Function," in *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, 2021, pp. 162–168.

[74] R. Ricart-Sanchez, P. Malagon, A. Matencio-Escolar, J. M. A. Calero, and Q. Wang, "Toward Hardware-Accelerated QoS-Aware 5G Network Slicing based on Data Plane Programmability," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 1, 2020.

[75] I. Hadj-Kacem, S. B. Jemaa, S. Allio, and Y. B. Slimen, "Anomaly prediction in mobile networks: A data driven approach for machine learning algorithm selection," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–7.

[76] A. H. Ahmed, S. Hicks, M. A. Riegler, and A. Elmokashfi, "Predicting high delays in mobile broadband networks," *IEEE Access*, vol. 9, pp. 168 999–169 013, 2021.

[77] Y.-H. Ping and P.-C. Lin, "Cell outage detection using deep convolutional autoencoder in mobile communication networks," in *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 2020, pp. 1557–1560.

[78] M. Said Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Network anomaly detection using LSTM based autoencoder," in *Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, 2020, pp. 37–45.

[79] A. H. Ahmed, M. A. Riegler, S. A. Hicks, and A. Elmokashfi, "RCAD: Real-time Collaborative Anomaly Detection System for Mobile Broadband Networks," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2682–2691.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. 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/TMC.2023.3312051

Mah-Rukh Fida is a senior lecturer at School of Computing and Engineering, University of Gloucestershire, UK. She was post-doctoral research fellow at Simula Metropolitan Centre for Digital Engineering in Norway during Nov. 2018 to March 2020. Sponsored by the prestigious UK Commonwealth Award, she completed her Ph.D. from The University of Edinburgh, UK in 2018. She received degree of M.Sc. (IT) with distinction from IMSciences, KP, Pakistan in 2010. Before PhD, Mah-Rukh served as a lecturer in Computer Science, at Shaheed Benazir Bhutto Women University, Pakistan. Dr. Mah-Rukh's existing work focuses on measurement and analysis of performance within mobile broadband networks. Her research interest includes impact of context, content and interplay of different network components on end-to-end quality of experience.

Azza H. Ahmed is a Ph.D. student at Simula Metropolitan Centre for Digital Engineering in Oslo, Norway. She received her Master's degree from the University of Nottingham in 2012. Her research interests include communication networks management and control, network performance optimization, network automation and machine learning to solve networks problems.

Thomas Dreibholz received his Diplom (Dipl.-Inform.) degree in Computer Science from the University of Bonn in Bonn, Germany in 2001. Furthermore, he received his Ph.D. degree (Dr. rer. nat.) in 2007, as well as his Habilitation (Priv.-Doz.) degree in 2012 from the University of Duisburg-Essen in Essen, Germany. Now, he works as Chief Research Engineer for the Simula Metropolitan Centre for Digital Engineering (SimulaMet) in Oslo, Norway. He has published and presented more than 90 research contributions at international conferences and in journals, on the topics of Reliable Server Pooling (RSerPool), the Stream Control Transmission Protocol (SCTP), Quality of Service (QoS), as well as multi-homed network and cloud infrastructures. Furthermore, he has contributed multiple Working Group and Individual Submission Drafts to the IETF standardisation processes of RSerPool and SCTP. He is also co-author of multiple RFC documents published by the IETF.

Andres F. Ocampo received the B.S. degree in telecommunications engineering from Pontifical Bolivarian University, Medellín, Colombia, in 2010, and the M.S. degree in telecommunications engineering from The University of Antioquia, Medellin Colombia, in 2016. Currently, he is a Ph.D. candidate at SimulaMet's Centre for Resilient Networks and Applications (CRNA) in Oslo, Norway. His doctoral research focus is to develop mechanisms to provide real-time support on the Cloud Radio Access Architecture for 5G and beyond generations of mobile systems.

Ahmed Elmokashfi is a Research Professor at Simula Metropolitan Centre for Digital Engineering in Norway. He is currently heading the Centre for Resilient Networks and Applications (CRNA), which is part of the Simula Metropolitan Centre that is funded by the Norwegian Ministry of Transport and Communication. Dr. Elmokashfi's research interest lies in network Measurements and Performance. In particular, he has been focusing on studying the resilience, scalability, and evolution of the Internet infrastructure; the measurement and quantification of robustness in mobile broadband networks; and the understanding of dynamical complex systems. Over the past few years,he has been leading and contributing to the development, operation and management of the NorNet testbed infrastructure, which is a countrywide measurement setup for monitoring the performance of mobile broadband networks in Norway. Dr. Elmokashfi received his Ph.D. degree from the University of Oslo in 2011.

Foivos Michelinakis is a research scientist at Simula Metropolitan, working on verification and validation of commercial and testbed 5G deployments and usecases. His research interests include Mobile Networks, Network Optimization and Content Distribution Networks. He obtained his Ph.D. on "Optimizing the delivery of Multimedia over mobile networks" from IMDEA Networks Institute and University Carlos III of Madrid in September 2018. He also holds a master in Telematics from University Carlos III of Madrid and a 5-year engineering diploma in Electrical Engineering from National Technical University of Athens.