Quality of Monitoring for Cellular Networks

Naser Hossein Motlagh, Shubham Kapoor, Rola Alhalaseh, Sasu Tarkoma, *Senior Member, IEEE*, and Kimmo Hätönen

***Abstract*—5G networks and beyond introduce a larger number of Network Elements (NEs) and functions than former cellular generations. The increase in NEs will, thus, result in significantly increasing the Management-Plane (M-Plane) data collected from the NEs. Therefore, the conventional centralized Network Management Systems (NMSs) will face fundamental challenges in processing the M-Plane data. In this paper, we present the concept of Quality of Monitoring (QoM) as a solution, which is able to reduce the M-Plane data already at the NEs. First, QoM aggregates the raw M-Plane data into Key Performance Indicators (KPIs). To these KPIs, the QoM applies a data-driven algorithm to define information loss limits for QoM classes specific for each KPI time series. Then, the QoM applies the classes for compressing the KPI data utilizing a lossy-compression method, which is a derivative of the Piece-Wise Constant Approximation (PWCA) algorithm. To evaluate the performance of the QoM solution, we use M-Plane raw data from a live LTE network and calculate four KPIs, while each KPI has different statistical characteristics. We also define three QoM classes named *Exact*, *Optimized*, and *Sharp*. For all KPIs, the class *Optimized* has a higher compression rate than the class *Exact*, while the class *Sharp* has the highest compression rate. Assuming that, for example, NEs of a network produce 280 MB of raw data containing information that needs to be transferred to the network operations center; we use KPIs to represent the information contents of the data, and QoM solution to transfer the data over the network. As a result, the QoM solution achieves an estimated 95% compression gain from the raw data in transfer.**

***Index Terms*—Quality of monitoring, QoM, cellular networks, data management, LTE-4G, and 5G.**

I. Introduction

THE EVOLUTION of the 5^{th} Generation of Mobile Networks (5G) and beyond is accompanied by an increase in the number of end devices and network services [1]. 5G networks are expected to have more distributed NEs (e.g., small cells) to meet the demand of ultra-low latency and high throughput to the end-users [2]. In addition, due to investment and user retention reasons, network providers are generally reluctant to scrap the old NEs immediately from their network. Network providers are compelled to integrate the new NEs into existing infrastructure instead of replacing the old ones. This increases the complexity and heterogeneity of the mobile networks leading to challenges in network management and operation [3]. The multiplication of NEs and the complexity of networks will therefore increase the volume of data sourced from the NEs; and brings up the data transfer, data storage, and data aggregation challenges at the network Management-Plane (M-Plane).

Indeed, to operate these complex mobile networks in an optimal state, fast collection of network statistics from the M-Plane enables monitoring and management of the networks in real-time. While the collection and processing of M-Plane data are facilitated by network management systems (NMSs), this data is used for network planning, maintenance of the end-users quality of services, and monitoring of NEs' performance [4]. It is worth noting that the "M-Plane" refers to functions, interfaces, protocols, data formats, and storage, which are used by NMSs and applications to control, configure and monitor the network status [5], [6].

To overcome the challenges presented above, the research in [5] presents a simple architecture for distributed computation and delivery of data. The architecture proposed therein performs aggregation and refinement of data at the nodes located at the network operator's edge cloud and near to producers of M-Plane data. In addition to the architecture presented in [5], in this paper, to deal with the challenges of M-Plane data, we propose the quality of monitoring (QoM) concept as a solution that uses a set of classes to perform mobile edge computation and compression of M-Plane data at the NEs. Whereas, each QoM class has an accuracy limit and allows a specific amount of information loss. Our proposed QoM solution can be implemented at different elements of the 4G, 5G and beyond networks (i.e., eNBs and gNBs) such as picocells, femtocells, relays, and cellular-based edge computers.

The QoM solution is designed to minimize the amount of data to be transferred, that is by filtering data already at the source and removing redundant or similar data. The QoM solution achieves compression in three stages. In the first stage, to reduce the amount of M-Plane data after collecting and parsing the data, semantic compression is performed by aggregating M-Plane data into the desired key performance indicators (KPIs). In the second stage, a lightweight data-driven method is implemented and identifies the information

Manuscript received June 30, 2021; accepted September 3, 2021. Date of publication September 14, 2021; date of current version March 11, 2022. The research was supported by the Nokia Center for Advanced Research (NCAR) and the Academy of Finland project IDEA-MILL with grant number 335934. The associate editor coordinating the review of this article and approving it for publication was L. Granville. *(Corresponding author: Naser Hossein Motlagh.)*

Naser Hossein Motlagh, Shubham Kapoor, and Rola Alhalaseh are with the Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland (e-mail: naser.motlagh@helsinki.fi; shubham.kapoor@helsinki.fi; rola.alhalaseh@helsinki.fi).

Sasu Tarkoma is with the Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland, and also with the Center for Ubiquitous Computing, University of Oulu, 90570 Oulu, Finland (e-mail: sasu.tarkoma@helsinki.fi).

Kimmo Hätönen is with the Department of Network Systems and Security Research, Nokia Bell Labs, 02610 Espoo, Finland (e-mail: kimmo.hatonen@nokia-bell-labs.com).

Digital Object Identifier 10.1109/TNSM.2021.3112467

loss limits for QoM classes. In the third stage, a lossy compression of performance metrics is performed by removing portions of data with redundant or small information content. This compression is performed so that the relative amount of information needed by the applications consuming this data does not get compromised. The QoM solution is based on the Publish-Subscribe (Pub-Sub) communication paradigm for data delivery [7] that helps to optimize the data collection and delivery from the NEs [5].

To evaluate the feasibility of QoM solution, we implement a Pub-Sub paradigm using Apache Kafka [8] for streaming of the compressed data. We also use the M-Plane data from four KPIs of different NEs from a live LTE network, each having different statistical characteristics. We define three QoM classes named *Exact*, *Optimized*, and *Sharp* and derive their parameters from the data during a two-week training period. Then, we apply a lossy-compression algorithm called the modified version of the Piece-Wise Constant Approximation (PWCA) algorithm for compression of the M-Plane data. The evaluation results show the effectiveness of the proposed QoM solution by reducing the volume of M-Plane data through the removal of data with small information content as well as redundant data.

The remainder of this paper is constructed as follows. Section II presents the state of the art on the data management. Section III explains the concept of the QoM and its core elements. Section IV presents the QoM classes. Section V explains the evaluation of QoM implementation and presents the evaluation results. Finally, Section VI concludes the paper.

TABLE I
SUMMARY OF ABBREVIATIONS AND NOTATIONS

Notation	Description
NE	Network Element
NMS	Network Management System
ML	Machine Learning
M-Plane	Management-Plane
KPI	Key Performance Indicator
MEC	Multi-Access Edge Computing
MSE	Mean Squared Error
SON	Self Organizing Network
NOC	Network Operator Center
RAN	Radio Access Network
QoE	Quality of Experience
PWCA	Piece-Wise Constant Approximation
mPWCA	modified PWCA
LTE	Long Term Evolution
QoM	Quality of Monitoring
DF	Data Fetcher
DS	Data Switch
DH	Data Hub
σ_A	Absolute Consecutive Deviation
$\mathcal{I}_{loss}$	Information Loss Limit
$PieceConstant$	List of Tuples
S	Time-series Data
N	Number of KPIs
k	Computation Counters
x	Number of NEs
K	Number of Transactions
V	Volume of KPI Data
C	Compression Rate
E	Error Rate

II. State of the Art

The evolving cellular networks such as 5G and beyond will become increasingly heterogeneous with a constantly increasing number of network elements (NEs). These numerous NEs will generate huge amounts of measurement data that can be used as M-Plane data to monitor and analyze the network performance as well as the quality of service. The M-Plane data are indeed numerical counters, such as the number of user equipment attaching to the NE or the number of handovers or different failures, and physical measurements, e.g., the user equipment's signal strength. In practice, these performance measurements are periodically generated and aggregated to KPI variables [9]. These KPIs provide necessary statistics to the network operators, helping them to identify their network's state such as the parts of the network which are overloaded or underutilized, as well as the malfunctioning parts of NEs. However, due to periodic reporting and a large number of performance measurements, KPI time series themselves contribute to the significant portion of M-Plane data [10]. Therefore, new approaches are required to efficiently manage the M-Plane data to guarantee the performance of the network. Indeed, using the traditional quality and performance management systems that are based on processing data at the management plane in a central location will not be an efficient approach for the management of multiplying network resources.

In literature, different studies present various approaches for improving the users' quality-of-experience (QoE) and performance management in cellular networks. For example, the research in [11] proposes a framework for mobile network optimization and uses a dataset collected from users' equipment and mobile networks to improve the users' QoE. The research in [12] aims to improve the users' QoE through multimedia services using specific KPIs and proposes neural networks to automatically classify KPIs. The work in [13] focuses on quality monitoring and estimating user's QoE in real-time using QoE-Agents based on a QoE-layered model. More specifically, this work aims to identify the location and the operation of the QoE-Agents based on the accuracy of the measurements and the load in networks. The results show acceptable error ranges from the measurements, low CPU utilization, and acceptable memory utilization. Another study in [14] extracts a large quantity of network operators' data to identify the most recurring use cases to reduce the churn rate and increase the average revenue per user. The work in [15] also monitors communication activity patterns of machine-to-machine devices over the cellular network and estimates the quality of services perceived by users.

To reduce the data volume and efficiently utilize the resources, other studies apply machine learning methods and compression techniques. For example, the study in [16] adopts neural network regression as compression and uses combining techniques including the coefficient averaging, euclidean distance, cosine similarity, and re-learning in order to achieve more accurate data representation and prediction. The research in [17] proposes an artificial intelligence-based lossless compression algorithm to reduce the size of text messages in the network. The research applies the algorithm on a string with average length of 160 characters and shows the algorithm leads to reduction of bandwidth utilization and cost consequently. The study in [18] proposes an algorithm which combines a recurrent neural network predictor and a lossless compression method; and uses genomic and text datasets and achieves around 20% reduction over the traditional compression method, i.e., Gzip. The work in [19] presents a two-level approach that selects a compression framework for individual data points in time-series and then proposes a neural network structure that tunes parameter values automatically. As a result, the framework is capable of improving compression ratio by up to 120% compared to other traditional compression methods. Moreover, another study considers two approaches of data compression techniques and formulates mixed-integer nonlinear problems in order to maximize the energy efficiency in the network. The study conducts a numerical analysis and confirms the performance advantages of the approaches by showing a higher energy efficiency of the network [20].

The existing studies in the literature utilize already stored data in offline mode and present different approaches for improving the users' QoE in cellular networks by applying ML methods and compression techniques. These studies aim to classify KPIs, estimate user's QoE, reduce churn rates, reduce the size of data in the network, or perform mobile network optimization by improving resource utilization and maximizing energy efficiency in the network. In contradistinction to these studies, we introduce the QoM concept as a solution, which uses a set of classes for the compression of M-Plane data at the mobile edge. The QoM uses live stream data from the network and aggregates M-plane data into the desired KPIs and then applies a lossy-compression technique to minimize the amounts of data to be transferred by filtering data already at the NE, leading to efficient utilization of the network resources. Next, we explain the QoM concept and its components.

III. Quality of Monitoring

The QoM is a solution, planned to optimize the management of M-Plane data by mobile network operators. Before transmitting the data, the QoM solution uses a set of classes to minimize the amount of data by removing redundant and repeated data, and data with zero information value. The QoM classes are used to differentiate data collection, data transfer and storage from different NEs as well as network services based on desired data quality and criticality of the equipment and services. The NEs and services that require the highest priority for data collection and monitoring are assigned to the best QoM class, while non-critical elements and services are assigned to a lower QoM class. The lower QoM class for data collection implies that the larger portions of data containing relatively small information are not collected, while the best QoM class collects whole data without any information loss. The QoM solution implements the classes for differentiated monitoring and data compression using a set of components. Based upon their functions and computational resource requirements, these components could

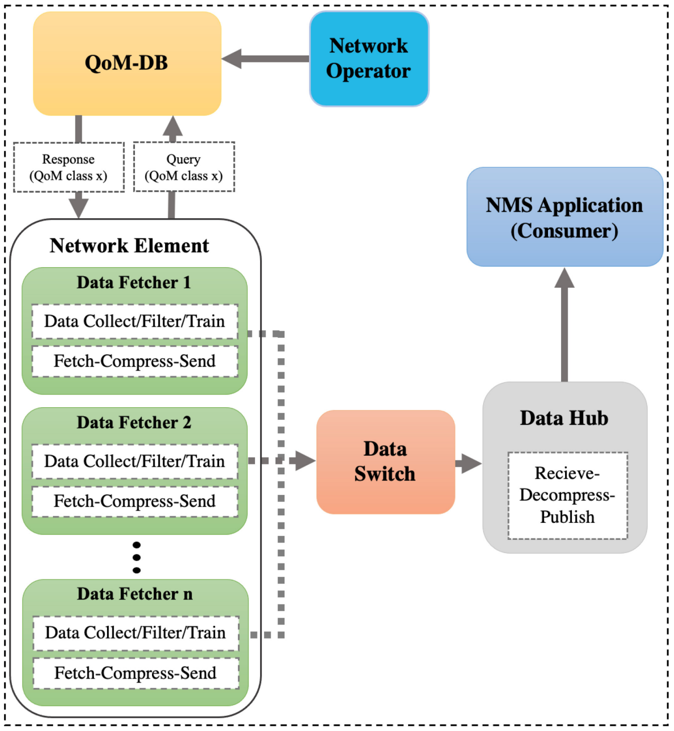


Fig. 1. The high-level system architecture of the QoM concept and it's core components.

be executed in mobile edge cloud or core cloud environments in cellular networks.

The QoM is planned based on the Pub-Sub communication paradigm which optimizes the data collection in two ways. Firstly, the request interaction from the consumer of data (i.e., NMS Application) is reduced by utilizing a subscription pattern. As a result, the consumer makes the subscription request once and the source of data publishes data either sporadically or periodically, depending upon the type of the subscription. There is no need for repeated data queries as in the case of polling-based systems [21]. Secondly, if there are multiple consumers interested in the same stream of data, instead of collecting the same data multiple times, the Pub-Sub paradigm collects the data only once from its source and delivers it to multiple consumers of data. The Pub-Sub communication paradigm thus helps the solution in saving bandwidth resources at the mobile edge. Furthermore, the Pub-Sub paradigm gives flexibility to the consumer to create, modify and cancel a subscription concerning the QoM class on the fly, without affecting other consumers. Fig. 1 illustrates the high-level system architecture of the solution. In the figure, the solid lines represent signaling and the uncompressed data, and dotted lines represent compressed data generated by a particular QoM class. The QoM requires setting up the following components:

Data Fetcher (DF): This component can be an integrated or a separated element that collects data from a network element or a service. A DF filters out the portions of data with zero or small information content. A DF performs the filtering based on a QoM class which is specified in the data collection function. A DF is installed at the multi-access edge computing (MEC) platform or a NE and operates at the edge cloud. A DF is designed to reduce computational and memory footprints. In QoM solution, a DF is the producer and publisher of data in the Pub-Sub paradigm. The DF collects raw M-Plane data from the NE, parses data, performs computations such as *KPI calculation* based upon KPI equations, compresses the data according to the given QoM class; and delivers the aggregated and compressed data to the subscribed NMS applications. A DF communicates with a centralized QoM Database (QoM-DB) to obtain the necessary KPI equations and QoM class definitions. A DF publishes the compressed and filtered data (shown with dotted arrows in Fig. 1) to the Data Switch (DS).

In addition, a DF is divided into two sub-components called *Fetcher* and *Sender*. *Fetcher* interacts with NEs and receives subscription requests from the Data Hub (DH), fetches raw M-Plane data from the NE, uses QoM class definitions received from QoM-DB and stores M-Plane data as training data, and applies KPI equation and performs *KPI calculation* based upon subscription. The *Sender* works as a data producer and interfaces with the DS. Once the Fetcher sub-component completes the *KPI calculation* of the subscriber's requested data, the data will be then available to the sender component. The Sender then publishes this compressed data to DS.

Data Switch (DS): This component is installed on the central cloud of an operator which has sufficient resources and a global view of all NEs connected to it. The DS is responsible for buffering and routing of data generated by DFs to DH. Each data producer (i.e., DFs) and data consumer (i.e., DH) should know the details of the brokers in the DS component. An example of a broker is Apache Kafka consisting of a Kafka cluster and a Kafka ZooKeeper. The Kafka ZooKeeper component of the DS is responsible for coordinating and managing the Kafka cluster that may also consist of multiple Kafka brokers. The DH subscribes to the DS on behalf of actual consumers (i.e., NMS applications). Then, the DS delivers subscription data to the DH that has subscribed for the topic.

Data Hub (DH): This component provides the filtered portions of the collected data for NMS applications needing it. The DH starts the subscription of data, works in a Pub-Sub manner, informs the DS about the new subscriptions, and delivers subscribed data to the NMS Application. The other tasks of the DH include acknowledging the decrease in subscription count if the actual consumers cancel their subscription, updating the subscription count if multiple actual consumers have requested the same subscription, and delivering data to all interested actual consumers for a particular subscription.

In the DS, the Kafka ZooKeeper component that is responsible for managing the Kafka cluster maintains the state of messages (which are being consumed by the particular DH). That is by maintaining the offset of the last message consumed by the DH. In addition, the Kafka ZooKeeper is responsible for creating new Kafka topics for new subscriptions and destroying a Kafka topic if there are no consumers left who are interested in that topic. Each producer (i.e., DF) and consumer (i.e., DH) should know the details of at least one Kafka broker in the DS. However, having information about more than one Kafka broker would be beneficial from a redundancy point of view when a Kafka broker fails.

QoM-DB: This component is a database that runs in a central location between the network operator and the NEs, and allows network functions and NEs to communicate with it. The QoM-DB enables the operators to maintain the *KPI calculation* equations of raw M-Plane data. The QoM-DB exposes two interfaces including *i*) the operator interface that defines necessary models containing parameters and equations, and the QoM class definitions for the particular KPI, and *ii*) the interface which is exposed to DFs and which is used to query about QoM classes applied. These QoM class definitions contain the information loss limits per QoM class for the particular KPI. Through the latter interface, the DFs also request equations for all the KPIs, whose subscription was made by the NMS application. After getting the KPI equations, DF performs the *KPI calculation* and compresses the KPI data using given QoM class definitions. The KPIs are calculated from one or more network (such as LTE) counters. Some KPIs may be calculated using just one counter while some other KPIs may use tens of counters, therefore, calculating KPI aggregates information of various counters into a single value.

IV. QoM Classes

In cellular network operations, a significant portion of M-Plane data is redundant or deviates by a very small amount, practically conveying no significant information. There are also various auxiliary NEs that are not critical but generate the least significant data that comprise a significant share in the overall M-Plane data. In addition, in current NMS solutions, the least significant data holds the same collection and storage priority as that of the non-redundant portion of data from the most critical NEs [22], [23]. Defining classes for the QoM solution allows the network operator to specify to what extent the M-Plane data should be collected from the NEs. In other words, QoM classes determine the information loss limit that the M-Plane data can tolerate in order to regenerate meaningful information at the receiving end.

In practice, QoM classes can be defined by two approaches: *1*) *Network operator-defined QoM class*, in which network operators use their prior knowledge about the network performance metrics to define acceptable information loss limits for a QoM class. *2*) *Data-driven QoM class*, which uses a data-driven method for determining information loss limits for a QoM class at the DF. In this approach, the data which flows in the DFs is collected and used as the training data set to obtain those limits.

Note that there are no known practical limitations on the number of QoM classes. However, in our study we implement the following three QoM classes.

Class Exact: This class corresponds to the exact (with least removal of data), i.e., best possible collection and storage of M-Plane data. In this class, in the compression before the transmission, subsequent redundant data values are removed. The data decompression thus restores the exact data series, and then the information is not lost. This class would be appropriate, for example, for core NEs such as mobility management entity and serving gateway, radio access network

(RAN) elements such as the critical sites, or the elements which are in the proximity of premium customers.

Class Normal: This class corresponds to minimal information loss by removal of all subsequent M-Plane data values, which differ by a small value δ such that $|X[i] - X[m]| \leq \delta << \widehat{X}$, where $X[i]$ is the current data point, $X[m]$ is the last reported data point, and $\widehat{X}$ is the median of data sequence X. Note that the removed data values are approximated by a single data value (i.e., piece constant). Before transmitting the data, the subsequent data values that are too similar to the last reported data point, are removed. The decompression of the compressed data belonging to this class thus produces almost equal data series with very minimal information loss. This class can be used by the network operators for monitoring the least critical NEs such as small/pico/femtocells, which are of no special importance.

Class Sharp: This class corresponds to small information loss by removal of all subsequent M-Plane data values which differ by the small value δ' such that $|X[i] - X[m]| \leq \delta' << \widehat{X}$, where $\delta < \delta'$, $X[i]$ is the current data point, $X[m]$ is the last reported data point and $\widehat{X}$ is the median of data sequence X. This class removes all data values less than δ' away from the last reported data point and represents them by a single value, i.e., the last reported data point. The decompression of the compressed data belonging to this class thus produces similar data series with small information loss. This class can be applied for monitoring auxiliary NEs in the network such as repeater RAN elements and picocells, femtocells, and gNBs in the least critical places.

V. Evaluation of QoM Solution

The proposed QoM concept implements the QoM classes for differentiated monitoring and compression of M-Plane data using a set of components as explained in Section III. Based upon the functions and computational resource requirements, these components can be implemented in a mobile edge cloud or core cloud environment in the cellular network architectures. In this section, we evaluate the QoM solution using an experiment. Hence, we first explain our implementation setup for QoM solution. Then, we evaluate the results achieved from the experiment.

A. Experimental Setup

To evaluate the proposed QoM concept, we carry out an experiment utilizing two private cloud environments provided by Nokia for research and development purposes. As shown in Fig. 2, one environment is based on the HP Matrix cloud [24] and another environment is based on OpenStack [25]. The different components of the QoM solution run in separate virtual machine instances of these two clouds. Fig. 2 illustrates the overall setup of the experiment. In the following, we explain the details of the implementations of the experiment.

Data Fetchers are distributed in both HP Matrix and OpenStack cloud environments and run in separate virtual machines. Each virtual machine has 1 GB of memory, 2.4 GHz dual-core processor, and Ubuntu as the operating system. In the setup, we implemented a total of 25 DFs, while 12 DF

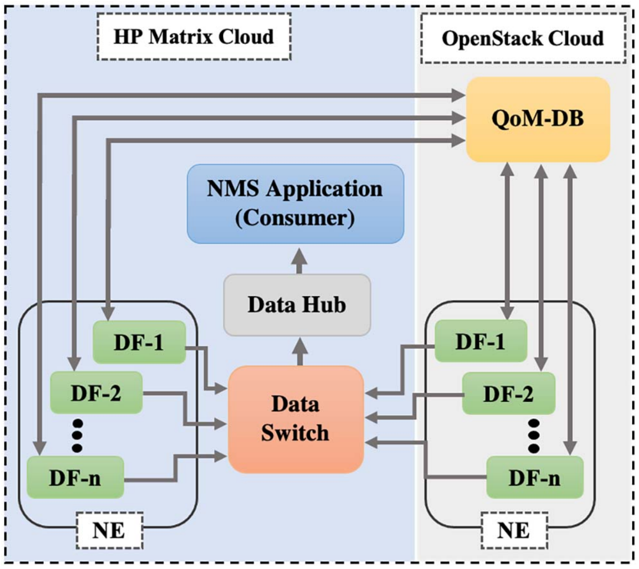


Fig. 2. Experimental setup and the data flow between the components of QoM solution. DF and NE refer to Data Fetcher and Network Element, respectively.

instances operate on HP Matrix cloud and 13 DF instances run on OpenStack cloud. Each DF instance replayed M-Plane data of live LTE Cells, which are part of a real network. As explained earlier in Section III, a DF fetches and sends the data. We implemented the Fetching using a Python script and we utilized Java programming language and the API of Kafka producer for sending the data. Then, we tested DF with a Python interpreter and JDK in the Linux environment.

Data Switch consists of one instance of Apache Kafka Broker and Kafka ZooKeeper. We used Apache Kafka for implementing Kafka Broker and ZooKeeper, which runs on two separate virtual machines in the HP Matrix cloud. Kafka Broker's virtual machine has 16 GB memory, a 2.3 GHz dual-core processor, and CentOS as the operating system. ZooKeeper's virtual machine has 2 GB memory, a 2.4 GHz processor, and Ubuntu as the operating system.

QoM-DB is deployed on a separate virtual machine in the OpenStack Cloud. This virtual machine has 2 GB memory, a 2.3 GHz processor, and Ubuntu as the operating system. In the setup, we used Python programming language in a Linux environment to implement QoM-DB as an HTTP server.

Data Hub implemented on a virtual machine in the HP Matrix cloud. The virtual machine has 2 GB memory, a 2.3 GHz processor, and Ubuntu as the operating system. We also implemented the DH in the Java programming language and tested it with JDK in a Linux environment.

Consumer is an emulated network operator center (NOC) that consumes the M-Plane data. The emulated NOC in a real network scenario would be an actual NOC that monitors the network operation through KPIs. In our setup, we deployed the emulated NOC on a virtual machine in the HP Matrix cloud. This virtual machine has 4 GB memory, a 2.4 GHz processor, and Ubuntu as the operating system. We implemented an emulated NOC as an HTTP server using Python and JavaScript programming languages, and then we made a subscription using the DH.

B. The LTE Network Data

For the experiment, we collected raw M-Plane data from a live LTE network in Northern Europe. The collected data which was about 4-week duration with hourly measurements of LTE performance counters were applied for KPI calculation and finding compression gains achieved from the QoM solution. Then, for evaluation purposes, we calculated four different KPIs. Since the KPIs did not ideally follow any standard statistical distribution, we performed the Cullen and Frey test [26]. The test revealed that a majority of KPIs follow the Beta Distribution [27], allowing us to compute the standard deviation σ for each KPI. Following are the four KPIs and their different statistical characteristics:

- *LTE_5017a:* a static KPI (*Median* = 100; σ = 0.02) whose value is generally 100 and rarely fluctuates.
- *LTE_5178a:* a moderately static KPI (*Median* = 99.65; σ = 2.1) whose value is often 100 and fluctuates occasionally.
- *LTE_5518a:* a non-static KPI (*Median* = 1001.93; σ = 321.32) whose values are changing often.
- *LTE_5520a:* a non-static KPI (*Median* = 906.04; σ = 315.6) whose values are changing often.

Algorithm 1 Modified Piece-Wise Constant Approximation (mPWCA) Algorithm for KPI Series Compression

```
1: Set S = [s[1], s[2], ..., s[n]]
2: Set Information Loss Limit ($\mathcal{I}_{loss} \geqslant 0$)
3: Set PWCA(s) = ()
4: Set i = 1
5: Set m = 0
6: Set PieceConstant = 0
7: Set CurrentPiece = ()
8: while (S.hasMoreValues()) do
9:     m = S[i]
10:    if (| PieceConstant - m | $\geqslant \mathcal{I}_{loss}$) then
11:        CurrentPiece.clear()
12:        append(m,i) to CurrentPiece
13:        append CurrentPiece[0] to PWCA(s)
14:        PieceConstant = CurrentPiece[0][0]
15:    else
16:        append(m,i) to CurrentPiece
17:    end if
18:    i = i + 1
19:    Publish PWCA(s)
20: end while
```

C. Comparing Compression Techniques

The traditional lossless compression and archiving techniques such as *zip*, *gzip*, *bzip2*, *xz*, and *tar* allow the original data to be perfectly reconstructed from the compressed data. These techniques package the data into a dense format by eliminating all the empty spaces or repeating bit patterns in the files [28]. Unfortunately, these techniques have major drawbacks when applying them to a streaming time series of NE performance data (which consist of a set of values of several different performance parameters and measurements). Whereas, each single data point is relatively small compared to the collection of all M-Plane data points for the same period. In addition, each data point contains relatively small amounts of redundant bit patterns. Therefore, the gain that the traditional lossless compression methods provide is small per data point. These lossless compression methods also rely on having the complete compressed bitstream available when decompressing the data, which can not be guaranteed in a real network environment. Because, if any of the data points are lost, the rest of the stream may become impossible to decompress.

In comparison, applying lossy compression for the time-series data will significantly reduce the amount of data. For example, Piece-Wise Constant Approximation (PWCA) is a compression technique, where a set of subsequent data points are represented by a constant value (such as a median or mean value called the piece). These data points that lie close to each other are assumed to be part of a single piece. The PWCA technique includes offline and online algorithms [29]. The PWCA offline algorithms can play a better role in archiving time-series data at the database end, but they cannot be used for compressing M-Plane data at its source (i.e., in the NEs, which have strict velocity requirements of M-Plane data) [30], [31]. Therefore, to implement lossy compression we propose a modified version of the PWCA online algorithm (mPWCA) for QoM compression, inspired by the algorithm proposed in [31]. Applying the mPWCA algorithm overcomes the limitations of the PWCA algorithm and enables faster data collection cycles. It also allows removing redundant data transfers which lead to reducing energy consumption. The logic of our proposed mPWCA is presented in Algorithm 1.

In the algorithm, in lines 1 and 2, the algorithm takes an online time-series $S = [s[1], s[2], \ldots, s[n]]$ and ($\mathcal{I}_{loss} \geq 0$) as the input; and then generates compressed online time-series in line 19 as it's output. In the algorithm, an online time-series $S = [s[1], s[2], \ldots, s[n]]$ is compressed to form a time-series PWCA(S) within the error bound $\mathcal{I}_{loss}$. Where $\mathcal{I}_{loss}$ can be an operator defined or from data derived information loss limit for a given QoM class. In line 10, the algorithm checks if each incoming data point qualifies as the member of the ongoing piece, if and only if it is not deviating from the ongoing piece's constant by more than the information loss limit ($\mathcal{I}_{loss}$), which is specified for compression. In lines 11 and 12, all of the data points which fall under the same piece are represented by the piece constant in the compressed series. In lines 13 and 14, on the creation of a new piece, the constant of the new piece is appended to the compressed list and a new *PieceConstant* is formed. In line 16, if a data point is deviating less than the information loss limit from the ongoing piece's constant, it is appended to the current piece. Note that in the algorithm, *CurrentPiece* is a list of tuples, where tuples are in the form of `[Time constant, KPI value]`. While *CurrentPiece*[0] represents the first tuple of *CurrentPiece* list, the *CurrentPiece*[0][0] represents the value of the first element of the first tuple, which in this case is the KPI value.

D. Compression Gain

The QoM solution achieves compression of data in two steps: *i*) First by performing KPI calculation that is by aggregating LTE raw data in the form of LTE counters to the

LTE KPIs, and *ii*) second by performing lossy compression of the LTE KPI data through the QoM class implementations. Within the first step, we aggregate the LTE counter data into LTE KPIs at the mobile edge. This aggregation leads to a significant reduction in data. For KPI calculation, if we assume that an operator is interested in N KPIs, while each KPI requires k counters. If these KPIs are not calculated at the network edge (i.e., DFs in this paper), then ($N \times k$) data values will be transferred. It is also worth noting that different LTE KPIs require a different number of counters for calculation, and hence the value of k will vary for different KPIs. In our study, the minimum and maximum values of counters k were 1 and 27, respectively. These values are from the LTE KPI documentation of the source network.

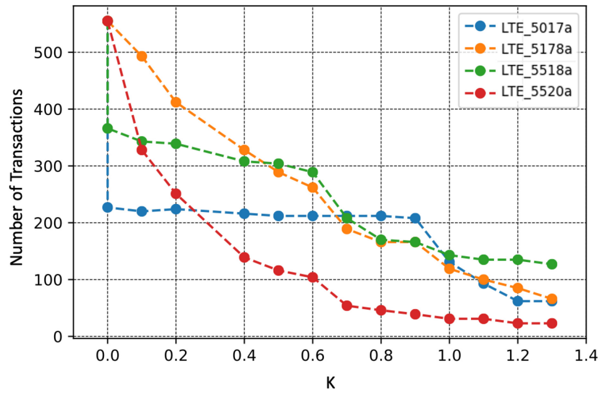
In real life, each NE is capable of producing several thousand parameters in which some of these parameters are updated every second. For instance, in the live LTE data used in this paper, there are about 2800 parameters produced by each of the NEs. Note that each network includes a massive number of operating elements, i.e., NEs. For example, in a large 5G network, there might be from one hundred thousand to one million NEs. From these figures, if we assume the number of NEs is equal to x, thus, we can derive an imaginary scale for the transferred data amounts by $10^5 \times 2800 \times 1/s \times x$ bytes, which results in the order of 280 $\times$ x MB/s. After performing *KPI calculation*, if we assume the average of 10 parameters per KPI is reduced, then the transferable data amount decreases to 28 $\times$ x MB/s per KPI. This number will even be further reduced by the QoM compression (in the second step of the QoM solution) as explained in the following.

In the second step, to show the performance of applying the lossy compression (as explained in Section V-C) on the KPI calculated data and to evaluate the compression gain achieved by QoM solutions; we conducted experiments by both networks operator defined and Data-driven QoM class definition approaches.

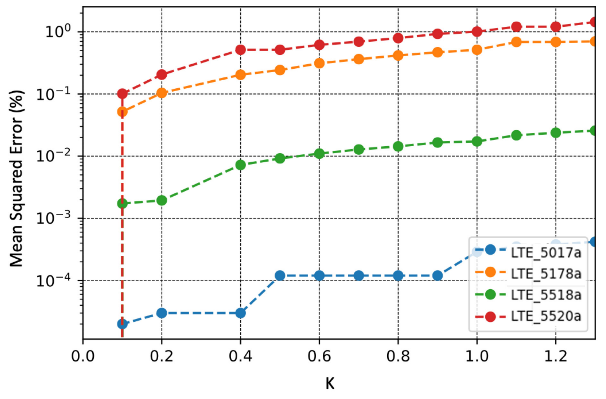
1) Network Operator Class Definition: In our study, the number of transactions refers to the number of KPI data points. These data points need to be sent from a DF to the DS. In our experiment, we compressed each of the four KPIs at the DF with different information loss limits. We also defined the information loss limits for a KPI as ($K \times \sigma$) and increased the K from 0.0 to 1.5.

All the network elements (in our case eNBs) produced the same set of KPI data. Note that if the KPI data comes from different networks, their distribution varies from one network to another. But, if the KPI data comes from one network, their distributions are often constant among elements having similar contexts. In the experiment, we used the data of four KPIs, which all come from the NEs of the same network. We applied QoM compression to all of their time-series data and then summed up the results to the network level. As our test data comes from only one network and we have achieved results with a deterministic process (which produces the same results every time for this data set), we do not observe any variation.

The results of network operator-defined classes are shown in Fig. 3. These results explain the amount of expected data loss



(a) Compression achieved by different KPIs.



(b) Mean square error of the compressed KPI data series.

Fig. 3. Compression achieved and Mean Square Error concerning different information loss limits ($k \times \sigma$) used for the compression.

when applying the compression algorithm within the proposed QoM solution. Fig. 3(a) illustrates the compression achieved in KPI data using different information loss limits ($K \times \sigma$). As we observe from this figure, with the increase in the information loss limits (K), the number of transactions (that refers to the number of KPI data to be transferred) considerably reduces for all KPIs. The descending trends in the shape of all KPIs show the effectiveness of implemented network operator-based class defined approach. The results in this figure illustrate that K is linearly correlated with the number of transactions, that is by increasing the values of K. The increase in K also significantly decreases the number of transactions which translates to considerably compressing of the data points. For instance, by selecting $K = 0.2$, the proposed mPWCA algorithm compresses a considerable amount of data for all of the KPIs. In the case of LTE-5518a (shown by red color) the amount of data reduces almost by half. In addition, setting a bigger value for K compresses the data significantly but at the price of losing more information. However, the compression reduction rate after $K = 0.6$ becomes almost stable.

Fig. 3(b) shows the variation of mean squared error (MSE) of the compressed time-series data. This error is expressed as the percentage of the corresponding time-series median value. As depicted in the figure, with the increase in the information loss limit (i.e., increasing values of K), all of the KPIs illustrate a small increasing trend by the MSE metric. For each KPI at each K value, the amount of MSE increases slightly. When $K = 0.2$, the compression rate is low, and accordingly when $K = 1.2$, the compression rate is the highest. Indeed, when the value of MSE is high the accuracy of the proposed solution is low, i.e., it efficiently compresses large amounts of data but loses the information. For example, if we select K = 1.2, the data can be compressed almost by 100%, leading to high MSE values and information loss for some KPIs. However, in this case, some other KPIs can still carry information that might be feasible for some monitoring tasks.

The results in the figures depict that the best value for K would be 0.6 as it guarantees efficient compression, i.e., a considerable amount of data reduction by the implemented algorithm (mPWCA). Indeed, this best value ensures optimal performance for the QoM solution. The results also approve the effectiveness of the implemented mPWMC algorithm in the QoM solution. These results therefore could be used by network operators, for instance, to decide the information loss limit definitions (i.e., setting the value of K) for different QoM classes, as $K = 0.6$ would be the best limiting number in this case.

2) Data-Driven QoM Class Definition: Due to the different distribution behaviors of KPI data in the real cellular networks, we use the Absolute Consecutive Deviation σ_A as the model to define information loss limits for the QoM Classes. Thus, for the set of time series data $\mathcal{X}$, such that $\mathcal{X} = (x_{(1)}, x_{(2)}, \dots, x_{(k)})$, where $x_{(k)} \in \mathcal{R}$ is the value observed by the real-world process at any time point k, we define a series of absolute consecutive deviation σ_A as $\sigma_A = (|x_{(1)} - x_{(2)}|, |x_{(2)} - x_{(3)}, |\cdots\cdots|, x_{(k-1)} - x_{(k)}|)$ and observe its value distribution within each transferred time series over the training period.

As the statistical metric, the σ_A can be used for determining information loss limits for the QoM classes. The σ_A depicts the changes and variations in the data. Thus, it enables the operator to decide which deviations between consecutive KPI values are insignificant to be removed in the compression and the deviations, which are significant to be reported to the consumer. In addition, the observation of value distribution of each σ_A also requires only a lightweight computation, which makes it suitable to be performed at the MEC platforms. To this end, in our implementation, we applied σ_A as the metric to define the information loss limit (shown by $\mathcal{I}_{loss}$ in Algorithm 1) and defined the following three QoM classes:

- *Class Exact:* Information loss limit was set to 0.
- *Class Optimized:* Information loss limit was set to 25^{th} percentile of σ_A of the training data.
- *Class Sharp:* Information loss limit was set to 50^{th} percentile of σ_A of the training data.

This data-driven QoM class definition method can be further enhanced in the future. It is possible to use lightweight ML methods to increase the accuracy of the method by identifying additional redundancies and correlations from the time series. For example, unsupervised clustering of M-Plane data could be used to identify repeating system states, where the deviations between consecutive values can vary. One such method could be K-Means clustering algorithm, which is one of the most popular unsupervised learning algorithms [32]. Indeed, the K-Means clustering algorithm can group unlabeled dataset instances into clusters based on similar attributes. In addition, the K-means clustering algorithm is a compressed sensing algorithm that can compress real-time and large datasets [33].

TABLE II
THE COMPRESSION (C) AND ERROR (E) PERCENTAGES FOR DIFFERENT KPIS ACHIEVED USING DATA-DRIVEN QOM CLASS COMPRESSIONS

	Class Exact		Class Optimized		Class Sharp	
KPI ID	**C(%)**	**E(%)**	**C(%)**	**E(%)**	**C(%)**	**E(%)**
LTE_5017a	63.78	0.00	63.78	0.00	63.78	0.00
LTE_5178a	37.66	0.00	37.66	0.00	53.15	0.28
LTE_5518a	6.3	0.00	38.92	3.24	71.35	11.49
LTE_5520a	4.5	0.00	36.22	5.82	52.97	12.27

Next, we train the DFs for the duration of two weeks and start the compression of the KPI series. In our implementation, we recorded the number of KPI transactions when the compression started to find compression gains. Table II presents the overall compression results including the compression gain and the error rate (MSE) achieved in compression of KPI data for the three QoM classes. In the results, KPI's MSE is expressed as the percentage of KPI's median value.

Let us study compression gains given in Table II. Let's consider the real-world example that we studied earlier in the second paragraph in Section V-D, where after *KPI calculation*, data volumes reduced from $280 \times x$ MB/s to $28 \times x$ MB/s $= V$. Let us now consider all of the elements in a network. Let us assume that in the network, we need the precise data of 10% of the NEs. In this case, the QoM class Exact is the accurate class to be applied. In the network for 40% of the NEs, it is accurate enough to use the QoM class *Optimized*. For the rest of NEs which are auxiliary, it is good enough to apply the class *Sharp*. Let us assume further that the types of analyzed KPIs represent the types of all KPIs of the NEs which are equally distributed. Thus, we can use the average of compression gains in each class to estimate the achieved compression in KPIs of NEs, to which the class has been applied. Hence, considering Table II as an example, the compression gain for class *Exact* is $(63.78 + 37.66 + 6.3 + 4.5)/4 \approx 28$. Correspondingly, compression gains for classes *Optimized* and *Sharp* equal approximately to 44 and 60 correspondingly. As a result, we achieve straight forward estimation for a data volume reduction of KPI data V: $(1 - 28/100) \times (10/100) \times V + (1 - 44/100) \times (40/100) \times V + (1 - 60/100) \times (50/100) \times V = 0.5 \times V$. Thus, the amount of transferred data gets down to $14 \times x$ MB/s from the original $280 \times x$ MB/s, which gives us an estimated 95% compression gain from the original data rate.

E. QoM Resource Consumption

Our proposed QoM solution is designed to be installed at the network edges at MEC platforms [34], which are known to have limited CPU and memory resources [35]. Thus, running applications which demand heavy computational resources may violate the MEC environments. To this end, we evaluate the CPU consumption and the memory utilization of

TABLE III
CPU UTILIZATION AND AVAILABLE MEMORY OF DATA FETCHER WITH 200 KPI SUBSCRIPTIONS

	With QoM		Without QoM
	Train	Compress	-
Mean CPU utilization	25.5%	28%	22%
Median CPU utilization	25%	27%	21%
Available memory	$\simeq$ 89 MB	$\simeq$ 80 MB	$\simeq$ 240 MB

the QoM solution. In our experiment which we explained in Section V-A, we measured the computational resource consumption by making two subscriptions from the NEs. The first subscription streamed the resource consumption of the DF and the second subscription streamed the KPI data (calculated by the DF from M-Plane raw data). Then, the DFs were set to compute and publish the increasing number of KPIs in each iteration, and also record the consumed computational resources during each iteration from the DFs. The resource consumption of the DFs was also measured using a Linux utility named Collectl [36]. Then, to study the computational resource utilization of the QoM, we run our test-bed two times. Without implementing QoM solution and when implementing the QoM as described in the followings:

Without QoM Setup: In this experiment, we monitored the resource consumption of DFs while the QoM solution was not implemented. The DFs were assigned to *i*) Fetch KPI equation from QoM-DB (Once at the start of subscription), *ii*) Fetch necessary counters from M-Plane data, and *iii*) Calculate KPIs and publish KPIs data to the DS.

With QoM Setup: In this experiment, we monitored the resource consumption of DFs while the QoM concept was implemented. We replayed the M-Plane data for the duration of 4 weeks. We further divided the experiment into two phases: 1) Training phase and 2) Compression phase. The training phase is required if in QoM-DB, the QoM class definitions are not defined by the network operator. During the training phase, the DFs learn the QoM class definitions and they don't perform compression. In our experiment, the training phase lasts until DF analyses M-Plane data of at least 2 weeks and learns QoM class definitions. After the training phase, the compression phase starts in which KPI data is compressed. In our experiment, we recorded the computational resource consumption by the DFs in both training and compression phases as follows. For both phases, the DF was assigned to *i*) Fetch the KPI equation from QoM-DB and *ii*) Fetch necessary counters from the M-Plane data for the KPI computation. In addition to these, for the training phase, the DF was also assigned to learn QoM class definitions and publish KPI data without compression. The compression phase is also assigned to compress KPI as per the defined QoM class specified by the QoM-DB. If definitions were not available by QoM-DB, the DF fall-backed to QoM class definitions learned by the DF and then publishes the compressed KPI data to the DS.

Results of Resource Utilization: Table III presents the performance results of CPU utilization and memory utilization measured during the experiment (in both cases of with and without implementing the QoM solution).

The results for without setting up the QoM showed that the CPU utilization increased almost linearly with KPI subscriptions. With 200 KPI subscriptions, the mean and median CPU utilization was around 22% and 21%, respectively. A relatively high CPU utilization peak was seen in each iteration when KPI subscription was started, which was taken as start-up overhead and was not considered as CPU utilization. With 200 KPI subscriptions and while performing *KPI calculations*, results show that a considerable amount of memory is consumed, such that approximately 240 MB of memory becomes available from the CPU's memory capacity of 800 MB.

The results of the experiment when setting up the QoM in both the training and compression phase showed that the CPU utilization and memory consumption at the DF was higher compared to experiments without the QoM setup. As presented in Table III, when implementing the QoM, available memories in training and compression phases were 89 MB and 80 MB, respectively. While, without implementing QoM, the available memory was 240 MB. Similarly, the mean CPU utilization when implementing QoM in training and compression phases were 25.5%, and 28%, respectively. When the QoM was not implemented, the CPU utilization was lower and the available memory decreased with the increase in KPI subscriptions.

These results are indeed natural since, in both experiments, the operation uses considerable amounts from the available resources like around 560 MB memory without implementing QoM, and 711 and 720MB when training and applying the QoM setup, respectively. In addition, the results are acceptable because of the availability of resources needed for running the QoM solution.

F. Comparison With the State-of-the-Art

The works in the state-of-the-art implement the traditional compression methods and the lossy compression techniques to use the available resources efficiently. These studies use different datasets, each having different volumes and have varying types and different data formats. While some studies compare the traditional compression methods, others implement ML algorithms to optimize resource consumption. While the ML algorithms used in the studies are not similar, some works consider CPU and while others use GPU to conduct a performance evaluation. Indeed, comparing the performance results of the studies in the state-of-the-art with the results obtained in this paper seems inappropriate. Because the datasets and methodologies applied in the literature are dissimilar to our implementation.

For example, the work in [17] uses an artificial intelligence-based lossless compression algorithm and applies a dataset that includes tiny strings with an average length of 160 characters per dataset. As a result, the work reduces the size of data significantly with a compression rate of 84.31%. The study in [18] uses genomic and text datasets and combines a recurrent neural network predictor and lossless compression method; and achieves around 20% reduction over Gzip on the datasets. The work in [19] uses a reinforcement learning-based approach for compression of time-series data and improves compression ratio by up to 120% (with an average of 50%), compared to compression methods called Gorilla, MO (Middle-Out), and Snappy. Then, the work studies bandwidth utilization of the method using a combined CPU and GPU platform for Gorilla, MO, and Snappy. The results show that the Gorilla and MO are stable with average performances of around 100 MB/s and 1.3 GB/s, respectively. Snappy's performance varies a lot by reaching around 60M B/s in most cases. Then, the work evaluates the compression performance using GPU and achieves a performance of around 900MB/s.

In our paper, we propose the QoM concept as a solution that uses three classes to minimize the amount of M-Plane data by removing redundant data and data with minimum information value at the network edge. Within the classes, at the network edge, the QoM solution aggregates the data into KPIs and applies a lossy compression to compress data. The implementation results applied to real-world LTE networks KPI data show that the QoM solution achieves a high compression gain for the three classes which apply different loss limits, and needs reasonable CPU and memory resources for operating.

VI. Conclusion

Cellular communication networks are evolving rapidly and appear with an increasing number of network infrastructure and elements. Current network monitoring solutions will face fundamental challenges in the future, if they continue using the conventional approached of monitoring the M-Plane data. While the increasing number of network elements generate more and more data; a significant portion of this data includes redundant or small information content which translates to the least significant and has no value for monitoring. In this paper, we presented the concept of Quality of Monitoring (QoM) as a solution for monitoring the M-Plane data which aggregates the data into KPIs and uses a set of classes to compress data at the mobile edge before monitoring. While the QoM solution uses a set of classes which each has a different information loss limit that can be defined by the network operator or can learn it using data-driven or machine learning methods at the network edge. The QoM then applies a lossy-compression algorithm called mPWCA algorithm to further compress the data. We evaluated the performance of the QoM solution using raw M-Plane data from a live LTE network and computed four KPIs, such that each KPI has a different statistical characteristic. The results show the significant performance of the QoM solution by considerable compression of the M-Plane data at the network edge and efficiently utilizing the network edge resources such as the CPU and the memory.

References

[1] J. Rodriguez, *Small Cells for 5G Mobile Networks*. Hoboken, NJ, USA: Wiley, 2014, pp. 63–104. [Online]. Available: https://ieeexplore.ieee.org/document/8043585

[2] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh, "5G URLLC: Design challenges and system concepts," in *Proc. 15th Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2018, pp. 1–6.

[3] M. Shafi *et al.*, "5G: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 6, pp. 1201–1221, Jun. 2017.

[4] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: Management requirements and challenges," *IEEE Commun. Mag.*, vol. 53, no. 1, pp. 278–285, Jan. 2015.

[5] V. Kojola, S. Kapoor, and K. Hätönen, "Distributed computing of management data in a telecommunications network," in *Proc. Int. Conf. Mobile Netw. Manag.*, 2016, pp. 146–159.

[6] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "Big data analysis-based secure cluster management for optimized control plane in software-defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 27–38, Mar. 2018.

[7] S. Tarkoma, *Publish/Subscribe Systems: Design and Principles*. Hoboken, NJ, USA: Wiley, 2012.

[8] Apache kafka. (2020). *Apache Kafka: A Distributed Streaming Platform*. Accessed: May 27, 2020. [Online]. Available: http://kafka.apache.org/documentation

[9] A. J. Garcia, M. Toril, P. Oliver, S. Luna-Ramirez, and R. Garcia, "Big data analytics for automated QoE management in mobile networks," *IEEE Commun. Mag.*, vol. 57, no. 8, pp. 91–97, Aug. 2019.

[10] G. Soós, D. Ficzere, P. Varga, and Z. Szalay, "Practical 5G KPI measurement results on a non-standalone architecture," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, 2020, pp. 1–5.

[11] K. Zheng, Z. Yang, K. Zhang, P. Chatzimisios, K. Yang, and W. Xiang, "Big data-driven optimization for mobile networks toward 5G," *IEEE Netw.*, vol. 30, no. 1, pp. 44–51, Jan./Feb. 2016.

[12] L. Pierucci and D. Micheli, "A neural network for quality of experience estimation in mobile communications," *IEEE MultiMedia*, vol. 23, no. 4, pp. 42–49, Oct.–Dec. 2016.

[13] E. Grigoriou, T. Saoulidis, L. Atzori, V. Pilloni, and P. Chatzimisios, "An agent-based QoE monitoring strategy for LTE networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–6.

[14] R. I. Jony, A. Habib, N. Mohammed, and R. I. Rony, "Big data use case domains for telecom operators," in *Proc. IEEE Int. Conf. Smart City/SocialCom/SustainCom (SmartCity)*, 2015, pp. 850–855.

[15] F. Ahmed, J. Erman, Z. Ge, A. X. Liu, J. Wang, and H. Yan, "Monitoring quality-of-experience for operational cellular networks using machine-to-machine traffic," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2017, pp. 1–9.

[16] J. Park, H. Park, and Y.-J. Choi, "Data compression and prediction using machine learning for industrial IoT," in *Proc. IEEE Int. Conf. Inf. Netw. (ICOIN)*, 2018, pp. 818–820.

[17] M. Abedi and M. Pourkiani, "AiMCS: An artificial intelligence based method for compression of short strings," in *Proc. IEEE 18th World Symp. Appl. Mach. Intell. Informat. (SAMI)*, 2020, pp. 311–318.

[18] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa, "DeepZip: Lossless data compression using recurrent neural networks," in *Proc. Data Compression Conf. (DCC)*, 2019, pp. 575–575.

[19] X. Yu *et al.*, "Two-level data compression using machine learning in time series database," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, 2020, pp. 1333–1344.

[20] T. T. Vu, D. T. Ngo, M. N. Dao, S. Durrani, D. H. Nguyen, and R. H. Middleton, "Energy efficiency maximization for downlink cloud radio access networks with data sharing and data compression," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 4955–4970, Aug. 2018.

[21] A. Clemm and R. Wolter, *Network-Embedded Management and Applications: Understanding Programmable Networking Infrastructure*. New York, NY, USA: Springer, 2012.

[22] P. Kumpulainen and K. Hätönen, "Local anomaly detection for mobile network monitoring," *Inf. Sci.*, vol. 178, no. 20, pp. 3840–3859, 2008.

[23] P. Kumpulainen, "Anomaly detection for communication network monitoring applications," Ph.D. dissertation, Tampere Univ. Tech., Tampere, Finland, 2014.

[24] HP. (2013). *HP Cloud System*. [Online]. Available: https://www.hpe.com/us/en/integrated-systems/cloud-system.html

[25] OpenStack. (2010). *OpenStack Cloud System*. [Online]. Available: https://www.openstack.org/

[26] A. C. Cullen and H. C. Frey, *Probabilistic Techniques in Exposure Assessment: A Handbook for Dealing With Variability and Uncertainty in Models and Inputs*. New York, NY, USA: Springer, 1999.

[27] A. K. Gupta and S. Nadarajah, *Handbook of Beta Distribution and Its Applications*. Hoboken, NJ, USA: CRC Press, 2004.

[28] A. Kaur, N. S. Sethi, and H. Singh, "A review on data compression techniques," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 5, no. 1, pp. 1–8, 2015.

[29] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proc. IEEE Int. Conf. Data Min. (ICDM)*, 2001, pp. 1–8.

[30] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Trans. Database Syst.*, vol. 27, no. 2, pp. 188–228, 2002.
[31] I. Lazaridis and S. Mehrotra, "Capturing sensor-generated time series with quality guarantees," in *Proc. IEEE 19th Int. Conf. Data Eng.*, 2003, pp. 429–440.
[32] K. P. Sinaga and M.-S. Yang, "Unsupervised *k*-means clustering algorithm," *IEEE Access*, vol. 8, pp. 80716–80727, 2020.
[33] N. Keriven, N. Tremblay, Y. Traonmilin, and R. Gribonval, "Compressive *k*-means," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2017, pp. 6369–6373.
[34] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," ETSI, Sophia Antipolis, France, White Paper, 2015.
[35] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.
[36] *Collectl Performance Monitoring Tool*. Accessed: Nov. 18, 2019. [Online]. Available: http://collectl.sourceforge.net/

Naser Hossein Motlagh received the D.Sc. degree in networking technology from the School of Electrical Engineering, Aalto University, Finland, in 2018. He is a Postdoctoral Researcher with the Nokia Center for Advanced Research, Department of Computer Science, University of Helsinki, where he was a Postdoctoral Fellow with Helsinki Center for Data Science Program, Helsinki Institute for Information Technology. His research interests include the Internet of Things, wireless sensor networks, environmental sensing, smart buildings, and unmanned aerial and underwater vehicles.

Shubham Kapoor received the M.Sc. degree in computer science from the University of Helsinki, Finland, in 2017. He is currently working as a Lead System Architect with the Swiss Institute of Bioinformatics, Basel, Switzerland. He has coauthored a couple of research publications and IPRs. His research interests include secure cloud computing, federated computation, machine learning, and data analytics.

Rola Alhalaseh received the Bachelor of Engineering degree in computer engineering from the Princess Sumaya University for Technology, Amman, Jordan, in 2011, and the M.Sc. degree in data science from the University of Helsinki, Finland, in 2018, where she is currently pursuing the Doctoral degree with the Department of Computer Science. Her research interests include the Internet of Things, Web applications security, neural networks, tactile Internet, machine learning, and edge computing.

Sasu Tarkoma (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Helsinki in 2006, where he is a Professor of Computer Science and the Head of the Department of Computer Science. He is a Visiting Professor with the 6G Flagship, University of Oulu. He has authored four textbooks and has published over 250 scientific articles. He holds ten granted U.S. patents. His research interests include Internet technology, distributed systems, data analytics, and mobile and ubiquitous computing.

Kimmo Hätönen received the Ph.D. degree in computer science from the University of Helsinki, Finland, in 2009. He is a Principal Scientist with Nokia Bell Labs, Espoo, Finland, where his responsibility areas include all sorts of data science applied to network and IoT data; currently especially exchange and analytics of streaming data. He has studied data, information, and knowledge related issues for over 30 years in research organizations with Nokia and University of Helsinki. He has authored and coauthored several tens of publications and been involved in introducing several methods, tools and products for communication network data analysis.