Adaptive Performance Analysis in IoT Platforms

Clovis Anicet Ouedraogo, Samir Medjiah, Christophe Chassot, Khalil Drira, Jose Aguilar

Adaptive Performance Analysis in IoT Platforms

Clovis Anicet Ouedraogo, *Graduate Student Member, IEEE*, Samir Medjiah *Member, IEEE*, Christophe Chassot, Khalil Drira, and Jose Aguilar *Member, IEEE*

***Abstract*—In this paper, we consider the problem of identifying multiple bottlenecks (a.k.a bottleneck analysis) in IoT Service Platforms. For QoS-constrained applications, IoT Platforms have grown in complexity with non-stationary workloads and inter-task dependencies created by data flows crossing the platform's nodes. These factors create multiple simultaneous "bottlenecks" (a bottleneck expresses overload in terms of request processing time on a given node, and contributes to QoS degradation). Multi-bottlenecks are non-trivial to analyze since they may escape typical assumptions made in classic performance analysis, such as analysis based on queuing theory models. Solving this analysis problem requires real-time collection and analysis of data that can be massive, and as a result, induce negative impacts on the performance of the NFV-based IoT Platform (NIP) (e.g., use of bandwidth, computing resource, and storage resource). Therefore, it needs to be adapted to the strict minimum allowing effective analysis. We build an adaptive performance analysis method that optimizes bottlenecks' identification for a monitoring overhead budget associated with the different available metrics. Instead of systematically collecting all the NIP metrics, the proposed process determines the best subset of metrics to consider for the efficiency of the performance analysis. The conducted experiments on a practical use case show that the proposed method exhibited high performances of the bottleneck analysis process, in the presence of different bottleneck types and durations, with very few false positives and false negatives.**

***Index Terms*—Internet of Things (IoT), Machine Learning (ML), Network Functions Virtualization (NFV), Performance Bottleneck Analysis, Fault Localization.**

ACRONYMS

IoT	Internet of Things
MIB	Multiple bottlenecks identification
MLC	Multi-label Classification
NFV-I	Network Functions Virtualization Infrastructure
NFV	Network Functions Virtualization
NF	Network Functions
NIP	NFV-enabled IoT Platform
QoS	Quality of Service
SOMS	Simple Overhead-sensitive Metrics Selection

I. INTRODUCTION

IOT ecosystem is evolving from dedicated IoT platforms[1], sketched for the requirements of a given IoT application domain, to integrated shared platforms such as oneM2M [1]. These shared platforms simultaneously support multiple application domains, such as smart grids, connected vehicles, home automation, public safety, and e-health. The next generation of the IoT ecosystem will connect billions of devices with extreme heterogeneity in terms of resources (e.g., CPU and RAM) capacities and limitations, and software and hardware technologies for connectivity, processing, and storage.

Virtualization is a crucial technique for the successful design and implementation of IoT platforms that handle heterogeneity. The virtualization technologies pushed by Cloud Computing are now reaching the networking domain. The Network Function Virtualization (NFV) approach [2] has been introduced to solve the challenges induced by conventional middlebox technologies, such as massive, costly deployments and complex management requirements, overloads and failures, and limited upgradability. The NFV technology tackles these challenges through the virtualization of Network Functions (NFs) on Cloud-enabled infrastructures [3]. NFV allows the instantiation, configuration, and duplication of Virtualized Network Functions (VNFs) in various locations according to the NIP operator needs, which avoids installing new physical equipment [4].

In general, meeting the strict QoS requirements of IoT applications through effective performance diagnosis remains an inescapable challenge [5]. Indeed, the integration of IoT Platforms, traditionally vertical to shared horizontal platforms, gives rise to performance bottlenecks, challenging to detect and mitigate. A bottleneck is a resource or an application component that limits the performance of a system [6]. [7] describes a bottleneck component as a potential root cause of undesirable performance behavior caused by a limitation (e.g., saturation) of some significant system resources associated with the component. Performance diagnosis is a two-step process: we first seek to detect QoS violations, and secondly determine the causes of this violation,i.e., the bottlenecks in terms of performances (e.g., CPUs saturations) associated with the resource of the NIP responsible for the assumed violation. This second step is known as the performance analysis step. This work focuses on this second step, when a violation has already been detected using, for instance, methods presented in [8]–[11]. Solving this analysis problem requires real-time collection and analysis of data characterizing the NIP's performance. This data collection can be massive, and as a result, can induce negative impacts on the performance of the NIP (e.g., use of bandwidth, computing resource, and storage resource) and on the reasoning time of the analysis method. Because of recent advances in the industry and the literature, we can draw the following conclusions. First, there are over 80 types of metrics available to monitor in a NIP deployed on a public cloud

This work was supported by Continental Digital Service France (CDSF) in the framework of the eHorizon Project on Connected Vehicles.

Clovis Anicet Ouedraogo, Samir Medjiah, Christophe Chassot, and Khalil Drira are with LAAS-CNRS (email: {ouedraogo, medjiah, chassot, drira}@laas.fr).

Jose Aguilar is with the Universidad de Los Andes (Venezuela) and Universidad EAFIT (Colombia) (email: aguilar@ula.ve).

[1]An IoT platform, also known as IoT software platform or IoT middleware implements an IoT architecture providing a variety of services to an IoT application, such as device connectivity, device management, data transfer, data management, data analytics, security, and visualization.

such as AWS (using EC2 VMs)[2]. Second, these metrics induce non-negligible monitoring overhead. The monitoring overhead is the amount of additional usage of resources by a monitored execution of a program compared to a regular (unmonitored) execution of the same program. In this case, resource usage encompasses the utilization of CPU, memory, I/O, and so on. Monitoring overhead concerning execution time is the most commonly used definition of overhead. In an ideal scenario, the overhead of collecting data increases with a constant value per access. Following [12], three causes of overhead are common to most application-level monitoring frameworks (i) instrumentation of the system under monitoring, (ii) collection of monitoring data (iii) writing or transferring the collected data. Finally, these metrics have different impacts on the efficiency of the analysis of bottlenecks [13]. In this context, and considering a maximum overhead not to be exceeded (i.e. budget), we formulated the following research question:

> *"How to determine the metrics that maximize the efficiency of NIP performance analysis and lead to a minimum cost given an allocated monitoring overhead budget?"*

By answering this question, we seek to build an adaptive method that optimizes the bottlenecks analysis performance regarding a monitoring overhead budget associated with the different available metrics.

Contributions. The significant contributions of this paper are summarized below.

- We model the problem of multiple bottlenecks identification (MIB) in NIPs as a Multi-Label Classification (MLC) problem, and we propose a classification of main categories of bottlenecks in NIPs;
- We propose an Overhead-sensitive Metrics Selection Algorithm to answer the research question. This algorithm is a heuristic that selects a subset of relevant metrics for a given monitoring overhead.
- We build a virtualized platform prototype implementing the experimental testbed to gather a training dataset. We designed the testbed to provide a training set that is representative of a real-world situation.
- We develop different supervised Machine-Learning (ML) algorithms to perform the identification of the bottlenecks. We numerically evaluate these MIB models, using the collected data in terms of: subset accuracy, coverage error, sensitivity, and specificity.
- We implemented the proposed Overhead-sensitive Metrics Selection Algorithm (SOMS) to find which metrics should be considered for the efficiency of the NIP analysis while optimizing the performance of the MIB model, not to label it as positive a negative sample and evaluate its performance. Our numerical results show that 81 metrics give the maximum precision (84%) of the MIB model. Up to 83% can be achieved even with a relatively limited metrics subset of 22 metrics.

The implementations and the experiment dataset are available at https://github.com/couedrao/APA4NIP.

[2]https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/metrics-collected-by-CloudWatch-agent.html

Organization. The remainder of the paper is structured as follows. Section II discusses the related work. Section III describes how the proposed approach is implemented in a real scenario and a motivating use case. Section IV details the system model. Section V describes the proposed methodology to tackle the multiple bottlenecks identification problems in NIPs with an allocated monitoring overhead budget. Section VI presents the experimental setup. Section VII is devoted to the evaluation of the proposed approach. Finally, our work results, their limits, and future work are discussed in the conclusion section.

II. Related work

Several fields, such as traditional IP Networks [14], Cloud Computing [15], and Big Data [16], consider the multiple bottlenecks identification problems. Moreover, regarding NFV or (NFV-enabled Platforms), most of the existing works consider the *fault detection problem* or the *fault recovery problem* in the fault management framework (see [17] for more detail). Nevertheless, few works deal with the *fault localization problem* (i.e. bottlenecks identification problem). In this paper, since we only aim to contribute to this domain in the IoT context, we consider the reference contributions made in the literature. In the following, we present a literature review analysis on NFV-enabled Platforms, including IoT which is an essential aspect of the proposed work.

Sauvanaud et al. propose, in [18] and [19], an approach to detect Service Level Agreements (SLAs) violations and initial symptoms of SLAs violations. In their approach, authors consider a fault injection tool to train a supervised learning algorithm to pinpoint the root anomalous VNF causing SLA violations. Experiments were performed in a virtual IP Multimedia Subsystem (Clearwater) testbed. Similarly, Gonzalez et al. propose, in [20] an offline machine learning-based method for the automatic identification of dependencies between system events, enhanced with summarization, operations on graphs, and visualization that help network operators identify the root causes of errors. Cui et al. explain, in [21] an analytic model based on the Cyclic Temporal Constraint Network (CTCN), which aims at the fault analysis of cyclic computer networks using temporal information. The proposed model relies on a given "predetermined candidate fault causes" to determine the most likely fault cause(s) with a given time interval(s) of occurrence(s). Cotroneo et al. describe, in [22] an approach to detect problems affecting the QoS, such as overload, component crashes, avalanche restarts, and physical resource contention in production NFV services. The method infers the service health status by collecting metrics from multiple elements in the NFV service chain and by analyzing their (lack of) correlation over time. Experiments were performed on an NFV-oriented Interactive Multimedia System. Cotroneo et al. propose, in [23] a dependability benchmark to support NFV providers at making informed decisions about which virtualization, management, and application-level solutions can achieve the best dependability. Authors define the use cases, measures, and faults to be injected. Their experiments, conducted in an IMS case study, suggest that the container-based configuration can be less dependable than

the hypervisor-based one, and point out which faults NFV designers should address to improve dependability. Additionally, authors describe in [24] potential guidelines for evaluating the reliability of NFV Infrastructures (NFVIs), intending to verify whether NFVIs satisfy their reliability and performance requirements, even in the presence of faults. The described guidelines are practices to be followed in terms of inputs, activities, and outputs. These practices are intended to be conducted by NFV designers that want to evaluate the reliability of their NFVI against quantitative performance, availability, and fault tolerance objectives, and to get precise feedback on how to improve its fault tolerance. Zhang et al. explain, in [25] a deep learning-based fault analysis method to predict a virtual network's failure. The proposed deep learning model enables the earlier failure prediction by using a Long Short-Term Memory (LSTM) network, which discovers the long-term features of the network history data. Mariani et al. propose, in [26] a fault localization approach based on machine learning and graph theory. In the proposed approach, the machine learning models are trained with correct executions only and compensate for the inaccuracy that derives from training with positive samples, the outcome of machine learning techniques with graph theory algorithms. Pfitscher et al. propose, in [27] a model based on queuing networks theory to quantify the guiltiness of each VNF on degrading the performance of a network service. A hybrid algorithm based on linear regression and neural networks is also introduced to adjust the model's parameters according to the environment particularities, such as the type and number of VNFs in the service. Experimental evaluations confirm the ability of the model to detect bottlenecks and quantify performance degradations. Tola et al. describe, in [28] an approach to estimate the end-to-end NFV-deployed service availability, and present a quantitative assessment of the network factors that affect the availability of the service provided by an NFV architecture. The proposed approach considers a two-level availability model where (i) the low level considers the network topology structure and NFV connectivity requirements through the definition of the system structure function based on minimal-cut sets and (ii) the higher level examines dynamics and failure modes of network and NFV elements through stochastic activity networks. Bouattour et al. propose, in [29] a model to identify the noise source in a virtualized infrastructure. First, an anomaly detection model based on unsupervised learning is proposed to identify the machines that are in an abnormal state in the infrastructure. An investigation of the cause is later achieved by searching, with a supervised learning algorithm, how anomalies are propagated in the system.

In summary, the existing literature lacks attention to NIP from three perspectives. First, to the best of our knowledge, no existing work in NFV-enabled Platforms considers taking into account the fact that multiple bottlenecks may arise among several resources in these platforms (i.e. the multiple bottlenecks identification problems). Second, none of the current studies consider the cost and the differentiated contribution of the metrics that can be used to operate the analysis. Thirdly, no approach takes into account the cost of the analysis (which we discuss here under the term budget). Note that the other works do not address it because in their considered contexts is not necessary; However, in our context (i.e. IoT), the limitation of resources in the node close to objects, this cost cannot be ignored.

In that direction, our contribution's main originality consists of combining several changes in the traditional approach to handle bottlenecks identification problems. The first change (Section V-A) consists of considering that multiple bottlenecks may arise among several resources in NIPs. The second change (Section V-B) consists of considering adapting the monitored metric to the strict minimum that allows practical bottlenecks analysis in NIPs. We use the term "monitoring overhead budget" and "overhead budget" interchangeably in the latter.

III. Preliminaries

In previous work [30], we proposed to enhance the existing approaches using and extending the emerging concept of Virtualized Network Functions (VNF), promoting End-to-End IoT traffic control. From the observation that, within NFV-enabled IoT Platforms (NIPs), the allocated resources are not fully used, we promote deploying traffic control network functions (NFs). Instead of systematically scaling a congested node, we dynamically deploy, on non-bottleneck nodes, additional NFs that exploit the available computing resources and apply a QoS-oriented policy while performing scaling actions on bottleneck nodes. Concerning the considered multi-constrained context, we formulate a multi-objective optimization problem to efficiently plan adequate NFs and scaling actions. The planner based on a Genetic Algorithm, called QoS4NIP (QoS for NFV-enabled IoT Platforms), iteratively generates solutions to the identified multi-objective optimization problem. Information of the nodes, regarding their status in terms of bottlenecks, were manually provided. In this work, we explore Machine Learning (ML) algorithms to analyze the bottlenecks in IoT Platforms.

A. Considered Framework

This paper's proposed solution interacts with the NIP's monitoring system, and the configuration enforcement components in a real scenario following the autonomic architecture model of [31]. As Fig. 1 depicts, the following components interact with the considered framework. The *Monitoring System* component [31] collects the details from the managed

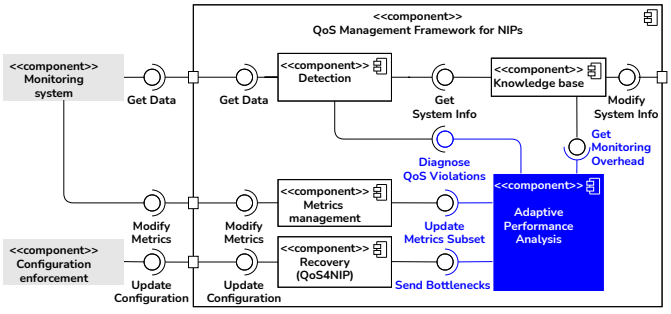

Fig. 1: Considered framework: QoS Management Framework for NIPs

NIP via monitoring agents. The details include data such as topology information, QoS, and performance metrics. The QoS Management Framework retrieves and stores these collected data for analyzing purposes. The *Configuration enforcement* component [31] provides the mechanism to schedule and perform the necessary changes to the system. Once the QoS Management Framework has generated a reconfiguration plan of the system, the planned and executed actions will lead to modify the state of one or more NIP nodes. The following components interact with the QoS Management Framework. The *Knowledge base* component stores the data used by the QoS Management Framework. The knowledge includes topology information, historical logs, metrics, IoT applications information, and allocated overhead budget. The *Detection* component uses complex models, such as time-series forecasting, to detect the violations on IoT applications' QoS. The *Detection* component is continuously invoked and takes the monitoring information as inputs. The output of the *Detection* component is performance data associated with a violation. The *Adaptive Performance Analysis* components analyze the non-trivial dependency in the provided data to analyze the bottlenecks causing a detected violation. This component is invoked by the *Detection* component when it detects a QoS violation (see Section V). The *Metrics management component* is in charge of increasing (or decreasing) the number of metrics to be observed in the NIP. This component is invoked by *Adaptive Performance Analysis* component every time the selected metrics subset is updated. The *Recovery* component (see [30]) determines the set of candidate actions to recover from identified bottlenecks. This paper is focused on the realization of the *Adaptive Performance Analysis* component.

B. Motivating use case

We present in this Section an adaptive performance analysis use case to be considered and evaluated. In this use case, we assume that the NIP service provider wants a flexible trade-off between the efficiency of the analysis and the monitoring overhead. Monitoring induces exchanges and processing of additional data messages that lead to additional consumption of resources (CPU, Memory, Bandwidth). These extra resources have, in fine, to be billed by Cloud/Fog providers. In this paper, we consider the monitoring overhead as an aggregate of all these financial costs. In the Cloud-to-Thing continuum [32], the availability and capacity of resources, namely compute, storage and connectivity, decrease when moving from Cloud to Things. Typically, IoT edge gateways, located near IoT devices, are small devices with limited processing, storage, and connectivity capabilities. In this work, we consider that the monitoring overhead is inversely proportional to the resources that are available on a given node. The monitoring carried out on IoT Edge gateways is thus more expensive than that carried out on nodes in the Cloud. We make the hypothesis that the monitoring overhead increases by 50% at each "level" of node, going from the Cloud level to the IoT device level (e.g. level 1: $cost = X$, level 2: $cost = 1.5 \cdot X$, etc., $costn = (1.5)^n \cdot X$). Expressing this variation in another way, or giving other values, would not call into question the approach proposed in this paper. We consider 3 situations where the allocated overhead budget fluctuates in time: *unlimited* budget, *modest* budget, and *austere* budget. To build the needed dataset to model the machine learning problems, we used a realistic workload to simulate the usage in the NIP. The taxi trips dataset provided by the City of Chicago's open data portal (https://data.cityofchicago.org) is associated with a variable traffic load in the NIP. This load changes during the day because taxi traffic is not the same during a day. As depicted in Fig. 2, we define the following scenarios based on the "Taxi signal count" in the Chicago taxi trips dataset. The overhead budget is unlimited in the first scenario (unlimited overhead budget between 7h-20h). In the second scenario (a modest overhead budget between 5h-7h and 20h-23h), the overhead budget is relatively limited. The overhead budget is severely limited in the third scenario (austere overhead budget between 0h-5h). Below we describe each scenario.

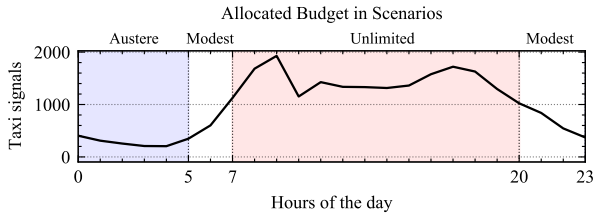

Fig. 2: Chicago Millennium Park taxi signal counts by hour of the day for Monday, February 06, 2017.

Unlimited budget scenario. We first investigate the case where the overhead budget is unlimited. This scenario occurs during the rush hours in Fig. 2 where the taxi signal number exceeds a thousand. During this period, we assume that the NIP service provider wants the efficiency of the analysis at its highest and does not set a limit to the monitoring overhead. Consequently, the best metrics subset that maximizes the efficiency of NIP performance analysis will be selected regardless of the associated overhead. In this scenario, the useless or irrelevant metrics will still be discarded.

Modest budget scenario. Let ω_u be the overhead induced by the selected metric subset in the previous scenario (Unlimited budget scenario). In a second time, we investigate the case where the overhead budget is 50% of ω_u. This scenario occurs during the hours where the taxi signal number is between five hundred and one thousand (see Fig. 2). We assume that the NIP service provider may tolerate an efficiency smaller than in the previous scenario during this period. The NIP service provider's primary concern is a trade-off between the efficiency of the analysis and the monitoring overhead. The result of this scenario is the selection of the best metrics subset that maximizes the NIP performance efficiency of the analysis with a minimum cost compatible with the 50% of ω_u monitoring overhead.

Austere budget scenario Pushing further the second scenario, we analyze the trade-off between the efficiency of analysis and the monitoring overhead in this third scenario. We assume that the NIP service provider may tolerate even less efficiency than in the previous scenario. This scenario occurs during the hours where the taxi signals count is lower

TABLE I: NOTATIONS

Names	Meanings
B	the number of possible bottlenecks
C_t	an observation cycle
$\mathbf{D}$	a multi-bottleneck training set
F_k	a Flow k of messages
h	the hypothesis to optimize
m	the number of samples
M_k	a Message on F_k
N_k	a set of network functions composing a Path$_k$
O_k	the monitoring overhead of every performance metric
$o_{p,n}$	the value of the monitoring overhead associated to the metric p on nf$n$
p	a performance metrics p
P	number of performance metrics
Ψ	the optimization criterion
S	a set of metrics without a metric $\theta_{p,n}$
Θ_k	the decision variable regarding which performance metrics is actually monitored
$\theta_{p,n}$	the value of the decision variable associated to the metric p on nf$n$
X_k	The monitored performance related to F_k during C_t
$x_{p,n}$	the mean value of the time series associated to the metric p on nf$n$ during a Ct
Y	the true Bottlenecks
$\hat{Y}$	the diagnosed Bottlenecks
fn$_j$	the false negative of the j-bottleneck
fp$_j$	the false positive of the j-bottleneck
nf$n$	a NF n in N_k
Path$_k$	a Path k
tn$_j$	the true negative of the j-bottleneck
tp$_j$	the true positive of the j-bottleneck

than five hundred (see Fig. 2). The overhead budget is 25% of ω_u. Consequently, the best metrics subset that maximizes the efficiency of the NIP performance analysis with a minimum cost compatible with the 25% of ω_u monitoring overhead will be selected.

When analyzing this use case, the desired efficiency of analysis is not the same over time. This is why we must adapt accordingly to the monitoring budget.

IV. SYSTEM MODEL

In this section, we propose a model for the considered system. For convenience, Table I lists the main notations.

A. NIP Model

In our work, we handle the NIPs that implement the common reference architectures, such as oneM2M [1]. We consider that NFV-I in the Cloud/Fog/Edge host Virtualized Network Functions (VNF), Application Network Function(ANF) [30] and Physical Network Functions (PNF) offering the NIP service to the IoT Application and IoT devices.

Fig. 3 depicts the NIP model used in this paper. A set of Network Functions (NF) make up this platform. In this ecosystem, the applications send their messages to the nodes of the platform. Then, the latter route them to other nodes or the objects containing the requested resources. For instance, when the IoT Application APP1 sends a message to the NF1 node requesting a resource available on Dev1, the message will then be routed successively to the NF2, NF3, NF4, NF5, and NF6. This application-level routing is done according to the REST architectural style, which most current IoT service providers implement (ex: AWS IoT Core, Microsoft Azure IoT,

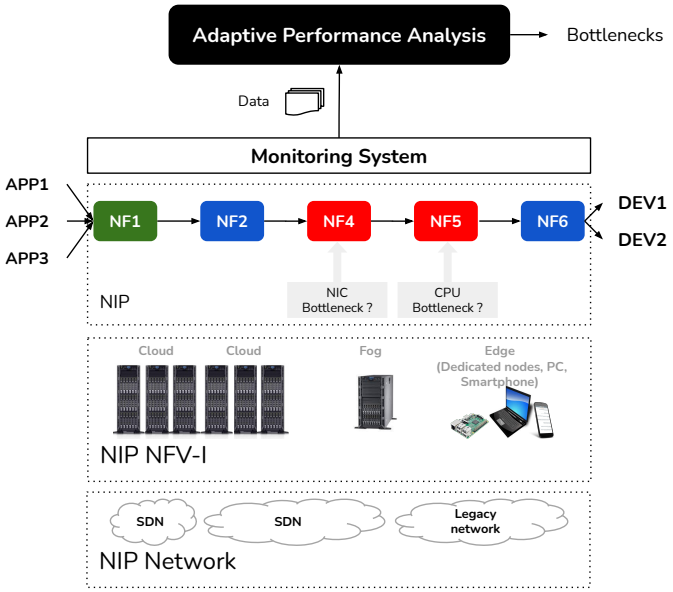

Fig. 3: System Model

oneM2M). To facilitate the presentation of the performance analysis system, we define a NIP to consist of a set of flows $F_1, F_2, F_3, ..F_K$. A flow F_k is a set of M_k successive messages $F_k = \{\text{msg}_1, \text{msg}_2, \ldots, \text{msg}_{M_k}\}$ exchanged between a source and a destination nodes. Each flow F_k will be routed through a predetermined Path$_k$. Path$_k$ is composed of a set of N_k network functions; Path$_k = \{nf_1, nf_2, \ldots, nf_{N_k}\}$. The source and the destination of a flow F_k are denoted as F_{k_S} and F_{k_D}, respectively. Hence, each network function may process several messages during a single observation cycle of C_t.

B. Performance Monitoring Model

For each NF (nf$n$) in the NIP, we propose to monitor P performance metrics (CPU, Disk I/O, etc). The monitored performance related to a flow F_k is denoted X_k.

$$X_k = \begin{pmatrix} nf_1 & nf_2 & \cdots & nf_{N_k} \\ x_{1,1} & x_{1,2} & \cdots & x_{1,N_k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N_k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{P,1} & x_{P,2} & \cdots & x_{P,N_k} \end{pmatrix} \quad (1)$$

where:

$x_{p,n}$= is the mean value of the time series associated to the metric p on node n during a cycle C_t. For simplicity, we consider only the mean value among a wide variety of others statistics extracted from the time series.

All the performance metrics are not necessarily monitored. Indeed, let Θ_k be the decision variable regarding which performance is monitored.

$$\Theta_k = \begin{pmatrix} nf_1 & nf_2 & \cdots & nf_{N_k} \\ \theta_{1,1} & \theta_{1,2} & \cdots & \theta_{1,N_k} \\ \theta_{2,1} & \theta_{2,2} & \cdots & \theta_{2,N_k} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{P,1} & \theta_{P,2} & \cdots & \theta_{P,N_k} \end{pmatrix} \quad (2)$$

where :

$$\theta_{p,n} = \begin{cases} 1 & \text{if the performance metric } p \text{ on nf}_n \text{ is monitored} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Let O_k be the monitoring overhead of every performance (considered in the NIP) on each NF nf_n for a flow F_k.

$$O_k = \begin{pmatrix} nf_1 & nf_2 & \cdots & nf_{N_k} \\ o_{1,1} & o_{1,2} & \cdots & o_{1,N_k} \\ o_{2,1} & o_{2,2} & \cdots & o_{2,N_k} \\ \vdots & \vdots & \ddots & \vdots \\ o_{P,1} & o_{P,2} & \cdots & o_{P,N_k} \end{pmatrix} \tag{4}$$

where:
$o_{p,n}$ is the monitoring overhead of the performance metric p on nf_n.

V. Adaptive Performance Analysis

In traditional computer systems (e.g., as modeled by queuing theory), a typical assumption is that their workloads consist of independent jobs. This assumption, which is valid for old-style batch-oriented processing and interactive users, guarantees the appearance of single bottlenecks for an entire system. Single bottlenecks can be relatively easily identified since they appear as resources reaching saturation. The "independent jobs" model does not hold for NIPs that rely on a different architecture style. Today's NIPs are pipelines of processing components, e.g., web servers, application servers, and database servers, introducing several strong dependencies among components. These dependencies may lead not only to one single bottleneck but potentially to multiple bottlenecks distributed throughout the whole system [7]. Indeed several works such as [32], [33] have considered the question of how to analyze multiple bottlenecks in a single run. We propose to answer this question in this work.

Our proposed method is intended to overcome the limitations described in Section VII. As indicated in the introduction, this work's fundamental objective is to determine which metrics should be considered for the best efficiency of the NIP analysis, given a tolerated overhead budget. First, the proposed method must identify the bottlenecks. This identification's output is human readable and is represented by a binary vector $\mathcal{Y}$ to describe the presence or not of bottlenecks in the Flow F_k. Second, the proposed method identifies the most relevant metrics to collect in a given scenario (i.e. with a tolerated overhead budget). To this end, an approach built on supervised learning is used. Based on an MLC algorithm, a feature selection wrapper algorithm (Simple Overhead-sensitive Metrics Selection – SOMS) is used to measure the relevance of a given metric (i.e. its role in determining the bottlenecks).

Some definitions need to be made clear to understand the proposed approach. Based on [34], we classified metrics into three disjoint categories: strongly relevant, weakly relevant, and irrelevant. Let $g(\cdot)$ be the SOMS algorithm learning hypothesis and let $S = \Theta_k - \{\theta_{p,n}\}$ be a set of metrics without a metric $\theta_{p,n}$. These categories of relevance can be formalized as follows.

Strong relevance: A metric $\theta_{p,n}$ is strongly relevant iff

$$g(\Theta_k) > g(S) \tag{5}$$

Weak relevance: A metric $\theta_{p,n}$ is weakly relevant iff

$$\begin{gathered} g(\Theta_k) = g(S) \text{ , and} \\ \exists S' \subset S \text{ , such that } g(\Theta'_k) > g(S') \end{gathered} \tag{6}$$

Irrelevance: A metric $\theta_{p,n}$ is irrelevant iff

$$\forall S' \subseteq S, g(\Theta'_k) \leq g(S') \tag{7}$$

The strong relevance indicates that the metric is always necessary for an optimal subset; it cannot be removed without affecting the efficiency of the analysis. Weak relevance suggests

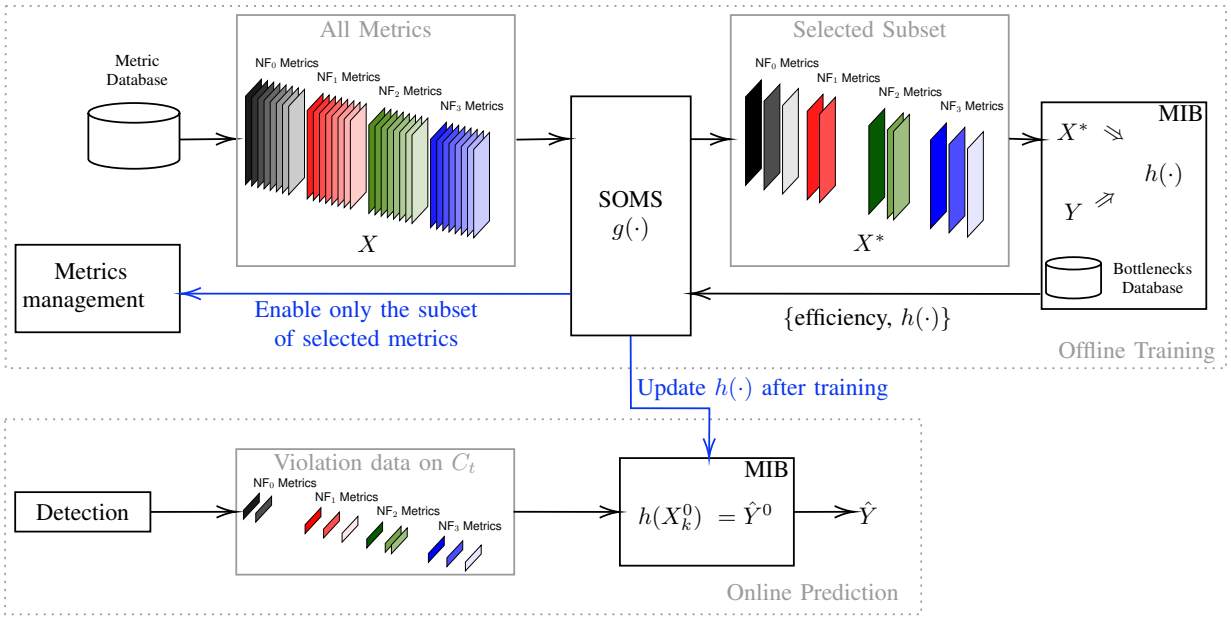

Fig. 4: Adaptive Performance Analysis Method

that the metric is not needed but may become necessary for an optimal subset at certain conditions. Irrelevance indicates that the metric is not needed at all. An optimal subset should include all strongly relevant metrics, none of irrelevant metrics, and maybe a subset of weakly relevant metrics.

As depicted in Fig. 4, the proposed methodology is as follows.

Off-line Training. In a supervised learning approach, there is a training step. In this step, the Adaptive Performance Analysis infers two functions $g(\cdot)$ (SOMS) and $h(\cdot)$ (MIB) from the training dataset. In the SOMS Algorithm (see Algorithm 1) each new subset is used to train and test a MIB model. Training a new model for each subset is computationally intensive, but provides the best performance [35] and is the only approach directly applicable to multilabel dataset [36]. After training SOMS Algorithm, the found optimal subset is sent to the *Metrics management* component, and only these metrics will be active for the online prediction step. The associated $h(\cdot)$ is also transferred to the online MIB.

Online Prediction. Once the optimal subset is found in the training step; the predictions are made online. When the *Detection* component catches a QoS violation, the corresponding data on the violation is gathered, and the online MIB is invoked to identify the bottlenecks.

A. Multiple bottlenecks identification (MIB)

Multiple bottlenecks identification (or Fault isolation) in IoT platforms is challenging because of the interactions between different network entities (e.g., wireless sensors, gateways) and protocols. The multiple bottlenecks identification problem can be viewed as an MLC problem in that we try to categorize the detected QoS violations into one or several of the existing bottleneck classes carefully arranged by an expert. In machine learning, a typical classification problem aims to extract models from training data with known class labels to predict the test data categories of which the class labels are unknown.

To formally describe the MLC problem, suppose $\mathcal{X} = \mathbb{R}^{P\times N_k}$ denotes the $(P \times N_k)$-dimensional instance space, and $\mathcal{Y} = y^1, y^2, \ldots, y^B$ denotes the bottleneck space with B possible bottlenecks. We define y^i as a possible bottleneck (property of the IoT platform node) that may have caused the detected QoS violations. Let a multi-bottleneck training set $\mathbf{D} = \{(X_k^i, Y^i)|1 \leq i \leq m\}$ be independently and randomly drawn according to an unknown probability distribution $\mathbf{P}(X, Y)$ on $\mathcal{X} \times \mathcal{Y}$. For each multi-bottleneck example (X_k^i, Y^i), $X_k^i \in \mathcal{X}$ and $Y^i \subseteq Y$ is the set of bottlenecks associated with X_k^i. The goal in MIB model is therefore to induce from $\mathbf{D}$ a hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$ that optimizes a criterion $\Psi(Y, \hat{Y})$ when it provides a vector of relevant bottlenecks $\hat{Y} = h(X_k^0) = (h_1(X_k^0), h_2(X_k^0), \ldots, h_B(X_k^0))$ for any unseen instance X_k^0.

Remark that the criterion Ψ is not necessarily unique. Indeed several criteria were retained to evaluate the MIB model (see Section VII-A).

B. Simple Overhead-sensitive Metrics Selection (SOMS)

In this section, to answer which metrics subset should be considered for the efficiency of the NIP analysis, we present a Simple Overhead-sensitive Metrics Selection (SOMS). The proposed SOMS Algorithm selects a subset of relevant metrics for a given overhead budget. Formally, SOMS solves the following optimization problem:

$$\begin{aligned} \text{optimize} \quad & g = \frac{1}{m}\sum_{i=1}^{m}\Psi(Y^i, h(X_k^i \odot \Theta_k)) \\ \text{subject to} \quad & \omega_{\text{admin}} \geq \omega \end{aligned} \tag{8}$$

where:

- ω_{admin} is the overhead budget tolerated by the NIP administrator for a flow F_k.
- ω (see Eq. 9) is the total monitoring overhead for a flow F_k,

In Eq. 8:

$$\omega = \sum_{p=1}^{P}\sum_{n=1}^{N_k}(\Theta_k \odot O_k)_{p,n} \tag{9}$$

The optimization problem described in Equations 8 and 9 stipulate that we want to optimize the objective function g which represents how bad are the predictions ($h(X_k^i \odot \Theta_k)$) compared to the Y^i. While solving g, the overhead budget tolerated (ω_{admin}) must remains greater or equal to the total monitoring overhead (ω).

The Overhead-sensitive Metrics Selection is an optimal subset selection problem (aka best subset selection). In general, this problem (i.e optimal subset selection) is nonconvex and is known to be NP-hard [37]. For this problem, we propose a heuristic based on the Forward Sequential Selection search strategy [38] that has been proven to constitute an efficient method to provide suitable near-optimal solutions in a short amount of time (see Section VI). This strategy follows a wrapper approach [39]. The general workflow of the SOMS Algorithm is presented in Algorithm 1.

From lines 1 to 4, Θ_k is initialize with a $P \times N_k$ Zero matrix, r is initialize with 0, and set of best metric S_b is set to $\emptyset$. Then, until the set of all metrics is reached, the Algorithm explored different combinations of metrics (Line 5). In line 6, the Algorithm initializes the set of evaluations of different combinations to $\emptyset$. For each possible combination, from lines 7 to 10, add the p metric on node n, evaluate the combination. In line 12, find the best combination. From lines 13 to 17, was this combination, the best of its size found so far? If no, switch to the best one; if yes, take the combination, store the newly found subset. In line 19, backtrack until better subsets are found. In line 20, initialize the set of evaluations of different combinations. From lines 21 to 25, repeat each possible combination, prune the p metric on node n, evaluate the combination, and find the best combination. In line 26, was a better subset of size $r - 1$ found? If yes, backtrack and store the newly found subset; if no, stop backtracking. In line 31, reached the best subset with the maximum monitoring overhead one can afford (i.e. the overhead budget)? If yes, return S_b (the set of best metrics found); if no, continue. The evaluation of the different combinations of metrics is performed from lines 34 to 39. In line 35 the monitoring overhead w is Compute from Equation 9 with Θ_k and O_k. From lines 36 to 39, can one afford the selected metrics? If

Algorithm 1: Overhead-sensitive Metrics Selection

```
// h: MIB model
// X: Metrics
// Y: Bottlenecks
// w_user: Tolerated overhead budget
// O_k: Metrics overhead
// S_b: Optimal subset
Input: X, Y, w_user, O_k
Output: S_b
begin
    Θ_k ← 0_{P,N_k}
    k ← 0
    S_b ← ∅
    while r < P × N_k do
        S_r ← ∅
        foreach {(p,n) | Θ_{k_{p,n}} = 0} do
            Θ*_k ← Θ_k
            Θ*_{k_{p,n}} ← 1
            S_r(p,n) ← evaluate(X, Y, Θ*_k)
        r ← r + 1
        (p,n) ← argmax S_r(·)
        if S_r(p,n) ≥ evaluate(X, Y, S_b(r)) then
            Θ_k ← S_b(r)
        else
            Θ_{k_{p,n}} ← 1
            S_b(r) ← Θ_k
            backtracking ← True
            while r > 2 and backtracking=True do
                S_r ← ∅
                foreach {(p,n) | Θ_{k_{p,n}} = 1} do
                    Θ*_k ← Θ_k
                    Θ*_{k_{p,n}} ← 0
                    S_r(p,n) ← evaluate(X, Y, Θ*_k)
                (p,n) ← argmax S_r(·)
                if S_r(p,n) < evaluate(X, Y, S_b(r−1)) then
                    r ← r − 1
                    Θ_{k_{p,n}} ← 0
                    S_b(r) ← Θ_k
                else backtracking ← False
        if S_b(r) = penalty then break
    return S_b

function evaluate(X, Y, Θ_k)
    Compute ω from Equation 9 with Θ_k and O_k.
    if ω_admin ≥ ω then
        s ← crossValidate(h, X[:, vec(Θ_k)], Y)
    else s ← penalty
    return s
```

yes, cross-validate the MIB model $h(\cdot)$ (see Section V-A) with the combination of metrics and return the score; if no, return a penalty score.

VI. Experimental Setup

To solve the formulated problem in a supervised learning fashion, we build a testbed to collect a training dataset. The testbed was designed to provide a training set that is representative of the real-world situation. In this Section, we offer a detailed description of the experimental testbed and the bottleneck injection campaign. We also perform an analysis of the collected multilabel dataset.

A. Testbed

We deployed on a virtualized platform a prototype implementing the experimental testbed (see Fig. 5) consists of nine

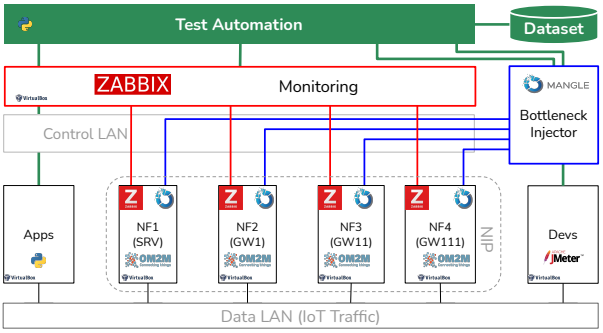

Fig. 5: Experimental Setup.

host machines: Applications (Apps) host, Devices (Devs) host, NF1 (SRV) host, NF2 (GW1) host, NF3 (GW11) host and NF4 (GW111) host. The Applications (Apps) host machine is equipped with 1 CPU, 0.5 GB RAM, 10 GB Disk space (corresponding to an AWS "T2.micro" server). The Devices (Devs) host machine is equipped with 1 CPU, 0.5 GB RAM, 8 GB Disk space (corresponding to a Raspberry Pi 1 Model B computer). The NF1 (SRV) host machine is equipped with 2 CPU, 2 GB RAM, 15 GB Disk space (corresponding to an AWS "T2.medium" server). The NF2 (GW1) host machine is equipped with 1 CPU, 1 GB RAM, 10 GB Disk space (corresponding to an AWS "T2.micro" server). The NF3 (GW11) host machine is equipped with 1 CPU, 1 GB RAM, 10 GB Disk space (corresponding to an AWS "T2.micro" server). The NF4 (GW111) host machine is equipped with 1 CPU, 0.5 GB, 8 GB Disk space RAM (corresponding to a Raspberry Pi 1 Model B computer). In order to build our testebed, the following software tools were used :

- Apache JMeter : an Apache project that can be used as a load testing tool for analyzing and measuring various services' performance, with a focus on web applications (https://jmeter.apache.org).
- Eclipse OM2M : an open-source implementation of oneM2M and SmartM2M standard for IoT services platforms initiated by LAAS-CNRS (https://www.eclipse.org/om2m).
- Zabbix : an open-source monitoring software tool for diverse IT components, including networks, servers, virtual machines (VMs), and cloud services (https://www.zabbix.com).
- VMware Mangle : a tool that enables running chaos engineering experiments seamlessly against applications and infrastructure components to assess resiliency and fault tolerance (https://vmware.github.io/mangle).

The testbed is composed of virtual machines (VMs) running on Ubuntu server 16.04. A JMeter Server is running in the Devices (Devs) host and produces the IoT workload with a request arrival rate of 20 requests per second. The considered NIP is the Eclipse open-source OM2M. The NIP nodes communicate through the Data LAN. The monitoring data are collected by the Zabbix open-source monitoring software. The bottlenecks injection and remediation are performed by VMware Mangle. The experiments are performed by an automation script (Test Automation). The *Test Automation* script

gathers and stores in the *Dataset* the monitoring data (from Zabbix) and the injected bottlenecks (from VMware Mangle). The commands and the monitoring data are sent through the Control LAN.

B. Bottlenecks Injection Campaign

Eight bottleneck types are considered and distinguished according to the NF resource they impact. They are referred to as CPU, Memory, Disk I/O, Disk space, Packet delay, Packet corruption, Packet duplicate, and Packet loss. The NFs selection probabilities follow a uniform distribution (i.e. each NF has the same probability of being selected). The injection campaign corresponds to the execution of an algorithm 2 that periodically performs bottleneck injections in NFs. An injection is defined by the targeted NF, its bottleneck type, intensity level, and duration. During a campaign, two consecutive injections are separated by μ (mean time between bottlenecks). A campaign consists of injecting all combinations of injections. An injection campaign parameters are as follows: target NFs listed in N_k, bottleneck types listed in B_t and their occurrence frequency listed in B_p, intensity levels listed in B_i, duration values listed in D_v and their selection probabilities listed in D_p. To perform the multiple bottlenecks injection, we use the following algorithm. Algorithm 2 is

Algorithm 2: Multiple Bottlenecks Injection

```
   // N_k: Set of Network Functions
   // B_t: Bottleneck Types
   // B_p: Occurrence frequency of Bottlenecks
   // B_i: Bottleneck intensities
   // D_v: Duration values
   // D_p: Probabilities of Duration
   // μ: Mean time between bottlenecks
   // B_ids: Injected bottlenecks IDs
   Input: N_k, B_t, B_p, B_v, D_v, D_p, μ
   Output: B_ids
1  begin
2      while injection do
3          b_t ← Choose a value in B_t following the distribution B_p
4          t ← Choose a value in D_v following the distribution D_p
5          n ← Choose a value in N_k following a uniform distribution
6          b_i ← B_i(b_t)
7          id ← CallMangleAPI(n, b_t, t, b_i)
8          B_ids ← Append(id)
9          Wait(μ)
10     return B_ids
```

executed by the *Test Automation script*. From lines 3 to 6, the targeted NF, its bottleneck type, its intensity level, and its duration are selected according to their associated probabilities. In line 7, the VMware Mangle component is invoked to perform the injection. In line 8, the injection information is collected and stored in the dataset. In line 9, the Algorithm waits μ time before another injection begins. Remark that the injection duration should be long enough to collect sufficient observations while short enough for the injection duration to be realistic.

Table II describes the injected bottlenecks during the campaign. The bottlenecks duration values are $\{60, 90, 120\}$. The probabilities D_p associated to the duration are $\{0.5, 0.3, 0.2\}$. The last campaign parameter μ is set to 30 seconds.

TABLE II: INJECTED BOTTLENECKS DURING THE CAMPAIGN

Name	B_p	B_i	Description
CPU	20	90%	High CPU utilization
Memory	15	90%	High Memory utilization
Disk I/O	12	5MB	High disk I/O utilization
Disk space	12	90%	High disk space utilization
Packet delay	11	200ms	High NIC usage creating additional delay
Packet duplicate	10	10%	High NIC usage creating packet duplication
Packet corrupt	10	10%	High NIC usage creating packet corruption
Packet loss	10	10%	High NIC usage creating packet loss

C. Overview of Multilabel Dataset

As presented in Fig. 6, multiple bottlenecks were injected in the considered testbed. The campaign lasts for 24h. With an observation cycle C_t set to 10 seconds, we gathered 8640 training samples. The number of collected metrics per NF $P = 26$. Over the whole testbed $P \times N_k = 104$ metrics were collected. For a complete list of the monitored metrics, see Appendix A. The number of bottlenecks is 8 per NF for a total of $B = 32$. The bottlenecks cardinality (i.e. the average number of bottlenecks per example in the dataset)is 1.960, and the bottlenecks density (the number of bottlenecks per example divided by the total number of bottlenecks, averaged over the samples) is 0.061. The bottlenecks frequency in the dataset per by NF is presented in Fig. 7.

VII. EVALUATION

A. Efficiency Criteria

Although the analysis result has multiple outcomes that can be classified into positive or negative. Such a grouping enables one to represent the comparison between a test and its reference in one 2×2 table, as depicted in Table III.

TABLE III: CONFUSION MATRIX

		True Bottlenecks	
Diagnosed Bottlenecks		True Positive (tp)	False Positive (fp)
		False Negative (fn)	True Negative (tn)

In Table III the abbreviations tp, fp, fn, and tn denote the number of respectively, true positives, false positives, false negatives, and true negatives. The term "True Positive" refers to the number of the samples correctly labeled as bottlenecks samples."True Negative" refers to the number of the samples correctly rejected as non-bottlenecks samples."False Positive" refer to the number of the samples wrongly labeled as bottlenecks samples. Finally, "False Negative" refers to the number of the samples wrongly rejected as non-bottlenecks samples. The same definitions are used throughout the paper. For each j-bottleneck the tp_j, fp_j, fn_j, and tn_j are defined as follows.

$$\text{tp}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 1 \text{ and } Y_j^i = 1) \tag{10}$$

$$\text{fp}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 1 \text{ and } Y_j^i = 0) \tag{11}$$

$$\text{fn}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 0 \text{ and } Y_j^i = 1) \tag{12}$$

$$\text{tn}_j = \sum_{i=1}^{m} 1(\hat{Y}_j^i = 0 \text{ and } Y_j^i = 0) \tag{13}$$

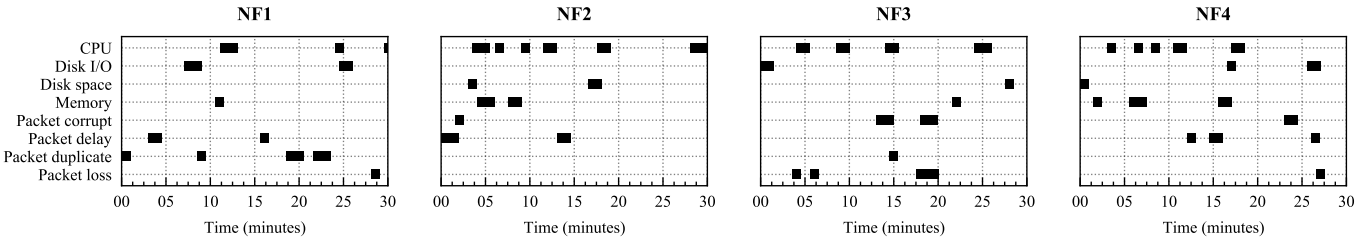

Fig. 6: Thirty-minute sample of injected bottlenecks per NF (NF1, NF2, NF3, NF4).

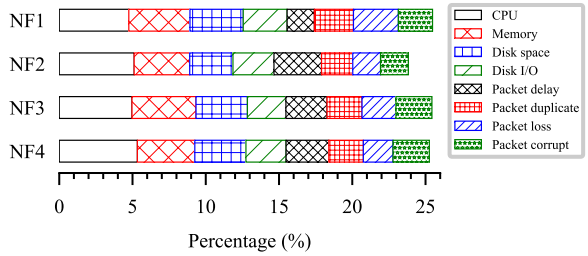

Fig. 7: Bottlenecks frequency in the dataset per by NF.

As stated in the motivation section, in this work we are interested in a MIB model that avoids false positive bottleneck. The *Subset accuracy* is not the most important criteria to consider for the proposed method efficiency. We use the positive predictive value (a.k.a precision) to indicate the probability that in the case of a positive test, that the NIP has the identified bottleneck. The ideal value of the *Precision*, with a perfect test, is 1, and the worst possible value would be 0. The average precision ($\Psi_{\text{Precision}}$) is therefore defined as follows.

$$\Psi_{\text{Precision}} = \frac{1}{B}\sum_{j=1}^{B}\frac{tp_j}{tp_j + fp_j} \tag{14}$$

Nevertheless, the *Subset accuracy*, and *Coverage Error*, are reported and discussed. The *Subset accuracy* measures the set of bottlenecks predicted for a sample that exactly matches the corresponding set of bottlenecks in Y. *Coverage Error* measures the average number of bottlenecks that have to be included in the final prediction such as all true bottlenecks are predicted. The *Coverage Error* is useful if one wants to know how many top-scored-bottlenecks the MIB model has to predict on average without missing any true one.

$$\Psi_{\text{Subset accuracy}} = \frac{1}{m}\sum_{i=1}^{m} 1(\hat{Y}^i = Y^i) \tag{15}$$

For a given prediction $\hat{Y}^i$ the estimated rank of the label j is denoted by $r_i(j)$. The most relevant label takes the top rank (1), and the last one only gets the lowest rank (B).

$$\Psi_{\text{Coverage Error}} = \frac{1}{m}\sum_{i=1}^{m}\max_{j\in Y_i} r_i(j) \tag{16}$$

Additionally, the *Sensitivity*, and *Specificity*, are reported to illustrate the performance of the classification models. *Sensitivity* measures the proportion of true positives that are correctly identified. *Specificity* measures the proportion of true negatives. Both ratios are independent of the bottleneck distribution in the dataset.

$$\Psi_{\text{Specificity}} = \frac{1}{B}\sum_{j=1}^{B}\frac{tn_j}{tn_j + fp_j} \tag{17}$$

$$\Psi_{\text{Sensitivity}} = \frac{1}{B}\sum_{j=1}^{B}\frac{tp_j}{tp_j + fn_j} \tag{18}$$

The Area Under the receiver operating characteristic Curve, or *AUC* (Ψ_{AUC}), is used in the literature to compare the performance of classifiers. The *AUC* has a crucial statistical property: the *AUC* of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative example. It is used for three specific purposes: determine the cutoff value with the highest *Sensitivity* and *Specificity*, evaluate the discriminating capacity of an analysis model, and compare the discriminative ability of different analysis models. The *AUC* is desirable for the following two reasons: *AUC* is scale-invariant (i.e. It measures how well predictions are ranked, rather than their absolute values; *AUC* is classification-threshold-invariant (i.e. It measures the quality of the model's predictions irrespective of the chosen classification threshold). In this way, the Ψ_{AUC} values are useful in our context to select the classification model to analyze the bottleneck. The best value of Ψ_{AUC} is 1, and the worst value is 0.

$$\Psi_{\text{AUC}} = \int_{x=0}^{1} \Psi_{\text{Specificity}}((1-\Psi_{\text{Sensitivity}})^{-1}(x))\,\mathrm{d}x \tag{19}$$

Below, we present the MIB and the SOMS evaluations.

B. Multiple bottlenecks identification (MIB)

There are two main approaches [40] to accomplish an MLC: problem transformation and algorithm adaptation. The former aims to produce a problem that can be processed with traditional classifiers (e.i, Single or Multiclass Classification). Conversely, the objective of the latter is to adapt existing classification algorithms to work with the MLC problem. Among the transformation methods, the most popular are those based on the MLC problem's binarization (i.e. Binary Relevance, Classifier Chain, and the Label Powerset). These transformation methods produce a multiclass problem from an MLC problem considering each label set as a class. In the algorithm adaptation approach, there are proposals of algorithms based on nearest neighbors, such as ML-kNN.

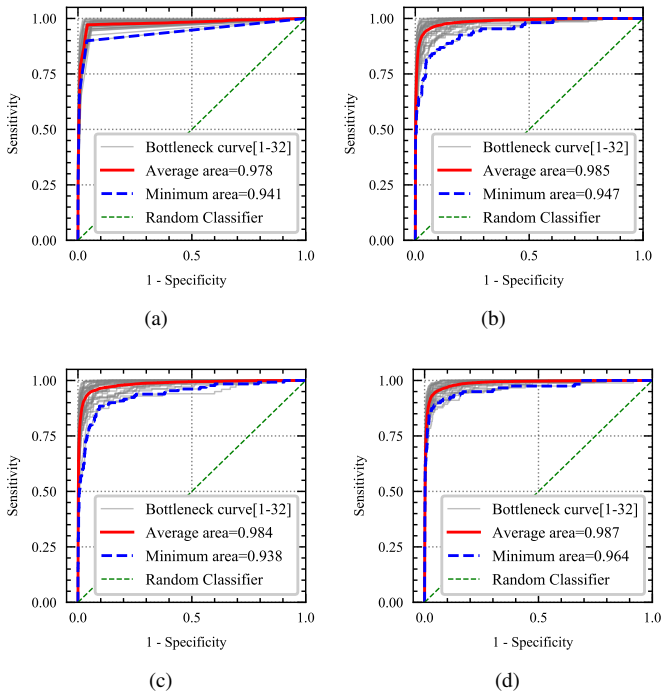

Fig. 8: MLC Receiver operating characteristic and AUC (Ψ_{AUC}). (a) ML-kNN ; (b) Binary Relevance; (c) Classifier Chain; (d) Label Powerset.

Selecting the right MLC algorithm is the next step to solve the considered problem.

We consider the ML-kNN, the Binary Relevance, the Classifier Chain, and the Label Powerset. We adopted the MLC Algorithm for the MIB model based on the Ψ_{AUC}. As in the literature, we use 75% of the collected data for training the different MIB models and 25% for the evaluations. In problem transformation algorithms (Classifier Chain, Binary Relevance, Label Powerset) a Multi-layer Perceptron is used as a base classifier.

The models were trained with scikit-multilearn [41]. In Fig. 8 (a) - (d) four curves are shown. The diagonal line (Random Classifier), shows the performance of a random guess. An intuitive example of random guessing is a decision by flipping coins. Points above the Random Classifier line represent good classification results (better than random); points below the line represent bad results (worse than random). The second curve (Minimum area) corresponds to the ROC of the smallest AUC. The third curve (Average area) corresponds to the average of ROC of all the bottlenecks The fourth curve (Bottleneck curve) presents each ROC of the bottlenecks. The best Ψ_{AUC} (average value $= 0.987$ and minimum value $= 0.964$) was obtained by Label Powerset, as shown in Fig. 8 (d). Label Powerset is a problem transformation approach that transforms a multilabel problem into a multi-class problem with one multi-class classifier trained on all unique label combinations found in the training data.

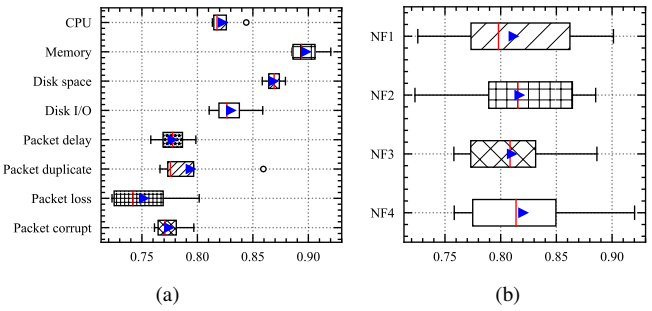

Fig. 9: Label Powerset Model Precision ($\Psi_{\text{Precision}}$). (a) Bottlenecks identification precision grouped by NF; (b) NF identification precision grouped by Bottlenecks.

TABLE IV: MULTI-LABEL CLASSIFIERS PERFORMANCE COMPARISON (WITH HYPER-PARAMETER OPTIMIZATION)

	ML-kNN	Binary Relevance	Classifier Chain	Label Powerset
Precison	0.8388	**0.8753**	0.8671	0.8253
Subset accuracy	0.5278	0.5366	0.5454	**0.6611**
Coverage Error	13.3852	12.5731	12.4852	**9.5255**
Specificity	0.9906	**0.9921**	0.9918	0.9891
Sensitivity	0.6791	0.7036	0.7048	**0.7357**

Additionally, as presented in table IV, the Label Powerset Algorithm performs better in $\Psi_{\text{Subset accuracy}}$ (0.6611), $\Psi_{\text{Coverage Error}}$ (9.5255) and $\Psi_{\text{Sensitivity}}$ (0.7357) than ML-kNN, Binary Relevance and Classifier Chain. However Binary Relevance has the higher score in $\Psi_{\text{Specificity}}$ (0.9921) and in $\Psi_{\text{Precision}}$ (0.8769). Label Powerset Algorithm will be used for validation purposes in the rest of this paper. The reader may see in [36] for further details about the Label Powerset algorithm.

In Fig. 9 we present a deeper look into the Label Powerset Algorithm performance. Fig. 9 (a) shows the bottlenecks identification *Precision* grouped by NF and Fig. 9 (b) shows the NF identification *Precision* grouped by bottlenecks type. In Fig. 9 (a), the Algorithm can identify with a minimun *Precision* > 0.81 the *Memory* bottleneck (average is 0.89 and median is 0.89), *Disk space* bottleneck (average is 0.86 and median is 0.86), *Disk I/O* bottleneck (average is 0.83 and median is 0.82), *CPU* bottleneck (average is 0.82 and median is 0.81). It can also identify with a minimun *Precision* > 0.72 *Packet duplicate* bottleneck (average is 0.79 and median is 0.77), *Packet delay* bottleneck (average is 0.77 and median is 0.77), *Packet corrupt* bottleneck (average is 0.77 and median is 0.77), *Packet loss* bottleneck (average is 0.75 and median is 0.74). From a NF perspective (see Fig. 9 (b)), the Algorithm can identify all the bottlenecks on the NF4 with an average *Precision* of 0.82 (minimun is 0.75 and median is 0.81), on NF2 with an average *Precision* of 0.81 (minimun is 0.72 and median is 0.81), on NF1 with an average *Precision* of 0.81 (minimun is 0.72 and median is 0.79), on NF3 with an average *Precision* of 0.81 (minimun is 0.75 and median is 0.80). The average *Precision* for the all bottleneck is 0.82.

In the Section below, we evaluate the Simple Overhead-sensitive Metrics Selection Algorithm in the Adaptive Performance Analysis use case described in Section III-B.

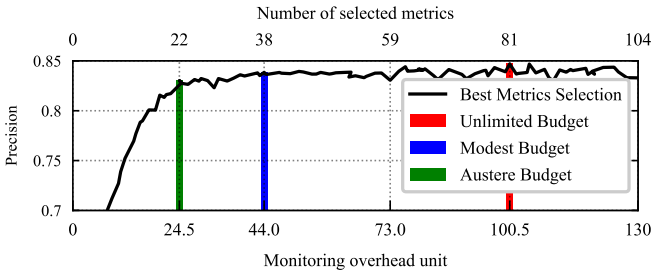

Fig. 10: SOMS Algorithm Precision.

C. Simple Overhead-sensitive Metrics Selection

In this section, we evaluate how the SOMS Algorithm finds which metrics should be considered for the efficiency of the NIP analysis while optimizing the MIB model's ability to minimize the false positive (i.e. the Precision). As stated in Section III-B, in three scenarios, the overhead budget changed in time: Unlimited overhead budget, Modest overhead budget, and Austere overhead budget. The SOMS Algorithm is implemented in Python 3, and we use 75% of the collected data for Algorithm training and 25% for the evaluation.

In Fig. 10, SOMS Algorithm removes or adds metrics at the time based on the MIB performance, until it reached all the metrics. The line (Best metric Selection) presents the progression of the *Precision* ($\Psi_{\text{Precision}}$) during the SOMS Algorithm execution. The numbers of selected metrics and the monitoring overhead are respectively shown on the first x-axis and the second x-axis. When all the metrics are selected, the *Precision* of the MIB model is 0.83. In an Unlimited Budget scenario, the maximum *Precision* is reached at 81 metrics with a monitoring overhead of $\omega_u = 100.5$. The remaining 23 metrics are irrelevant and do not increase *Precision*. The Modest Budget scenario's monitoring overhead ω_{admin} is set to 50.25, and the best subset metric compatible with this budget contains 38 metrics for a monitoring overhead of $\omega = 44$. The Austere Budget scenario's monitoring overhead ω_{admin} is set to 25.125, and the best subset metric compatible with this budget is composed of 22 metrics for a monitoring overhead of $\omega = 24.5$. The maximum precisions in the different scenarios are 0.84, 0.83, and 0.83 respectively, for the Unlimited Budget scenario, the Modest Budget scenario, and the Austere Budget scenario. Note that the *Precision* of the Unlimited Budget scenario is greater than the initial *Precision* (where all metrics are selected) of the MIB model.

We present an in-depth look at the performance associated with different scenarios. As Fig. 11 shows, in addition to the MIB model *Precision*, other criteria are considered: *Subset accuracy*, *Coverage error*, *Sensitivity*, and *Specificity*. The first criterion considered is the *Subset accuracy* ($\Psi_{\text{Subset accuracy}}$). In Fig. 11 (a) When all the metrics are selected the *Subset accuracy* is 0.65. When the best metric subset is selected in the Unlimited Budget scenario, the *Subset accuracy* is 0.66. Remark that by carefully selecting the reverent metrics, the SOMS Algorithm increases the MIB model *Subset accuracy*. In the Modest Budget and the Austere Budget scenarios, the *Subset accuracy* is 0.64. The All Metrics scenario does better than the Modest Budget scenario and Austere Budget scenario. However, with only 38 and 22 metrics respectively for the Modest Budget scenario and the Austere Budget scenario, the MIB model only loses 0.007 points of *Subset accuracy* for the Modest Budget scenario and 0.012 points for the Austere Budget scenario. In Fig. 11 (b) the different *Coverage Error* are displayed. With all the metrics, the *Coverage Error* is 9.85, while in the Unlimited Budget scenario, the *Coverage Error* is lower (9.38). In the Modest Budget scenario, the *Coverage Error* is 9.57. In the Austere Budget scenario, the *Coverage Error* is 9.65. Fig. 11 (c) the different *Sensitivity* are displayed. The *Sensitivity* when all the metrics are considered is 0.81, while when carefully selecting the reverent metrics (in the Unlimited Budget scenario), the *Sensitivity* is 0.83. In the Modest Budget and Austere Budget scenarios, the *Sensitivity* is 0.84. Fig. 11 (d) the different *Specificity* are displayed. The *Specificity* when all the metrics are considered is 0.98, while when carefully selecting the reverent metrics (in the Unlimited Budget scenario), the *Specificity* is 0.99. In the Modest Budget and Austere Budget scenarios, the *Specificity* is 0.98. We also observe here that the All Metrics scenario does better than the Modest Budget scenario and Austere Budget scenario. However, with only 38 metrics for the Modest Budget scenario and 22 metrics for the Modest Budget scenario the MIB model only loses less than 0.001 points of *Specificity* about the All Metrics scenario with 104 metrics. Concerning the trade-off found, these 3 scenarios can be relevant at certain times of the day (especially when the NIP does not have a very important workload).

D. Discussion

As earlier stated, our goal in this paper is to build an Adaptive Performance Analysis method that optimizes the

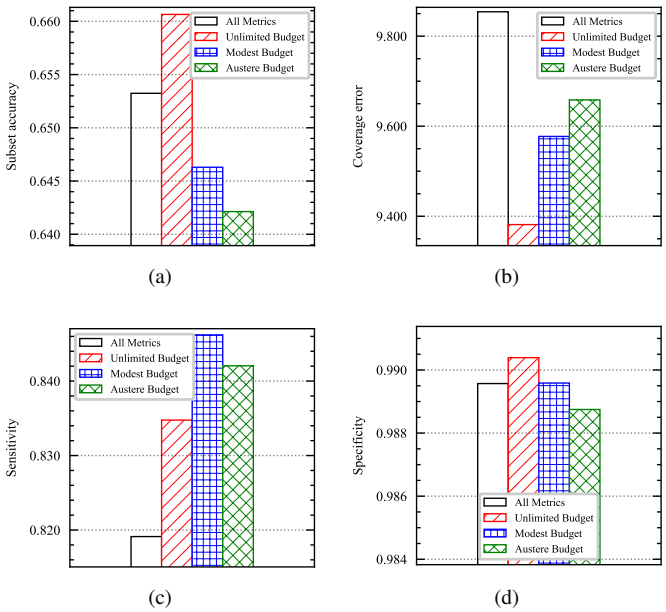

Fig. 11: Performance in different scenarios. (a) Subset accuracy; (b) Coverage error; (c) Sensitivity; (d) Specificity.

bottlenecks analysis performance regarding a monitoring overhead budget associated with the different available metrics. The proposed method relies mainly on two machine learning models: the MIB and the SOMS. The MIB model is used for the multiple bottlenecks analysis and the SOMS model is used for the metric selection optimization. Regarding the MIB model selection, we benchmark five multilabel algorithms. The results show that the compared algorithms demonstrate good performance. However, the Label Powerset, outperformed in *Coverage Error*, showing that on average, we need to go down to the 9th bottlenecks (ranked) to cover all the relevant bottlenecks of the sample. Hence the *Subset Accuracy* and the *Sensitivity* results justify using Label Powerset as a base algorithm for the MIB model.

To achieve the metric selection regarding a monitoring overhead budget, we have proposed SOMS (a feature selection heuristic). SOMS optimizes the MIB model *Precision*. By analyzing the results, we observe that the *Precision* criterion is not sufficient to decide on the choice of metrics in the different scenarios. Indeed, other criteria such as the *Subset accuracy*, the *Coverage error*, the *Sensitivity*, and the *Specificity* are important to take into account to choose adaptively (in time) the best subset of metrics (see Fig. 11). The proposed method exhibited high performances for the considered use case in the presence of different bottleneck types. The SOMS algorithm determines the metrics that maximize the efficiency of the analysis and have a minimum overhead compatible with an allocated overhead budget. In the classic scheme (i.e. "All metrics") which consists in recovering all the monitoring metrics (without seeking to minimize the monitoring overhead), we observe underperformance on all the considered criteria (*Subset accuracy*, the *Coverage error*, the *Sensitivity*, and the *Specificity*). This underperformance is explained by the presence of metrics that act by their lack of correlation with the bottlenecks as noise on the performance of the classification model (MIB). In NIPs where the scarcity of resources is exacerbated at the edge of the network, the main drawback of this approach is that it is not adapted. To remedy this drawback, a second scheme (dynamically implemented with the SOMS algorithm) is to do it with a variable budget over time, which takes into account variations in the use of the NIP resources from one moment to another. This scheme is broken down into 3 scenarios. The first scenario ("Unlimited Budget") studied in this paper is to minimize the cost of monitoring overhead, without taking into account the notion of budget. In this scenario, we try to maximize the efficiency of the diagnosis, while ruling out irrelevant metrics, as defined in Section V of the paper. We then observe the best results on the *Subset accuracy*, the *Coverage error*, and the *Specificity*). By extending and continuing this scenario in scenarios two and three, respectively "Modest Budget" and "Austere Budget" we seek to determine if it is possible to maximize the performance of the MIB under a fixed monitoring budget. We observe in the modest budget scenario and the austere budget scenario that a trade-off can be found between the budget that we set for the monitoring and the performance of the MIB. Compared to the classic scheme, the advantage of SOMS is that it allows adaptive monitoring to be carried out (spread over a day) according to the resources of the NIP, here expressed in terms of monitoring budget.

Nevertheless, our general approach shares all supervised learning algorithms' intrinsic limitations regarding the need to have a representative and complete training dataset to make a useful analysis. Accordingly, the method is likely to be less efficient if an unknown bottleneck occurs during operation. This problem can be mitigated by frequently re-training the models (MIB and SOMS) with the data collected continuously from the NIP.

The computational complexity of Label Powerset is upper bounded by $\mathcal{O}(\min(m,\ 2^B))$, but is usually much smaller in practice [42]. The SOMS Algorithm computational complexity is upper bounded by $\mathcal{O}(2^{P\times N_k})$ [43].

According to the previous experimental results, it is possible to conclude that our approach gives useful information, to make decisions about the NIP bottlenecks, to improve the QoS.

VIII. Conclusions

We have proposed in this paper a new overhead-sensitive approach for multiple bottleneck identification in NIPs. This approach combines a multilabel classification algorithm (Label Powerset) and a metrics selection algorithm called SOMS (Simple Overhead-sensitive Metrics Selection). We considered the specific and challenging case of the NFV-enabled IoT Platforms (NIPs), where de facto heterogeneity is stressed by the emerging context of the recent networking technologies for routing and connectivity, the computation infrastructure for processing and storage, and the varying constraints of data producers and consumers' devices. We considered the case of the horizontal NIPs that increase the heterogeneity by addressing the cross-domain interoperability. We implemented our approach on top of OM2M, the reference implementation of the international standard oneM2M [1]. We showed by emulating different scenarios where the overhead budget varies. Using all the platform metrics may increase the model's generalization error by keeping irrelevant features or noise.

We hope this study provides useful insights into how one can adaptively analyze performance bottlenecks in NIPs (i.e determine the right metric subset to collect) while efficiently controlling the induced monitoring overhead. The first line of future research is to investigate this approach in other contexts such as Blockchain-based IoT platforms (e.g. [44]), where the constraints are not related directly to resource scarcity but to resource consumption. The second line of future research would be to formulate a multi-objective problem to take into account multiple criteria in the SOMS algorithm. It would also be interesting to extend this method to consider a hybrid approach combining supervised and unsupervised learning algorithms (e.g., based on the clustering of observations like in our previous work in [45]), and take advantage of the benefits of each of these distinct algorithms while mitigating their weaknesses to identify known bottleneck as well as an unknown bottleneck. Finally, considering the injected bottleneck types investigated in our experiments, it is assumed that they are representative of the manifestation of a large set of bottlenecks located in the NFs. We still need to assess the representativeness of such bottleneck types.

APPENDIX

List of the 26 monitored metrics per NF (From the official OS Linux Template of Zabbix).

/: Free inodes in %	/: Space utilization
/: Used space	/boot: Free inodes in %
/boot: Space utilization	/boot: Used space
Available memory	Available memory in %
CPU idle time	CPU iowait time
CPU softirq time	CPU system time
CPU user time	CPU utilization
Context switches per second	Free swap space
Free swap space in %	Interface enp0s8: Bits received
Interface enp0s8: Bits sent	Interrupts per second
Load average (15m avg)	Load average (1m avg)
Load average (5m avg)	Memory utilization
Number of processes	Number of running processes

REFERENCES

[1] oneM2M, "Technical specificationts-0002-v2.7.1: Requirements," *oneM2M*, vol. 1, pp. 1–24, 2016.

[2] ETSI, "Network functions virtualisation (nfv); architectural framework. 2014," *Group Specification*, 2014.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[4] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[5] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in IoT: A systematic mapping," *J. Syst. Softw.*, vol. 132, pp. 186–203, 2017.

[6] B. Gregg, *Systems performance: enterprise and the cloud*. Pearson Education, 2013.

[7] S. Malkowski, M. Hedwig, and C. Pu, "Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 118–127.

[8] J. Qiu, Q. Du, Y. He, Y. Q. Lin, J. Zhu, and K. Yin, "Performance anomaly detection models of virtual machines for network function virtualization infrastructure with machine learning," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 2018, vol. 11140 LNCS, pp. 479–488.

[9] F. Schmidt, F. Suri-Payer, A. Gulenko, M. Wallschlager, A. Acker, and O. Kao, "Unsupervised anomaly event detection for VNF service monitoring using multivariate online arima," *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 2018-December, pp. 278–283, 2018.

[10] Z. Li, B. Y. Zhao, H. Zheng, Z. Ge, A. Mahimkar, J. Wang, J. Emmons, and L. Ogden, "Predictive analysis in network function virtualization," *Proc. ACM SIGCOMM Internet Meas. Conf. IMC*, pp. 161–167, 2018.

[11] G. Yu, Z. Cai, S. Wang, H. Chen, F. Liu, and A. Liu, "Unsupervised Online Anomaly Detection with Parameter Adaptation for KPI Abrupt Changes," *IEEE Trans. Netw. Serv. Manag.*, vol. PP, no. c, p. 1, 2019.

[12] J. Waller, "Performance benchmarking of application monitoring frameworks," *Softwaretechnik-trends*, vol. 35, 2014.

[13] T. Wang, J. Xu, W. Zhang, Z. Gu, and H. Zhong, "Self-adaptive cloud monitoring with online anomaly detection," *Future Generation Computer Systems*, vol. 80, pp. 89 – 101, 2018.

[14] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates, "G-RCA: A generic root cause analysis platform for service quality management in large IP networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1734–1747, 2012.

[15] J. Weng, J. H. Wang, J. Yang, and Y. Yang, "Root Cause Analysis of Anomalies of Multitier Services in Public Clouds," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1646–1659, 2018.

[16] H. Zhou, Y. Li, H. Yang, J. Jia, and W. Li, "BigRoots: An Effective Approach for Root-Cause Analysis of Stragglers in Big Data System," *IEEE Access*, vol. 6, pp. 41 966–41 977, 2018.

[17] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," *arXiv preprint arXiv:1701.08546*, 2017.

[18] C. Sauvanaud, K. Lazri, M. Kaaniche, and K. Kanoun, "Anomaly Detection and Root Cause Localization in Virtual Network Functions," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 196–206, 2016.

[19] C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. Da Silva Silvestre, "Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned," *J. Syst. Softw.*, vol. 139, pp. 84–106, 2018.

[20] J. M. N. Gonzalez, J. A. Jimenez, J. C. D. Lopez, and H. A. Parada, "Root Cause Analysis of Network Failures Using Machine Learning and Summarization Techniques," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 126–131, 2017.

[21] Y. Cui, J. Shi, and Z. Wang, "Fault propagation reasoning and diagnosis for computer networks using cyclic temporal constraint network model," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 47, no. 8, pp. 1965–1978, aug 2017.

[22] D. Cotroneo, R. Natella, and S. Rosiello, "A Fault Correlation Approach to Detect Performance Anomalies in Virtual Network Function Chains," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, vol. 2017-October, pp. 90–100, 2017.

[23] D. Cotroneo, L. De Simone, and R. Natella, "NFV-bench: A dependability benchmark for network function virtualization systems," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 934–948, 2017.

[24] ——, "Dependability Certification Guidelines for NFVIs through Fault Injection," *Proc. - 29th IEEE Int. Symp. Softw. Reliab. Eng. Work. ISSREW 2018*, pp. 321–328, 2018.

[25] L. Zhang, X. Zhu, S. Zhao, and D. Xu, "A novel virtual network fault diagnosis method based on long short-term memory neural networks," *IEEE Veh. Technol. Conf.*, vol. 2017-Septe, pp. 1–5, 2018.

[26] L. Mariani, C. Monni, M. Pezze, O. Riganelli, and R. Xin, "Localizing Faults in Cloud Systems," *Proc. - 2018 IEEE 11th Int. Conf. Softw. Testing, Verif. Validation, ICST 2018*, pp. 262–273, 2018.

[27] R. J. Pfitscher, A. S. Jacobs, L. Zembruzki, R. L. dos Santos, E. J. Scheid, M. F. Franco, A. Schaeffer-Filho, and L. Z. Granville, "Guiltiness: A practical approach for quantifying virtual network functions performance," *Comput. Networks*, vol. 161, pp. 14–31, 2019.

[28] B. Tola, G. Nencioni, and B. E. Helvik, "Network-Aware Availability Modeling of an End-to-End NFV-Enabled Service," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 4, pp. 1389–1403, 2019.

[29] H. Bouattour, Y. B. Slimen, M. Mechteri, and H. Biallach, "Root Cause Analysis of Noisy Neighbors in a Virtualized Infrastructure," *IEEE Wirel. Commun. Netw. Conf. WCNC*, vol. 2020-May, pp. 10–15, 2020.

[30] C. A. Ouedraogo, , S. Medjiah, C. Chassot, K. Drira, and J. Aguilar, "A cost-effective approach for end-to-end qos management in nfv-enabled iot platforms," *IEEE Internet of Things Journal*, 2020.

[31] J. Kephart, J. Kephart, D. Chess, C. Boutilier, R. Das, J. O. Kephart, and W. E. Walsh, "An architectural blueprint for autonomic computing," *IBM White paper*, pp. 2–10, 2003.

[32] D. Battré, M. Hovestadt, B. Lohrmann, A. Stanik, and D. Warneke, "Detecting bottlenecks in parallel dag-based data flow programs," in *2010 3rd Workshop on Many-Task Computing on Grids and Supercomputers*. IEEE, 2010, pp. 1–10.

[33] S. Malkowski, M. Hedwig, D. Jayasinghe, J. Park, Y. Kanemasa, and C. Pu, "A new perspective on experimental analysis of n-tier systems: Evaluating database scalability, multi-bottlenecks, and economical operation," in *2009 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2009, pp. 1–10.

[34] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 121–129.

[35] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. Ieee, 2015, pp. 1200–1205.

[36] G. Tsoumakas, I. Katakis, and I. Vlahavas, *Mining Multi-label Data*. Boston, MA: Springer US, 2010, pp. 667–685. [Online]. Available: https://doi.org/10.1007/978-0-387-09823-4_34

[37] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM journal on computing*, vol. 24, no. 2, pp. 227–234, 1995.

[38] J. Reunanen, *Search Strategies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 119–136.

[39] R. Kohavi, G. H. John *et al.*, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[40] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 8, pp. 1819–1837, 2013.

[41] P. Szymański and T. Kajdanowicz, "A scikit-based Python environment for performing multi-label classification," *ArXiv e-prints*, Feb. 2017.

[42] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Random k-labelsets for multilabel classification," *IEEE transactions on knowledge and data engineering*, vol. 23, no. 7, pp. 1079–1089, 2010.

[43] J. Doak, "An evaluation of feature selection methods and their application to computer security," 1992.

[44] Y. Jiang, Y. Zhong, and X. Ge, "Iiot data sharing based on blockchain: A multi-leader multi-follower stackelberg game approach," *IEEE Internet of Things Journal*, 2021.

[45] L. Morales, C. A. Ouedraogo, J. Aguilar, C. Chassot, S. Medjiah, and K. Drira, "Experimental comparison of the diagnostic capabilities of classification and clustering algorithms for the qos management in an autonomic iot platform," *Service Oriented Computing and Applications*, vol. 13, no. 3, pp. 199–219, 2019.

Clovis Anicet Ouedraogo received his Ph.D. degree and M.Sc. degree in computer science from the Institut National des Sciences Appliquées de Toulouse, France, in 2017 and 2021, respectively. He also received the Engineering degree in networks and telecommunications from the École Nationale des Sciences Appliquées de Safi, Morocco, in 2016. He is currently a Data scientist at Equativ (formerly Smart), France. His main research include Autonomic computing, Optimization and Machine Learning.

Samir Medjiah received a Ph.D. (2012) in computer science from the University of Bordeaux, France. He is an associate professor at Paul Sabatier University in Toulouse (France) and a research scientist at the Laboratory for Analysis and Architecture of Systems (LAAS-CNRS). His main research interests include application-network co-optimization, software-defined networking, network virtualization, M2M communications, and IoT applications. He is actively working on R&D projects related to M2M/IoT.

Christophe Chassot received the engineering degree and the MS degree (DEA) in computer science from the National Polytechnic Institute of Toulouse (INPT) in 1992, and the Ph.D. degree in computer science from INPT in 1995. He is a full professor at the National Institute of Applied Sciences of Toulouse (INSA) and the deputy director of the Electrical engineering and Computer Science Department of the INSA. His primary teaching topics deal with computer networks at basic and advanced levels. He is also an associate researcher at LAAS-CNRS. His primary fields of interest include service-oriented and component-based autonomic transport services/protocols and end-to-end signaling architectures for self-adaptive management of QoS in heterogeneous networks.

Khalil Drira received the Master degree in computer science from INP, Toulouse, in 1988, and the Ph.D. and HDR degrees in computer science from Universite Paul Sabatier Toulouse in 1992 and 2005, respectively. Since 1992, he assumes a full-time research position in CNRS, France. His research interests include cooperative network IoT services, platforms and applications. His research activity addresses topics in this field focusing on Software architectures and communication services. He continues to be involved in national and international conferences and journals. He serves as a member of the program journals in the fields of software architecture as well as IoT and Internet networks. He has also been an Editor of several proceedings, books, and journals.

Jose Aguilar received the title of Systems Engineer from the Universidad de Los Andes, Venezuela, in 1987, the M.Sc. degree in computer science from the Université Paul Sabatier, France, in 1991, and the Ph.D. degree in computer science from the Universit'e Rene Descartes, France, in 1995. He completed Postdoctoral studies with the Department of Computer Science, University of Houston, from 1999 to 2000, and the Laboratoire d'analyse et d'architecture des systèmes (LAAS)", CNRS, Toulouse, France, from 2010 to 2011. He is currently a Full Professor with the CEMISID, Escuela de Ingeniería de Sistemas, Universidad de Los Andes, Mérida, Venezuela. He has published more than 500 articles and ten books in the field of parallel and distributed computing, computer intelligence, and science and technology management. His research interests include artificial intelligence, semantic mining, big data, emerging computing, and intelligent environments. He is a member of the Mérida Science Academy and the IEEE CIS Technical Committee on Neural Networks.